13

Process Extensions

As developers gain experience, they usually develop a practiced and comfortable personal working style. Although this is normal, it can cause problems. The trouble is that the best style for a particular job depends on many factors that are more important than your personal preferences. The kind of application being developed, the tools used, and the practices of the other team members can all affect the best way to do a job. One factor that should have great impact on a developer's programming style is the scale of the product being developed. Methods that work well for small jobs are often inadequate for larger ones.

Although the practices that work reasonably well for small programs are often not appropriate for large programs, many of the practices that are effective for large programming jobs are equally effective for small ones. Examples are personal quality management, planning, and data gathering. As the scale of a project increases, additional practices are needed. Examples are team inspections, team planning, and team coordination. These practices are needed as soon as you switch from working alone to working with a team. These process changes are dictated by the need to do cooperative work instead of working alone.

Although some level of design coordination is usually helpful on small team projects, small-scale design problems are generally fairly simple, and rudimentary design practices are often adequate. The problem is that design work takes time, and when experienced developers write small programs, they often find that the time required to produce a documented design is largely wasted. These designs

are generally small and simple enough for one person to completely understand. That is why, for example, many software developers spend very little time producing designs and much more time writing code. Because they intuitively understand the design, the first time they document the design is during the coding. However, by not using well-practiced and sound design methods when developing small programs, developers often have design problems when they work on large-scale systems.

Because we all learned to program by writing small programs, the practices we learned as students often stick with us, even when they are no longer appropriate. To appreciate the problem, consider Figure 13.1. Here, the code-intensive developers spent relatively little time in design. For every hour of coding time, they spent less than 15 minutes in design work. Conversely, the design-intensive developers spent at least as much time in design as in coding. As shown in the figure, on average, the designers took longer than the coders to develop the PSP exercise programs.

However, this is not the entire story. One of the most important lessons of the PSP is that a little time properly invested early in the process can save a great deal of time later. Code reviews are one example. Even though reviews take time, that time is generally made up almost immediately by reduced compiling and testing time. The much greater later savings in system testing and use are an added bonus. For design, the long-term savings are not as clear. One indication of the benefits of design work is shown in Figure 13.2: the designers generally produced smaller

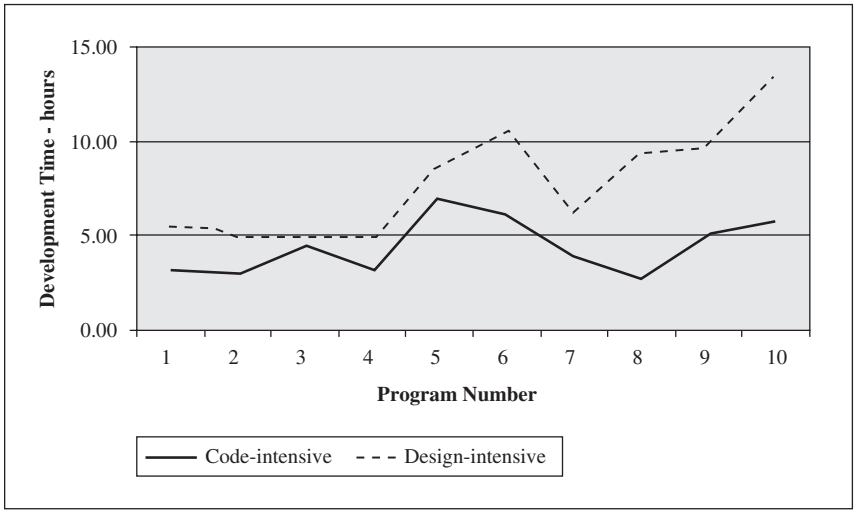

FIGURE 13.1 DEVELOPMENT TIME VERSUS DESIGN TIME (810 DEVELOPERS)

