31422 02 pp. 011-034 r3ah.ps 2/9/05 5:21 PM Page 11

2

The Baseline Personal Process

If you are an experienced software developer, you probably wonder why you need a process. After all, you have written many programs and know what to do. How can a process help? A process can help in several ways.

First, most of us would like to improve the way we do our jobs. To do this, however, we must know how we work and where we could do better. Just like Maurice Greene, the runner described in Chapter 1, there is a big difference between generally knowing what you do and having a precisely defined and measured process. With a defined process, you can easily determine where you are having problems and identify ways to improve.

Second, even if you don't plan to improve the way you work, you will almost certainly have to commit to dates for completing your assigned tasks. To responsibly do this, however, you will need a plan. As you will see in the next few chapters, if you don't have data on your prior performance, it is almost impossible to make an accurate plan. And, as you will also see, to get the data you must have a precise definition of how you work. That is a process.

Finally, even if you don't need data, you will likely work at one time or another on large or complex jobs. Here, where the job steps are not obvious and where you must coordinate your work with others, you must agree with your coworkers on how to do the work. Regardless of what you choose to call it, the definition of how you plan to work is a process.

Therefore, whatever the name or format, you need a process. What you may not yet appreciate is that a defined, measured, and planned process will help you to make accurate plans, work predictably, and produce high-quality products. This book shows you how.

2.1 What Is a Process?

A process is the sequence of steps required to do a job, whether that process guides a medical procedure, a military operation, or the development or maintenance of software. The process definition is the description of the process. When properly designed, a software process definition will guide you in doing your work. When team members all follow different processes, or more commonly, where they use no defined processes at all, it is like a ball team with some members playing soccer, some baseball, and others football. On such teams, even the best individual players make a losing team. In contrast, when all team members follow a well-structured and properly defined process, they can better plan their work, coordinate their activities, and support each other.

The software process establishes the technical and management framework for applying methods, tools, and people to the software task. The process definition identifies roles, specifies tasks, establishes measures, and provides exit and entry criteria for the major steps. When properly used, a defined process also helps to ensure that every work item is properly assigned and that its status is consistently tracked. In addition, defined processes provide orderly mechanisms for learning. As better methods are found, they can be incorporated into the process definition. This helps all teams learn and it shows new teams how to take advantage of the experiences of their predecessors.

In summary, a defined process identifies a job's principal steps:

- It helps you separate routine from complex activities.
- It establishes the criteria for starting and finishing each process step.
- It enhances process understanding and provides a sound basis for process automation.

A defined process also includes measures:

- These measures can help you to understand your performance.
- They help you and your team manage your work.
- The process measures also help you to plan and manage the quality of the products you produce.

A defined process provides a sound basis for project management:

- You can make detailed plans and precisely measure and report your status.
- You can judge the accuracy of your estimates and plans.
- You can communicate precisely with users, other developers, managers, and customers about the work.

A defined process provides a solid foundation for process management and improvement:

- The process data help you identify the process steps that cause the most trouble.
- This will help you improve your personal performance.
- The process will simplify training and facilitate personnel mobility.
- Well-designed process definitions can be reused or modified to make new and improved processes.

2.2 Defining Your Own Process

Because every project and every team is different, when you work on a TSP team you will define your own processes. Rather than starting from scratch, however, you will have a family of proven TSP processes to build on. You will also have the experience gained from using the PSP processes in this book. The first step in preparing you to be on a TSP team is learning how to use an efficient personal process like the PSP. This chapter defines the initial process you will use for the first exercise in this book. In subsequent chapters, this process will be enhanced with new techniques and methods.

When you define your personal process, abstract tasks become structured and subject to analysis. Until you define the tasks in sufficient detail, however, you will not be able to improve. For example, it is hard to reason about a vaguely defined design process. Once you separate the design work into defined steps, however, you can understand the various elements of design work and see how they relate. To use a sports analogy, when people start weight lifting they think about building their arm, leg, or torso muscles. However, more experienced bodybuilders work on specific muscles: the deltoids, pecs, and abs. The clearer the focus, the better they can improve in each area. A precise process will help you to focus on the most promising areas for personal improvement.

2.3 Baseline Process Contents

The principal objective of PSP0, the baseline process, is to provide a framework for writing your first PSP program and for gathering data on your work. The data from the first PSP programs provide a comparative baseline for determining the impact of the PSP methods on your work. The PSP0 process is shown in simplified form in Figure 2.1. The scripts guide you through the process steps, the logs help you to record process data, and the plan summary provides a convenient way to record and report your results.

The PSP0 process provides the following benefits:

- A convenient structure for performing small-scale tasks
- A framework for measuring these tasks
- A foundation for process improvement

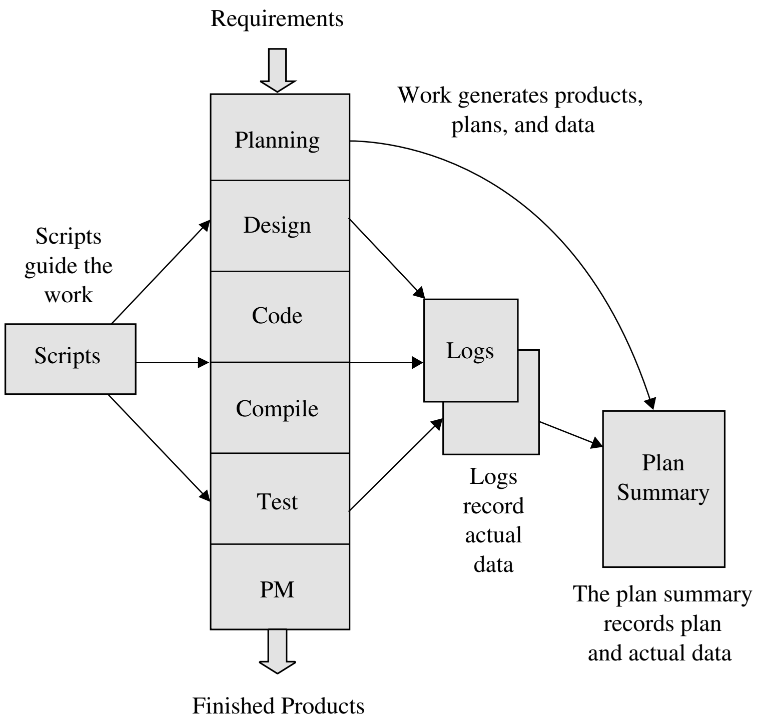

FIGURE 2.1 PSP0 PROCESS FLOW

Convenient Working Structure

When you perform any but the most trivial task, you always face the question of how to attack it. What do you do first, second, and so on? Even when you do this task planning properly, it takes time. With a defined process like PSP0, you can save task-planning time because you already determined how to do the tasks when you developed the process. When you do similar tasks many times, it is more efficient to develop and document the process once, rather than stop to devise a new process each time you start a job.

The PSP has been tested with many users, so there is no question that it works. In this book, you will use these predefined processes as a way to gain the knowledge and experience needed to later define the processes that best fit your needs. Although a defined process may sound restrictive, it is not. In fact, if you find your personal process inconvenient, uncomfortable, or not helpful, change it. Don't try to change the process until you have finished the PSP course, but after that it is yours and yours alone to manage. Then, when your process doesn't work for you, change it until it does.

Measurement Framework

A defined process establishes the process measures. This enables you to gather data on the time you spend on each task, the sizes of the products you produce, and the numbers of defects you inject and remove in each process step. These data help you to analyze your process, to understand its strengths and weaknesses, and to improve it. A defined process also gives measurements explicit meaning. When you define the coding phase, for example, the time spent in coding or the number of defects injected during coding are clear and specific values with precise definitions. This precision, however, requires that each process phase have explicit entry and exit criteria. With the baseline PSP0 process, some of the entry and exit criteria are not very precise. This is because PSP0 is a simple process. In later chapters, with progressively refined processes, these process steps are better defined.

Foundation for Improvement

If you don't know what you are doing, it is hard to improve the way you do it. Most software professionals would argue that they know what they do when developing software. However, they can rarely describe their actions in detail. When you follow a defined process, you are more aware of how you work. You observe your own behavior and can often see how to break each larger step into smaller elements. PSP0 is a first step in building this more refined understanding.

2.4 Why Forms Are Helpful

The PSP uses several forms. Although you might initially view this as a disadvantage, you will find the forms to be an enormous help. Consider, for example, how forms could help in planning your work. Any reasonably challenging job generally involves the following series of steps:

1. Determine what is to be done.
2. Decide how to do it.
3. Do it.
4. Check to make sure it is correct.
5. Fix any problems.
6. Deliver the final result.

Even for relatively simple tasks, the preparation and checking steps can take a significant amount of time. Suppose, for example, you were asked to produce a plan for your next development job. If you had never made such a plan, it would take some time to figure out what the plan should contain and how to produce it. Then you would have to decide on the plan format. Finally, you would make the plan. When you finished, you would still have to make several checks. First, did you leave out anything important? Second, was the information correct? Finally, are the format and content appropriate?

Suppose, instead, that you have a planning form. Now you don't need to decide what data to provide; the form tells you. All you do is fill in the blanks. The form may even provide guidance on how to do the required calculations. Then, when you are through, you should check to ensure you did the work properly. With a form, this is easy. You just ensure that all of the spaces contain data. Properly designed forms improve your efficiency and help you produce complete and correct results.

Just like a process, however, good forms are hard to develop and they must be periodically reviewed and revised to ensure that they still meet your needs. It is also important to ensure that all the process forms are designed as a coherent whole and supported by a well-designed tool. When forms are not carefully designed, they may call for duplicate information, terminology may be inconsistent, or formats may have confusing variations. This can cause inefficiency and error.

2.5 The PSP Process Elements

The overall PSP0 process is shown in Figure 2.1 (page 14). In the planning step, you produce a plan to do the work. Next are the four development steps: design, code, compile, and test. At the end, in the postmortem step, you compare your actual performance with the plan and produce a summary report. Although these planning and postmortem steps may not seem necessary when you are writing small programs, they are needed to gather the data to manage and improve your personal process. If the program is so simple that planning seems pointless, then making the plan should be a trivial task. However, these “trivial” programming jobs often hold surprises that a plan could have helped you anticipate. Also, as you gain experience, you will find that the postmortem is an ideal time to think about your data and to see where and how to improve.