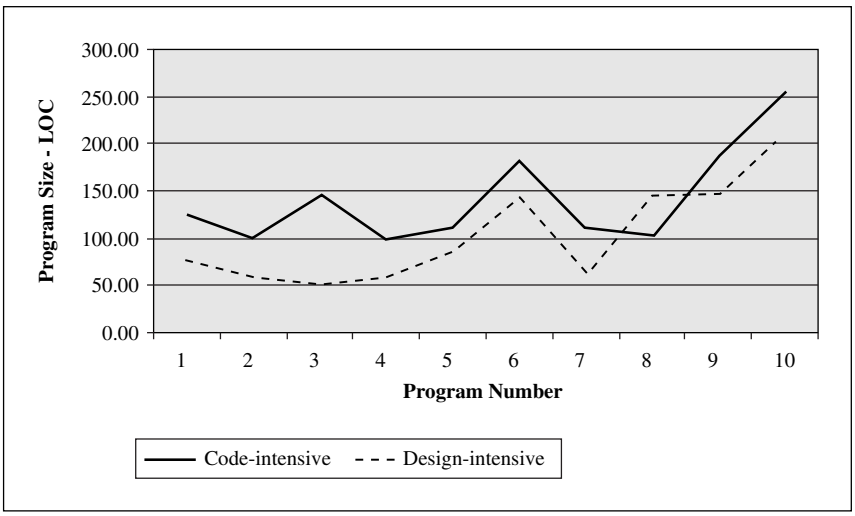

FIGURE 13.2 PROGRAM SIZE WITH AND WITHOUT DESIGN (810 DEVELOPERS)

programs than the coders. Although design work pays, the fact that it doesn't pay off right away makes it much more difficult for developers to take the time to produce sound and precise designs. The PSP has you produce designs not because you need them for the class exercises but because you need to practice sound design methods with small programs if you are ever to use them with large programs.

13.1 Customizing the Development Process

There are many ways to write programs, and some are more effective than others. The key is to use a process that both works for you and produces the desired results. Because every process has advantages and disadvantages, you should use different processes for different situations. When you write programs in only one way, you cannot be uniformly effective with all of your work. For example, in developing applications for secure environments, you must produce designs, regardless of how long it takes. Security is a design issue and it is almost impossible to find security flaws by searching through source code. If, as with the PSP exercises, the problems are so simple that a design is not usually needed to write the code, development time will generally be less when you just bang out code. Conversely,

when security or safety is important, when small program size is critical, or when you are developing large or complex systems, a design is needed, regardless of how long it takes. Therefore, you need at least two defined and practiced personal processes: one that works when you need to produce a design and another one for when you do not.

The principal challenge addressed by this chapter is how to change your processes as you work on progressively larger and more complex systems. Although there is no general answer, a well-defined and practiced personal process will enable you to answer these questions for yourself. This chapter discusses some of the principles and strategies for dealing with these process development issues, both when developing new processes and when enhancing existing ones. Starting with a discussion of the PSP process strategy, the chapter covers process development concepts and a suggested way to define personal and team processes. At the end, the chapter describes four example processes.

13.2 Why Define a Process?

There are two reasons to define a process. First, you need a process to provide the data and the framework for planning, tracking, and managing your work. Second, you need a process to guide the work while you are doing it.

To appreciate the importance of the first reason, consider the six PSP principles discussed in Chapter 8 (see p. 157). They explain why a defined process is essential for doing quality work and for planning, tracking, and managing that work. Without a defined process, you can't specify or interpret the measures needed for planning and managing the work. Once you agree with the need for a defined process, you next need to identify the processes required for both your work and that of your team. To do this, consider your personal preferences, the type of work to be done, and the working environment. In addition, because most of us work with rapidly changing technology, consider the kinds of work you expect to do in the future.

The sizes of the software products that you develop in the future will probably be much larger than those you develop today. The software in even simple products will soon be larger than individuals can intuitively master. Although individual developers will still develop software components, their work will increasingly involve large teams, and even groups of teams. A process that was optimum for your last project will probably not be the optimum one to use for a job that is five to ten times larger. As the sizes of the products you develop increase, your process should scale up as well. Although you should not use a more sophisticated process than you need, you should tailor your personal and team processes to fit your current project.

Processes are useful for guiding such personal activities as writing the PSP exercise programs in this book, and they are even more helpful for larger projects and for working on development teams. Just as in sports, development teams are most effective when all of the team members have common goals and they all understand and agree on the strategy, plan, and process for doing the work.

13.3 The PSP Process Strategy

Because software development is intellectual work and because our skills and abilities are dynamic and varied, there is no canned process that everyone will find convenient and usable. This is particularly important for complex system design. Such creative work would be seriously inhibited by an inconvenient or annoying process. The objective, therefore, is for development teams to define their own processes and to change these processes whenever they see better ways to do the work.

Conversely, it would be difficult, if not impossible, for anyone to define a usable personal process before they had actually used one. That is why this book begins by having you use the predefined PSP processes. Then, after you have used these processes and seen how they work, you will be better able to define processes for yourself and your team.