A process may be simple or complex. Large defined processes, for example, will have many elements or phases, each of which can also be viewed as a defined process. At the lowest level, the process phases are composed of steps that have no further defined substructure. In a large process, for example, each step in the overall script would be a phase. At the next level, each phase would itself have a process definition with the steps being lower-level tasks. In the case of PSP0, the planning, development, and postmortem phases have process definitions, but design, code, compile, and test are named as unrefined steps. When a process element has a definition and a structure, I call it a phase. When it has no defined structure, I call it a step or a task.

The Planning Phase

A planning form guides you in producing and documenting your plan, and it provides a consistent format for the results. Although your initial PSP0 plans will not be very complex, they will provide some initial planning experience and data.

The Postmortem Phase

The postmortem phase is an important part of every process. When you have completed even a small project, you have a large amount of potentially useful data. Unless you promptly record and analyze the data, you probably will not remember precisely what you did or be able to correct any data-gathering errors. Even if you wait only a week to produce the summary report, you probably will waste time checking the data and trying to remember what you did and why. By learning and practicing the planning, reporting, analysis, and postmortem steps with these PSP exercises, you will build project planning and management skills. Experience with these small exercises will help you to develop the habits you will need to plan, track, and manage the larger-scale work you will do later on a TSP team.

2.6 The PSP0 Process

The PSP scripts guide you through the process steps. The principal script elements are its purpose, the entry criteria, the phases (or steps) to be performed, and the exit criteria. The PSP0 Process Script is shown in Table 2.1. It describes in words the simple process structure shown in Figure 2.1. A second PSP0 script, the Planning Script, is shown in Table 2.2. It briefly summarizes the planning steps for PSP0. With a simple process like PSP0, you may not need to look at the script very often, but with a more complex process, it is a good idea to use the script much like a checklist. Then, at the beginning and end of every process phase, you can verify that you didn't overlook anything important.

The planning and postmortem phases are quite clear from the scripts in Tables 2.1, 2.2, and 2.4, but the development phase in Table 2.3 has four steps: design, code, compile, and test. Until these steps have been explicitly described,

TABLE 2.1 PSP0 PROCESS SCRIPT

Purpose		To guide the development of module-level programs
Entry Criteria		• Problem description • PSP0 Project Plan Summary form • Time and Defect Recording logs • Defect Type standard • Stopwatch (optional)
Step	**Activities**	**Description**
1	Planning	• Produce or obtain a requirements statement. • Estimate the required development time. • Enter the plan data in the Project Plan Summary form. • Complete the Time Recording log.
2	Development	• Design the program. • Implement the design. • Compile the program, and fix and log all defects found. • Test the program, and fix and log all defects found. • Complete the Time Recording log.
3	Postmortem	Complete the Project Plan Summary form with actual time, defect, and size data.
Exit Criteria		• A thoroughly tested program • Completed Project Plan Summary form with estimated and actual data • Completed Time and Defect Recording logs

TABLE 2.2 PSP0 PLANNING SCRIPT

Purpose		To guide the PSP planning process
Entry Criteria		• Problem description • Project Plan Summary form • Time Recording log
Step	**Activities**	**Description**
1	Program Requirements	• Produce or obtain a requirements statement for the program. • Ensure that the requirements statement is clear and unambiguous. • Resolve any questions.
2	Resource Estimate	Make your best estimate of the time required to develop this program.
Exit Criteria		• Documented requirements statement • Completed Project Plan Summary form with estimated development time data • Completed Time Recording log

TABLE 2.3 PSP0 DEVELOPMENT SCRIPT

Purpose		To guide the development of small programs
Entry Criteria		• Requirements statement • Project Plan Summary form with estimated program development time • Time and Defect Recording logs • Defect Type standard
Step	**Activities**	**Description**
1	Design	• Review the requirements and produce a design to meet them. • Record in the Defect Recording log any requirements defects found. • Record time in the Time Recording log.
2	Code	• Implement the design. • Record in the Defect Recording log any requirements or design defects found. • Record time in the Time Recording log.
3	Compile	• Compile the program until there are no compile errors. • Fix all defects found. • Record defects in the Defect Recording log. • Record time in the Time Recording log.
4	Test	• Test until all tests run without error. • Fix all defects found. • Record defects in the Defect Recording log. • Record time in the Time Recording log.
Exit Criteria		• A thoroughly tested program • Completed Time and Defect Recording logs

TABLE 2.4 PSP0 POSTMORTEM SCRIPT

Purpose		To guide the PSP postmortem process
Entry Criteria		• Problem description and requirements statement • Project Plan Summary form with development time data • Completed Time and Defect Recording logs • A tested and running program
Step	**Activities**	**Description**
1	Defect Recording	• Review the Project Plan Summary to verify that all of the defects found in each phase were recorded. • Using your best recollection, record any omitted defects.
2	Defect Data Consistency	• Check that the data on every defect in the Defect Recording log are accurate and complete. • Verify that the numbers of defects injected and removed per phase are reasonable and correct. • Using your best recollection, correct any missing or incorrect defect data.
3	Time	• Review the completed Time Recording log for errors or omissions. • Using your best recollection, correct any missing or incomplete time data.
Exit Criteria		• A thoroughly tested program • Completed Project Plan Summary form • Completed Time and Defect Recording logs

there is no way to tell when each step starts or ends. One common confusion, for example, concerns the distinction between coding and compiling. While the compiler is first run, that is clearly compile time, but how do you classify the time spent making the necessary coding changes to fix the defects found during compilation? In this book, I suggest that you classify the time spent correcting compile defects as compile time, and the time spent correcting and compiling test defects as test time. Similarly, the exit criteria call for a thoroughly tested program. Thorough testing should be defined for each program during planning, requirements, or design. Until all of the entry and exit criteria are precisely defined, however, the measurements for these steps are imprecise. To learn how to define and measure a process, I have found it most effective to start with imprecise measurements and then use the resulting data to make the process and its measures more precise.

2.7 PSP0 Measures

PSP0 has two measures:

1. The time spent per phase
2. The defects found per phase

The time spent per phase is a simple record of the clock time spent in each part of the PSP process. Although recording time and defect data can take a little time, several PSP support tools are available. When using one of these support tools, time recording is simple and takes little time. Time recording is discussed in Section 2.8, and PSP support tools are covered at the end of the chapter.

Recording defect data is a little trickier. For PSP0, record the specified data for every defect you find during compiling and testing. A defect is counted every time you change a program to fix a problem. The change could be one character, or it could be multiple statements. As long as the changes pertain to the same compile or test problem, they constitute one defect. Note that you determine the defect count by what you change in the program. Some compilers, for example, generate multiple error messages for a single defect. If these were all connected to one problem, then you would count it as one defect.

Also count defects in the test cases and other materials but treat each test case as a separate program and record defects against it. Defect recording is discussed in Section 2.9. Record all the defects found and fixed in a program against the new code written for that program. When you work on a team, you will often be enhancing larger programs, so you should separately record the defects found in the older base program against that program and not against the new code you are developing. In general, don't record any defects from any other programs, test procedures, or support systems against your new code. Doing so would make it difficult to assess the quality of your work.

The reason for gathering both time and defect data is to help you plan and manage your projects. These data will show where you spend your time and where you inject and fix the most defects. The data will also help you to see how your performance changes as you modify your process. You can then decide for yourself how each process change affects your productivity and the quality of your work products.

2.8 Time Recording

Table 2.5 shows the PSP Time Recording Log and Table 2.6 shows the Time Recording Log Instructions. Even though the time logs in various PSP support tools may look somewhat different, they all record the same basic data:

- The project or program being worked on
- The process phase for the task
- The date and time you started and finished working on the task
- Any interruption time

- □ The net or delta time worked on the task
- □ Comments

Before using any PSP support tool, you must enter your name, the project, and any other information required by the tool. When you start an activity, such as planning your first PSP program, enter the date and the time you start. Then, when you finish planning, enter the time you finished. The tool will automatically calculate the difference between the start and stop times in the *Delta Time* column. If you were interrupted during this phase, record the interruption time in the *Interruption Time* column.

With activities that take several hours or more, you will often be interrupted by phone calls, questions, or other things. If you consistently ignore the time such interruptions take, you will not know how much time you actually spent on any activity. Though you might think that the interruption times are so small that you could ignore them, they often turn out to be as long or longer than the actual time spent on the task itself. Unfortunately, with most interruptions, it is impossible to tell at the outset how long they will take. You could note in the *Comments* column when the interruption started and then calculate the lost time when the interruption was over. For an 18-minute interruption, for example, you would enter 18 in the *Interruption Time* column. Then the PSP support tool would subtract 18 minutes from the elapsed time at the end of the task.

Often, you will forget to record the start or stop times of a phase or an interruption. When you realize this, make your best estimate of the time involved. If you do this promptly, it is likely to be fairly accurate. I handle interruptions by keeping a stopwatch at hand and starting it when I am interrupted. I often work at home and my wife initially objected to being "on the clock." She has gotten used

TABLE 2.5 THE PSP TIME RECORDING LOG

Student	____________________	Date	__________
Program	____________________	Program #	__________
Instructor	____________________	Language	__________

Project	**Phase**	**Start Date and Time**	**Int. Time**	**Stop Date and Time**	**Delta Time**	**Comments**

TABLE 2.6 TIME RECORDING LOG INSTRUCTIONS

Purpose	Use this form to record the time you spend on each project activity. For the PSP, phases often have only one activity; larger projects usually have multiple activities in a single process phase. These data are used to complete the Project Plan Summary. Keep separate logs for each program.
General	Record all of the time you spend on the project. Record the time in minutes. Be as accurate as possible. If you need additional space, use another copy of the form. If you forget to record the starting, stopping, or interruption time for an activity, promptly enter your best estimate.
Header	Enter your name and the date. Enter the program name and number. Enter the instructor's name and the programming language you are using.
Project	Enter the program name or number.
Phase	Enter the name of the phase for the activity you worked on, e.g., Planning, Design, Test.
Start Date and Time	Enter the date and time when you start working on a process activity.
Interruption Time	Record any interruption time that was not spent on the process activity. If you have several interruptions, enter their total time. You may enter the reason for the interrupt in comments.
Stop Date and Time	Enter the date and time when you stop working on that process activity.
Delta Time	Enter the clock time you actually spent working on the process activity, less the interruption time.
Comments	Enter any other pertinent comments that might later remind you of any unusual circumstances regarding this activity.

to it, however, and I have found it very convenient. It is actually easier to start and stop a stopwatch than to make notations in the Time Log. Some support tools have a built-in "stopwatch" function for this purpose. Again, when you occasionally forget to start or stop the stopwatch, make your best estimate as soon as you remember.