Once you have used the PSP to develop several small programs, you will want to expand and adapt it to meet your project's needs. For example, if you plan to develop a large system, you may decide that these PSP processes are not adequate and want to adapt the PSP to your particular environment. You could even develop a new process to perform totally different tasks such as writing reports, building databases, conducting tests, developing requirements, or redeveloping existing programs. Although there is no cookbook method that fits all process development situations, this chapter provides some process definition guidelines and some customized processes to guide you in your own process development work.

13.4 Defining a Process

So far in this book, you have used processes without worrying about where they came from, but processes must be developed. Even for a simple PSP process, this development can take a fair amount of work. Although process definition is relatively straightforward, a structured process-definition process can help you to do process development work properly and to be more productive.

Process Elements

A personal process contains the following items:

- Scripts that describe how the process is enacted and that refer you to pertinent standards, forms, guidelines, and measures.
- Forms that provide a convenient and consistent framework for gathering and retaining data. These forms specify the data required and where to record them. When the process is supported by the development environment, the forms will generally be included in the tool.
- Standards that guide your work and provide a basis for verifying product and process quality. The PSP provides example coding and defect type standards as well as design review and code review checklists. Although these are adequate for the example programs in this book, enhance them or develop new ones to fit your particular needs.
- Process improvement. The Process Improvement Proposal (PIP) is a process defect-reporting procedure. It is part of the process improvement process. Include a similar procedure in every process that you develop. A multiperson process should also include means for receiving, evaluating, and responding to PIPs as well as a periodic procedure for assessing and updating the process.

With the TSP, the process that you and your team use belongs to you. You and your teammates should work together to change it to best fit your evolving needs.

Process Definition

Define a software process in much the way that you develop software products: start with the users' needs and end with final testing and release. For a personal process, the normal negotiation and support activities of most software projects are unnecessary. Therefore, the steps for defining a personal process are as follows:

- Determine your needs and priorities.
- Define the process objectives, goals, and quality criteria.
- Characterize your current process.
- Characterize your target process.
- Establish a process development strategy.
- Define your initial process.
- Validate your initial process.
- Enhance your process.

Although these steps are listed in a specific order, they need not be done in that order. It is likely, for example, that the needs and goals steps will affect each other, as will many of the other steps. Do them in whatever order is most appropriate for your situation; just make sure that you address all of the steps.

Determining Process Needs and Goals

The purpose of a software development process is to produce software products. These products must, in turn, satisfy users' needs. Hence, a process definition should consider both the product and the process objectives. To do a complete job of addressing these needs, however, you should follow an orderly procedure. TSP teams start with a launch process that walks them through such a procedure for defining their processes, so this chapter focuses on defining a personal process. The process-definition principles are the same for both cases, however, so this discussion will also provide useful background for defining team processes. The TSP launch is described in Chapter 14.

For a personal process, the objective is to identify the improvements and enhancements that will help you to do better work. If you already know the changes you want to make, list them, and then make a plan for the work. If you merely know that you'd like to make some changes but don't know which ones to make or how to make them, start by using the process you already have and fill out a PIP form whenever you find something you'd like to change. For any process, the principal items to define are as follows:

- The process purpose or objective
- Entry criteria
- Any general guidelines, usage considerations, or constraints
- The process steps
- Process measures
- Quality criteria
- Exit conditions

Before defining any process, you must know the purpose of the process, how it is to be used, and the entry criteria. You should also know the exit criteria and understand the quality objectives and measures. After you have defined these items, you are ready to start the process-definition work.

13.5 Process Evolution

It is practically impossible to produce a usable process the first time you try. The problem is that the work of defining a process changes that process. This concept is best explained by considering the five processes shown in Figure 13.3. The perceived process is what you think you do. If you have not defined your process, you will have an idealized perception of it. In any complex activity, we often know what we should do. In doing it, however, we usually encounter complications. You may be rushed and skip a review, or you may defer a design update. If someone were to ask you what you did, you would probably not mention all of these "exceptions."

The perceived process is what you would like to believe you do, whereas the actual process is what you really do, with all its omissions, mistakes, and oversights. The official process is what you are supposed to do. That process may be out-of-date or totally impractical, but it is what management would describe as your process. The target process is your objective—the ideal that you would like to reach.