By keeping an accurate Time Log, you will discover that you are frequently interrupted. Because each interruption breaks your train of thought and is a potential source of error, reducing interruption frequency and duration can pay big dividends. Having such data helps you understand the interruption problem and figure out ways to address it. In one case, a TSP team member who used the PSP had a cubicle right beside the copy center and people would stop to chat with her when they were waiting for an available machine. Her interruption time records helped her justify a request to move to a different cubicle.

A common time-recording problem is deciding when a process phase starts or ends. With compile, for example, the time you enter for compile is the time you spend getting the first clean compile. As you find and fix defects in test, however, you will recompile each fix. Enter this time as part of the test phase and not as part of compile. Compile ends with the first clean compile. Even if you are designing as part of fixing a defect or you are struggling with an implementation problem, record your time in the phase in which you found and fixed the problem.

Although you must use judgment in deciding the phase you are in, the basic rule is to enter what seems most appropriate to you. With some interpretative and fourth-generation systems, this can be confusing, particularly when there is no compile step. With these systems, skip the compile phase and go directly from code to test. I discuss compile more completely in Section 2.11, and cover incremental development issues in Section 2.12.

2.9 Defect Recording

An example Defect Recording Log and its instructions are shown in Tables 2.7 and 2.8. When you first encounter a defect and decide to fix it, start counting the fix time. Once you have fixed the defect, enter all of the data for that defect. Most tools will automatically enter a unique defect number, but you will have to enter the defect type, the phase in which the defect was injected, the phase in which it was removed, the fix time, and a brief description of the defect. If you don't know any of these time or phase data, enter your best estimate. The *Remove* phase is the one in which you found the defect. If, for some reason, you fix a defect in a phase other than the one in which you found it, enter the phase in which it was *found* and note in the *Description* section where it was *fixed.* Also, if you make a mistake fixing one defect and later find and fix that new defect, note the number of the defective fix in the *Fix Ref.* column. With a little practice, you will usually be able to enter the data for one defect in a minute or less. The example data in Table 2.7 show how one developer completed his Defect Log for PSP exercise Program 1.

For defect type, use the number from the Defect Type Standard shown in Table 2.9. Although this standard is quite simple, it should be sufficient to cover most of your needs. If you prefer to use a different standard, you can develop your own, but wait until after you have completed PSP training and have enough data

TABLE 2.7 A DEFECT RECORDING LOG EXAMPLE

Defect Types

10 Documentation	**60 Checking**
20 Syntax	**70 Data**
30 Build, Package	**80 Function**
40 Assignment	**90 System**
50 Interface	**100 Environment**

Student	Student 3	Date	1/19
Program	Standard Deviation	Program #	1
Instructor	Humphrey	Language	C

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.
1	1/19	1	20	Code	Comp.	1	

Description: Missing semicolon.

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.
		2	20	Code	Comp.	1	

Description: Missing semicolon.

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.
		3	40	Des.	Comp.	1	

Description: Wrong type on RHS of binary operator, must cast integers as float.

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.
		4	40	Code	Comp.	1	

Description: Wrong type on RHS, constant should be 0.0.

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.
		5	40	Code	Comp.	1	

Description: Wrong type on RHS, had to cast an integer as float.

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.
		6	40	Des.	Comp.	7	

Description: Exponent must be integer, sqrt. Integral not correct.

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.
		7	80	Code	Test	1	

Description: Std. dev. incorrect, subtract when should divide.

TABLE 2.8 PSP DEFECT RECORDING LOG INSTRUCTIONS

Purpose	Use this form to hold data on the defects you find and correct. These data are used to complete the project plan summary.
General	Record each defect separately and completely. If you need additional space, use another copy of the form.
Header	Enter your name and the date. Enter the program name and number. Enter the instructor's name and the programming language you are using.
Project	Give each program a different name or number. For example, record test program defects against the test program.
Date	Enter the date you found the defect.
Number	Enter the defect number. For each program or module, use a sequential number starting with 1 (or 001, etc.).
Type	Enter the defect type from the defect type list summarized in the top left corner of the form. Use your best judgment in selecting which type applies.
Inject	Enter the phase when this defect was injected. Use your best judgment.
Remove	Enter the phase during which you fixed the defect. This will generally be the phase when you found the defect.
Fix Time	Enter the time you took to find and fix the defect. This time can be determined by stopwatch or by judgment.
Fix Ref.	If you or someone else injected this defect while fixing another defect, record the number of the improperly fixed defect. If you cannot identify the defect number, enter an X.
Description	Write a succinct description of the defect that is clear enough to later remind you about the error and help you to remember why you made it.

to do a competent job. In addition, keep the number of defect types small until you understand your personal defect data and can be explicit about the types that most concern you. When you make a new standard, do it with care because changing the defect standard can invalidate your historical data.