When you first define your process, the objective should be an initial process that both represents reality and moves you in the direction of your target process. Generally, the initial process will be fairly close to your perceived process, but when you try to use this process, the shortcuts and omissions will stand out. This realization will change what you do. You will recognize areas that lack discipline and resolve to do better the next time. When you adjust the first definition, you will again define what you should do, rather than what you actually do. After repeating this cycle several times, your process and your actions should converge on a realistic initial process.

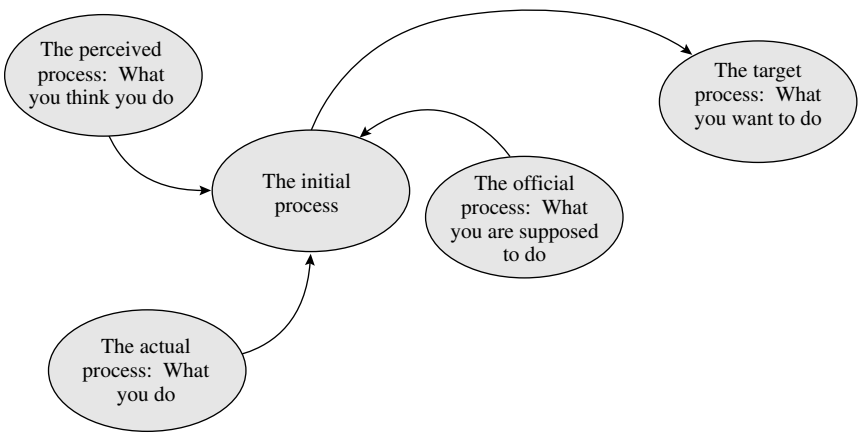

FIGURE 13.3 THE FIVE PROCESSES

Although the initial process will not likely be the same as either the original perceived or actual processes, it should be a more disciplined and stable starting point. This is one reason why defined processes are of higher quality than undefined ones. The act of defining them makes the oversights and omissions more obvious. This increases process fidelity and improves performance. The principal reason why our processes are so poor is that we often don't do what we know we should do.

Your personal process cannot usefully evolve until it reasonably represents what you actually do. When it does, you can measure your performance, identify areas for improvement, and make useful process changes. When your actual, official, and perceived processes converge on your initial process, your process improvement efforts will be most successful.

Characterizing the Current Process

"If you don't know where you are, a map won't help" (Humphrey 1989). To improve human-intensive processes, we must understand how these processes currently work. Although the PSP provides a baseline for software work, you need a picture of where you are and where you intend to go before you can establish a plan to get there.

In general, process-improvement efforts are most successful when they reflect small incremental enhancements, rather than a few large changes. Hence, you should not attempt to completely define a new process. If the task that the process is to implement is one that you currently perform, start with a simple definition of your current work and ask the following questions about it:

- How well do you understand the current process? Can you describe its principal steps, how they relate, and the time spent on each?
- What problems do you have with the current process? Make a list and determine their relative priorities.
- Do the steps in the current process have explicit entry and exit criteria? Make these as specific as you can. If they are unstated or general, the process measurements will be poorly defined and not very helpful.
- Is the current process planned and tracked? If you do not now consistently plan and track your work, this should be your first improvement priority.
- Is your current process sufficiently well measured to permit quantitative improvement? After introducing planning and defining entry and exit criteria, incorporate time and quality measures into the current process.
- Do you have a current process baseline? For the PSP, you now have productivity, quality, and planning data. These can provide a starting point for your process-improvement planning.

Even if you do not have historical data, you can often find some useful measures. You can generally estimate past performance by reconstructing the dates for prior jobs and measuring the sizes of the products produced. Such rough numbers will not be very accurate, but they will provide a crude baseline. Although these approximations are a poor substitute for precise data, they are better than nothing.

Characterizing the Target Process

"If you don't know where you're going, any road will do" (Humphrey 1989). The target process is the ideal. It is the goal you seek to reach. The reason to characterize the target process is to set improvement priorities. You can then relate the process goals and objectives to this target. If minimum cycle time is the highest priority, for example, you might establish parallel process activities, even at additional cost. To minimize late requirements changes, define early steps to validate the requirements, as well as a rigorous change-management procedure.

Next, postulate the target process structure and establish measurable criteria for the major process elements. At the outset, establish only general criteria. An example might be fully defining the target process, including that it predictably produce quality products, and that every process phase be characterized and measured. When some of these measurements are available for your current process, they will help you to decide on an improvement strategy, set achievable goals, and track against these goals.

Don't be too concerned if you have trouble defining a target process. You are likely to find that the only way to learn how to define a process is to define it. Start by examining other processes or copy what I did for the PSP. When you have defined an initial process, use it, measure it, and analyze its behavior. Then you will have the data and experience to make further improvements. Once you have defined the target process, ask the following questions about it:

- Which steps are new and which do you currently perform?
- Which steps do you understand and which are ill-defined?
- Which steps will you frequently repeat and which will be single shot?
- Which steps will consume the most time?
- Where do you expect the most problems?

Use the answers to guide your process-improvement priorities.

The Process Development Strategy

Now that you know where you are and where you want to go, establish the route. Even simple processes have many steps, and they are not all equally important.

Start by briefly defining those steps that you know how to do. If some are vague or confusing, observe how you do them or talk to people who might better understand them. One approach is to pick steps that someone else has performed successfully. If you have no one to learn from, observe someone doing the process, or find the most knowledgeable people on the subject and hold a technical review. You can, of course, even produce a simple process definition and observe yourself. Remember, however, that defining a general conceptual process that no one understands will not be of much help when you are doing the work. To set improvement priorities, ask some new questions and repeat some of the same questions you asked about the target process:

- Which steps do you understand and which are ill-defined?
- Which steps are frequently repeated and which are single shot?
- Which steps take the most time?
- Which steps are the source of the most quality problems?
- Which steps are the source of the most planning and tracking problems?
- Which steps are the source of the most technical problems?

To get the data that you need for continuing process improvement, include the following initial priorities in the process development strategy:

- Start with a project planning phase. This is an essential first step because it helps to make your process predictable. You cannot consistently manage or improve an unpredictable process.
- Include basic resource and product measures, such as time spent by phase, LOC or database elements produced, pages of documentation produced, and so on.
- If practical, include quality measures such as defects by phase or user-reported problems.
- Include estimates for all measured items.
- Define planning and reporting forms for all measured items.
- Define a final report to summarize project results, report key process findings, and suggest process improvements.
- Define a postmortem phase for recording and analyzing the data, producing a final report, and submitting process improvement proposals.

Defining the Initial Process

Now that you have characterized the current and target processes and established objectives, goals, and a strategy, produce the scripts, standards, and forms for the initial process. This initial process should be relatively close to your current

process, but include a few changes that move you towards the target process. Although you should establish aggressive improvement plans, it is almost impossible to make more than a very few changes at any time.

In defining the initial process, start by considering the entire process as a single phase that includes multiple steps. Next, treat each step as a phase and define its steps. Continue this refinement procedure until you reach a refinement level that is reasonably consistent with your current process knowledge and needs.

There is little point in defining process steps that are not current problems. Conversely, when you do not understand a process area, you cannot define it in much detail. The appropriate definition level is as deep as you can knowledgeably go in the areas that cause you the most trouble. Define what you know and then carry the definition a little further. This helps you to learn about this step when you next perform it. It will also give you ideas about how to further refine it the next time.

For example, with the PSP, you started with only general planning, development, and postmortem steps. The first enhancements focused on planning. Measures were introduced to ensure that the appropriate data were gathered, and measurement and estimating techniques were introduced. The postmortem step then ensured that you retained and analyzed the data.

The next PSP improvement refined the design, design review, code, and code review steps. Although the PSP evolution stopped at this point, you could further refine any or all of these steps as you learn more about them. You could, for example, extend the design process to cover a particular design method. This could help you to match the design templates to your design methods, and to provide data on how you spend your time or where you make errors.

Validating the Initial Process

Now it's time to test the process. First, walk through a simulated enactment using data from a project that you have recently completed. If you do not have the needed data, assume some and do a walk-through. Next, try the process on a small project or a prototype. The better the test, the more likely you are to flush out any problems.

13.6 Example Processes

The following sections describe the process scripts for four example processes: the Process Development Process (PDP), the PSP3.0 Process, the Prototype Experimental Process (PEP), and the Product Maintenance Process (PMP). Although

none of these are likely to precisely fit your needs, they should give you some ideas about how to define similar processes for yourself or for your team.

The Process Development Process (PDP)

An example process development script is shown in Table 13.1. Although this process is not for developing code, it is similar to the PSP and should use all of the PSP methods, including PROBE. For a complete PDP process, also produce a plan form and a review checklist.