TABLE 2.9 PSP DEFECT TYPE STANDARD

Type Number	Type Name	Description
10	Documentation	Comments, messages
20	Syntax	Spelling, punctuation, typos, instruction formats
30	Build, Package	Change management, library, version control
40	Assignment	Declaration, duplicate names, scope, limits
50	Interface	Procedure calls and references, I/O, user formats
60	Checking	Error messages, inadequate checks
70	Data	Structure, content
80	Function	Logic, pointers, loops, recursion, computation, function defects
90	System	Configuration, timing, memory
100	Environment	Design, compile, test, or other support system problems

Fix Time

Recording fix time can be a bit more of a problem. You will likely fix most compile defects in about a minute, so you can generally enter 1 under *Fix Time.* In some cases, however, you may think you can fix a problem in a minute but find that it takes a lot longer. If you do not use a stopwatch to track fix time, enter your best estimate. In test, however, it is easy to lose track of the time spent fixing each defect. In this case, it is generally wise to use a stopwatch or to record the clock time when you start and stop working on each fix. Some PSP support tools provide a timer for this purpose.

The Multiple-Defect Problem

One common problem concerns multiple defects: While fixing one defect, you encounter and fix another. In this case, separately record each defect and its fix time. For example, if you spent a few minutes fixing defect 18 while working on a more complex defect (defect 17), deduct the fix time for defect 18 from the total fix time for defect 17. Figure 2.2 illustrates this example for the data shown in Table 2.10. These defects were fixed in the test phase so they are all test defects:

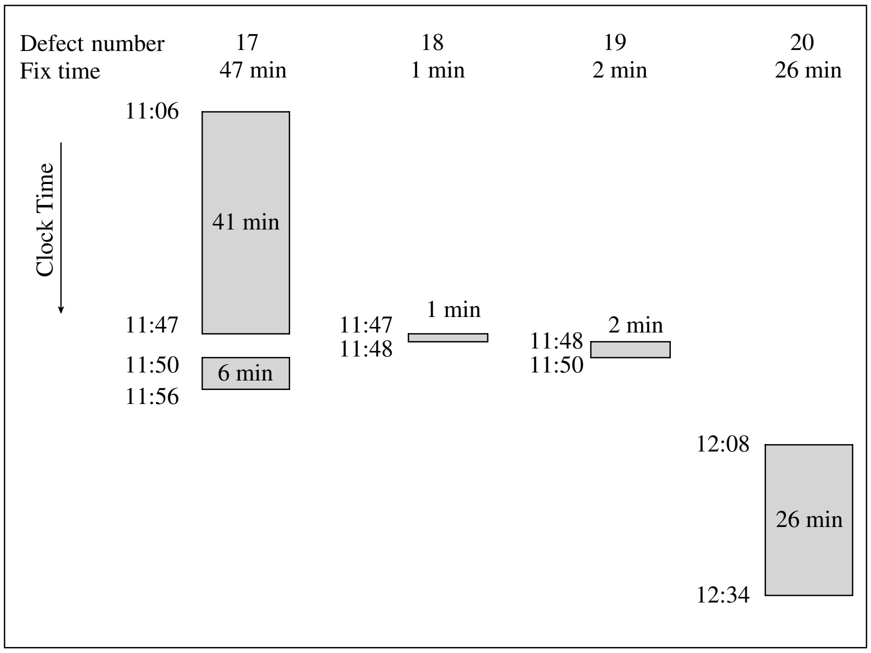

FIGURE 2.2 A DEFECT FIX TIME EXAMPLE

- Defect 17 is a logic defect you encountered in testing at 11:06. After 41 minutes, you had a fix ready to compile.
- While compiling the fix for defect 17, you encountered defect 18—a typographical error you made while fixing defect 17. You found it at 11:47 and fixed it in 1 minute.
- While fixing defect 18, you noticed defect 19 and started to fix it at 11:48. It was a wrong name that had been introduced in coding. It took 2 minutes to fix.
- After fixing defect 19, you spent 6 more minutes compiling and testing the fix to defect 17 before you were convinced you had fixed it. Your total elapsed time was 50 minutes but the actual fix time for defect 17 was only 47 minutes.
- A little later in testing, you found defect 20. It is a logic error that you injected while fixing defect 17. It took 26 minutes to fix.