The PSP process also calls for the Process Improvement Proposal (PIP) form that was introduced with PSP0.1 for recording process problems and improvement suggestions (PIP forms are available at **www.sei.cmu.edu/tsp/psp** and are included in the assignment kits.). Many process problems concern minor details. Such details, however, make the difference between an inconvenient process and an efficient one. Although you may remember that some form or step was a problem, you are likely to forget the details pretty quickly if you do not record them. Keep blank PIP forms at hand and record each of your process improvement ideas as they occur to you. Surprisingly, if you wait even a few minutes, the idea will often be hard to remember. With a personal process, details are important. To fix the details, religiously complete PIPs. They provide an invaluable source of improvement ideas, even for a personal process.

TABLE 13.1 THE PROCESS DEVELOPMENT PROCESS SCRIPT (PDP)

Purpose		• To guide process development or enhancement
Entry Criteria		• A new or updated process is needed. • Requirements or PIPs are available describing the process needs. • The PDP forms, scripts, standards, and review checklist are on hand.
General		• The PDP process may be followed by one developer or an entire team. • When used by teams, PDP is incorporated into the team process. • Develop the scripts, forms, standards, and other elements in that order. • Break larger processes into sections and use PDP to develop each section.
Step	**Activities**	**Description**
1	Produce the Process Plan	• Inventory the existing process (if any). • Produce the new or modified process conceptual design. • Estimate the new, reused, modified, and deleted process elements. • Separately estimate the development effort for the scripts, forms, standards, support materials, and other process elements. • Estimate the total time required for each process element type. • Estimate the total time required for each process phase. • Complete the PDP Plan Summary. *(continued)*

TABLE 13.1 (continued)

Step	Activities	Description
2	Process Requirements	• Analyze the PIPs or other requirements documents. • Define the requirements for the new or modified process.
3	High-level Design	For new or modified processes • Define any process constraints such as training, special tools, or support. • Define the process flow and principal phases.
4	Process Scripts	• Produce or modify each process script. • Using the review checklist, review and correct each process script.
5	Process Forms	• Produce or modify each process form. • Using the review checklist, review and correct each process form.
6	Process Standards	• Produce or modify each process standard. • Using the review checklist, review and correct each process standard.
7	Other Process Elements	• Produce or modify the other process elements. • Using the review checklist, review and correct the other process elements.
8	Process Notebook	• Produce or modify the process notebook, if required. • Using the review checklist, review and correct the process notebook.
9	Process Test	• Test the process by following the script to complete all of the forms. • Where possible, use the process on a real project. • Record all problems and improvement suggestions on PIPs. • Adjust the process to address all of the PIPs.
10	Peer Review	• Where possible, have peers review the process to identify errors, and suggest modifications. • Update the process based on the review results.
11	Process Packaging	• Package the finished process in a form suitable for distribution and use. • Check the final package for errors and make needed corrections.
12	Postmortem	• Record the process data on the Plan Summary. • Calculate the element and phase times and rates. • Record these data in the process database for use in future planning. • Note any PDP process improvements on form PIP.
Exit Criteria		• A completed high-quality process ready for distribution and use • A Project Summary report • Planning data for future use

The PSP3.0 Process

The PSP3.0 process was originally introduced with the first PSP text (Humphrey 1995). It is the cyclic process that I have found helpful for developing programs of several thousand lines of code. Although only the Development Script is shown in Table 13.2, the rest of the PSP3.0 process is described on pages 641 to 649 and pages 725 to 740 of *A Discipline for Software Engineering* (Humphrey 1995).

TABLE 13.2 PSP3.0 DEVELOPMENT SCRIPT

Purpose		To guide development of component-level programs
Entry Criteria		• Problem description or component specifications • Process forms and standards • Historical estimated and actual size and time data
General		• Where the forms and templates are in a support tool, print and retain completed copies in a personal file or notebook. • Where suitable tool support is not available, complete and retain paper copies of all required PSP3 forms. • Record the time spent in each process step. • Record all defects found.
Step	**Activities**	**Description**

1	Requirements and Planning	• Obtain the requirements and produce the development plan.
		  • requirements document
		  • design concept
		  • size, quality, resource, and schedule plans
		• Produce a master Issue Tracking log.

2	High-level Design (HLD)	Produce the design and implementation strategy. • Functional Specifications • State Specifications • Operational Specifications • development strategy • test strategy and plan
3	High-level Design Review (HLDR)	• Review the high-level design. • Review the development and test strategy. • Fix and log all defects found. • Note outstanding issues on the Issue Tracking log. • Log all defects found.
4	Development (PSP 2.1)	• Design the program and document the design in the PSP Design templates. • Review the design and fix and log all defects. • Implement the design. • Review the code and fix and log all defects. • Compile the program and fix and log all defects. • Test the program and fix and log all defects. • Complete the Time Recording log. • Reassess and recycle as needed.
5	Postmortem	Complete the Project Plan Summary form with the actual time, defect, and size data.
Exit Criteria		• A thoroughly tested program • Completed Project Plan Summary with estimated and actual data • Completed Estimating and Planning templates • Completed Design templates • Completed Design Review checklist and Code Review checklist • Completed Test Report template • Completed Issue Tracking log • Completed PIP forms • Completed Time and Defect Recording logs

The PSP3.0 process is cyclic and it builds a large program in a sequence of small incremental steps. Depending on your development style, you could make each step anywhere from about 10 LOC up to hundreds or even thousands of LOC. I found that increments of 30 to about 100 LOC were best, but this is a matter of individual preference; I do not believe that any increment size is generally optimum. What I found most interesting about the PSP3.0 process was that, with high-quality increments, this cyclic strategy actually improved my productivity above what I had achieved with small programs.

Prototype Experimental Process (PEP)

The Prototype Experimental Process (PEP) script shown in Table 13.3 is the one I found most helpful when I first started programming with the .NET environment. Because I was unfamiliar with many of the available functions, I had to experiment before I could use them. Rather than waste time designing something I didn't understand, I devised this hybrid process to help me learn the new environment. The general approach was to get a function to work and then to design it. Initially, I had to write entire programs this way, but I soon learned enough that I could experiment with only one or two new functions before producing the design. Once I had learned the .NET functions needed for the programs I was writing, I switched to the PSP2.1 process.

When you work with an unfamiliar environment, consider using a PEP-like process for your personal work. Then, when a prototype meets one or more of the product's requirements, there is no need to throw away the code you have produced and start over. However, if the prototype is to be used in a delivered product, you must document its design, review it, and have your team inspect it. Even when the prototype will not be part of a delivered product, it is a good idea to document its design, if only to build the design skills needed for working on a TSP team.

The Product Maintenance Process (PMP)

A common challenge that many TSP teams face is making small modifications to large legacy systems. Although the PSP methods will help you to produce nearly defect-free enhancements, these base systems are often highly defective and can have many test defects. A common question is, "How can I use the PSP to fix the quality problems of these large existing systems?" The Product Maintenance Process (PMP) script shown in Table 13.4 is one approach to this problem. Even if this personal process does not precisely meet your needs, it should give you ideas about how to handle such maintenance problems.

The principle behind the PMP is that defects tend to cluster. I like to think of programs as gardens with occasional weeds. Although just picking these lone

TABLE 13.3 PROTOTYPE EXPERIMENTAL PROCESS SCRIPT (PEP)

Purpose		To guide PSP experimental program development
Entry Criteria		• Requirements statement or a problem description • All PEP process scripts, forms, standards, templates, and logs • Historical data for estimated and actual size, time, and quality
General		• Where the forms and templates are in a support tool, print and retain completed copies in a personal file or notebook. • Where suitable tool support is not available, complete and retain paper copies of all required PSP forms. • Record time spent, defects found, and product size produced at each process step.
Step	**Activities**	**Description**
1	Planning	• Base development planning on the best available requirements information. • Produce the program's conceptual design. • Use the PROBE method and Size Estimating template to estimate the program size, development time, and prediction intervals. • For projects of more than a few days duration, complete Task and Schedule Planning templates. • Enter the plan data in the Project Plan and Quality Summaries.
2	Requirements and Strategy	• Assess the plan and conceptual design to identify risks and uncertainties. • Where the risks are significant, start with a prototype development phase.
3	Prototype Development	Develop one or more working prototypes to resolve the identified risks and uncertainties.
4	Development (PSP2.1)	• Produce and document the product's design in State, Operational, Functional, and Logic Specification templates. • Provide designs for all the prototype elements that will be used in the final product. • Review, analyze, and correct the design. • Plan and develop tests to verify correct program operation. • Where required, implement the program's design. • Review and correct the implementation. • Compile, build, test, and fix the program until all tests run without error.
5	Postmortem	• Complete the Project Plan and Quality Summaries with actual time, defect, size, and quality data. • Update the Design and Code Review checklists to reflect new defect data. • Produce PIPs for any process improvement ideas.
Exit Criteria		• A documented, analyzed, reviewed, tested, and corrected program • Project Plan and Quality Summaries with estimated and actual data • Completed Estimating, Planning, Design, and Test Report templates • Updated Design and Code Review checklists. • Completed Defect Recording, Time Recording, and Issue Tracking logs • A personal project notebook with plan and actual data • Completed PIP forms