The total elapsed time for all of this fix activity was 76 minutes, with 12 minutes of intervening test time between the end of defect 17 at 11:56 and the start of defect 20 at 12:08. You presumably stopped the clock on defect 17 at 50 minutes

TABLE 2.10 A MULTIPLE-DEFECT EXAMPLE

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.
1	1/10	17	80	Des.	Test	47	

Description: StepPointer advanced extra step when CycleCount loop exceeded limit

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.
		18	20	Test	Test	1	17

Description: Misspelled Step-counter when fixing 17

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.
		19	50	Code	Test	2	

Description: Noticed SummaryFile improperly referenced

Project	Date	Number	Type	Inject	Remove	Fix Time	Fix Ref.
		20	80	Test	Test	26	17

Description: Forgot to adjust SyncPointer when fixing 17

when you thought you had it fixed. However, subsequent testing showed that you introduced defect 20 while fixing defect 17. This defect took another 26 minutes to fix. Here, defects 18 and 20 are both noted in the Defect Recording Log in Table 2.10 as injected during testing. The fact that these were fix errors is noted with an entry in the *Fix Ref.* space. You will find this information helpful when you later analyze your data. Only the times for defects 18 and 19 are deducted from the elapsed time of 50 minutes to arrive at the fix time of 47 minutes for defect 17. Your Defect Recording Log now has four new defects, as shown in Table 2.10.

These defect data are only used by you (and later by your TSP team) to decide where and how to improve your process. Although numerical precision is not critical, the fix times help identify the time saved by your process changes. Fix times for identical defect types are generally five to ten or more times longer in the test phases than earlier in the process. These fix time data will demonstrate the value of removing defects early in the process. They will also help you evaluate the cost and schedule consequences of your personal process improvements.

In the *Description* spaces, describe each defect in enough detail so you can figure out how to prevent similar defects in the future or how to find them earlier in the process. A simple description, usually just a few words, is generally adequate. For a more complex defect, however, you may have to write a more complete description. Make this description clear enough that, even weeks later, you can decide how that defect could have been prevented or found earlier. Because you will use the data to help improve your process, you need good records.

Record the data on each defect when you fix it. If you don't, you probably won't remember it later. In any but the smallest programs, there are generally so many defects that they are easily confused. Once you join a TSP team, your defect data will help you and your teammates to consistently produce products that have few if any defects left after customer delivery. You will also save many hours, weeks, or even months of test time. To get these test-time savings, however, you must have formed good defect-recording habits during PSP training and you must continue recording and analyzing your defects when you subsequently work on a TSP team.

2.10 The PSP0 Project Plan Summary

The PSP0 Project Plan Summary form is shown in Table 2.11, and its instructions are provided in Table 2.12. If you properly log all of your time and defect data, most PSP support tools will automatically enter the actual values in the Project Plan Summary form. You will, of course, have to enter the plan data yourself.

In planning Program 1, use whatever method you can to estimate how much total time you think it will take to write the program. If you don't have a better way, you will have to guess. Enter this number in the support tool as the total planned time for Program 1. Because you don't yet have data on the distribution of your development time by phase, you cannot allocate your total time among the PSP phases for the first program.

During the postmortem, examine the data in the completed Project Plan Summary form. Although the support tool will have entered the actual time and defect data for you, you must ensure that these data are correct. For example, if the Plan Summary showed that you injected more defects than you removed, check the Defect Log to find and fix the problem. Similarly, if the times for any phases look wrong, check the Time Log to find the mistake and fix it. Example data for a completed Project Plan Summary form are shown in Table 2.11.

The *To Date* column in the Project Plan Summary form holds the actual time and defect data for the programs you have written for the PSP course. For example, Student 3 spent 30 minutes in designing Program 1. Because this was his first program, he also entered 30 in the *To Date* column for design. With Program 2, if he spent 25 minutes in design, the actual design entry would be 25, but the *To Date* design entry would be 55 (30 + 25). The defect *To Date* entries are made in the same way.

The *To Date %* column shows the distribution of the *To Date* data among the phases. For example, Student 3 injected 2 defects in design and 5 during coding for a total of 7. Because 2 is 28.6% of 7, he entered 28.6 in the *To Date %* column for actual defects injected in design. The other *To Date %* entries are calculated in the same way.

31422 02 pp. 011-034 r3ah.ps 2/9/05 5:21 PM Page 31

TABLE 2.11 EXAMPLE OF THE PSP0 PROJECT PLAN SUMMARY

Student	Student 3	Date	1/19
Program	Standard Deviation	Program #	1
Instructor	Humphrey	Language	C

Time in Phase (min.)	**Plan**	**Actual**	**To Date**	**To Date %**
Planning		5	5	4.3
Design		30	30	25.6
Code		32	32	27.4
Compile		15	15	12.8
Test		5	5	4.3
Postmortem		30	30	25.6
Total	180	117	117	100.0

Defects Injected	**Actual**	**To Date**	**To Date %**
Planning	0	0	0.0
Design	2	2	28.6
Code	5	5	71.4
Compile	0	0	0.0
Test	0	0	0.0
Total Development	7	7	100.0

Defects Removed	**Actual**	**To Date**	**To Date %**
Planning	0	0	0.0
Design	0	0	0.0
Code	0	0	0.0
Compile	6	6	85.7
Test	1	1	14.3
Total Development	7	7	100.0
After Development	0	0	

2.11 The Compile Phase

The compile phase generally causes a lot of debate. First, if you are using a development environment that does not compile, then you should merely skip the compile step. If, as is usual, you have a build step, however, you can record the build time and any build errors under the compile phase. However, in my experience, the build activity is generally so short and contains so few build defects that the resulting data are not very useful. As a result, at least with VB.NET, I recorded my build time as well as my build defects under test. If you would like a separate record of these data, you could record them under compile.

For those environments that have a compile step, the compile data are extremely useful and I urge you to gather complete compile time and defect data.

TABLE 2.12 PSP0 PLAN SUMMARY INSTRUCTIONS

Purpose	To hold the plan and actual data for programs or program parts
General	"To-Date" is the total actual to-date values for all products developed.
Header	Enter your name and the date. Enter the program name and number. Enter the instructor's name and the programming language you are using.
Time in Phase	Enter the estimated total time. Enter the actual time by phase and the total time. To Date: Enter the sum of the actual times for this program plus the To-Date times from the most recently developed program. To Date %: Enter the percentage of To-Date time in each phase.
Defects Injected	Enter the actual defects by phase and the total actual defects. To Date: Enter the sum of the actual defects injected by phase and the To-Date values for the most recent previously developed program. To Date %: Enter the percentage of the To-Date defects injected by phase.
Defects Removed	To Date: Enter the actual defects removed by phase plus the To-Date values for the most recent previously developed program. To Date %: Enter the percentage of the To-Date defects removed by phase. After development, record any defects subsequently found during program testing, use, reuse, or modification.

Some developers feel that the compile defects are unimportant and not worth recording. We now have data on thousands of programs written with the PSP and it is clear that those developers who gathered and analyzed all of their defect data were more productive and produced much higher quality products than those who did not. Therefore, do your best to gather complete defect data during PSP training. It will not take much time and the results will be worth the effort.

2.12 Incremental Development

There are many styles and methods for writing programs. Although some people will argue that one method is better than another, you are the only person who can make that determination. Unfortunately, most developers make such decisions based solely on their intuition and what other people tell them. With the PSP, however, you will have the data to determine which methods most help you to efficiently produce quality programs for predictable costs and on committed schedules.

Some developers, when they take the PSP course, argue that the only way they can develop programs is to write a few lines of code and then compile and test them to make sure that they run. Then they write another couple of lines and compile the combined old and new code and run it. They follow this incremental approach until they have a complete running program. Although there are many potential problems with this approach, no development approach is entirely problem-free.

Unfortunately, this particular method of developing programs requires a different process than the one presented with the PSP. Although there are many ways you could define a process to work this way, these processes are generally difficult to teach. The problem is not that the processes are inefficient or bad, but they are hard to measure and the resulting process data are difficult to interpret.

You could use whatever method you prefer for writing the PSP programs and merely record the small increments of time for each step. For example, if you spent three minutes writing two lines of code, a minute compiling the code, and another three minutes testing it, you could record your time that way. Then, with the next few lines, another four minutes of coding would bring your coding time up to seven minutes, and so forth. You could then accumulate your time in this way until you completed the program. However, doing this has two disadvantages. First, some of the PSP methods introduced a little later will be more difficult to handle with this type of process. Second, your data will not be comparable to that of the many thousands of developers who have already taken the PSP course. This will make it more difficult for your instructor to assess your work and to advise you on how best to improve.

In Chapter 13, I discuss various example processes you might want to try out after you complete the PSP course. One of these is the PSP3 process, which follows an iterative development strategy. However, PSP3 presumes that your iterations are each somewhat larger than a few lines of code. I also discuss the process I used in learning VB.NET. Here, I was using a very large and complex environment with many canned functions that I did not know. I found it most convenient to use a prototyping process that first produced running code. I then produced and reviewed the program's design. This process enabled me to experiment with the various functions to find the ones that best fit my needs without worrying about recording all the defects normally associated with learning. Once I found something that seemed to work properly, however, I did record all the subsequent defects I found.

During the PSP course, try to follow the process given in the scripts. Then, after you have learned the PSP, feel free to modify the process in whatever way suits you. However, as you do, gather data on your work and use that data to guide you in deciding on the best processes and methods to use.

2.13 PSP Tool Support

When I originally developed and taught the PSP, no support tools were available, so we had to manually gather the data with paper forms, make the required analysis calculations by hand or on a spreadsheet, and store the data on paper records or a personal data repository. Although this manual process was inconvenient and error prone, it took surprisingly little time. Now, with the available PSP support tools, data recording is relatively painless and the required process calculations are done automatically. For information on the available PSP support tools, see **www.sei.cmu.edu/tsp/psp**.

2.14 Summary

A defined process can help you to understand your performance as a software developer and to see where and how to improve. This chapter introduces the PSP0 process, which you will use to complete the initial programming assignment in the text.

With the PSP0 process, you measure the time spent in each development phase and the defects you inject, find, and fix. You will also use a support tool for defect and time recording. Then you will use these data in later chapters to make better plans and to improve the quality and productivity of your work.

2.15 Exercises

The standard assignment for this chapter uses PSP0 to write one or more programs. The assignment kits for these programs can be obtained from your instructor or at **www.sei.cmu.edu/tsp/psp**. The PSP assignment kits contain the specifications for the exercise programs and the PSP process scripts. In completing the assignments, faithfully follow the specified PSP process, record all required data, and produce a program report according to the specification given in the assignment kit.