TABLE 13.4 PRODUCT MAINTENANCE PROCESS SCRIPT (PMP)

Purpose		To guide PSP enhancement and repair of legacy systems

Entry Criteria		• Requirements statement or a problem description
		• All PSP and PMP process scripts, forms, standards, templates, and logs
		• Historical data for estimated and actual project size, time, and quality
		• Historical data for legacy system test defects and customer-reported defects
		  • defect data for every system component
		  • for defective components, defect data by module

General		• Rank the replacement modules by level of user impact. • Where the forms and templates are in a support tool, print and retain completed copies in a personal file or notebook. • Where suitable tool support is not available, complete and retain paper copies of all required PSP and PMP forms. • Record time spent, defects found, and product size produced at each process step.
Step	**Activities**	**Description**
1	Planning	• Base development planning on the enhancement requirements and the resources allocated for cleaning up (remediating) the legacy system. • Produce and document the enhancements' conceptual designs. • Use the PROBE method and Size Estimating template to estimate the size of each enhancement, the development time, and the prediction intervals. • Include the allocated remediation effort in the Task and Schedule plans. • Enter the plan data in the Project Plan and Quality Summaries.
2	Requirements Review	Review and inspect the enhancement requirements and resolve any open issues or questions.
3	Maintenance Strategy	• From the enhancement requirements and conceptual design, identify the legacy components and modules to be modified. • Review the legacy-system defect data and classify components and component modules as defect-free, defective, and replacement. • Rank the replacement modules in priority order, based on user impact. • Identify all of the defective and replacement modules that are to be changed by the enhancements.
4	Enhancement–Defect-free Modules	Follow a personal PSP process to develop, review, inspect, and test the required enhancements for each defect-free module.
5	Enhancement–Defective Modules	For each defective module with one or more planned enhancements • Review the unmodified module and correct any defects found. • If any module has more than one defect found or has design problems, hold a team inspection and fix all defects found. • If the module's design problems are serious, reclassify the module for replacement and proceed to script step 6. • Test and regression test the fixed module before making the enhancements. • Follow a PSP2.1-like process to develop, review, inspect, and test the module's enhancements.

TABLE 13.4 (continued)

6	Enhancement of Replacement Modules	For each replacement module with one or more planned enhancements • Document the unmodified module's design with the PSP Design templates. • If the design has serious defects or is overly complex, produce a new design and implementation, and review and inspect the design and implementation. • If the design appears sound, review and fix the design and code, hold a team inspection and fix all of the defects found. • Test and regression test the fixed module before making the enhancements. • Follow a personal PSP process to develop, review, inspect, and test the module's enhancements.
7	Module Remediation	If remediation resources are available, start with the highest-ranked replacement module that has no planned enhancements. • Document the unmodified module's design with the PSP Design templates. • If the design has serious defects or is overly complex, follow a PSP2.1-like process to produce a new design and implementation and to review and inspect the design and implementation. • If the design appears sound, review and fix the design and code, hold a team inspection and fix all defects found. • Test and regression test the replacement module.

8	Postmortem	• Complete the Project Plan and Quality Summaries with actual time, defect, size, and quality data.
		• Update the Design and Code Review checklists to reflect new defect data.
		• Produce a remediation report, identifying the replaced components and modules.
		  • Summarize the time, size, and defect data for each module repair and replacement.
		  • Update the historical database with the remediation data for use in planning future maintenance efforts.
		• Produce PIPs for any process improvement ideas.

Exit Criteria		• Documented, analyzed, reviewed, tested, and enhanced legacy program • Project Plan and Quality Summaries with estimated and actual data • Completed Estimating, Planning, Design, and Test Report templates • Updated Design and Code Review checklists • Completed Defect Recording, Time Recording, and Issue Tracking logs • Personal project notebook with plan and actual data • Maintenance report and updated database of remediation data • Completed PIP forms

weeds is usually adequate, occasionally you will come across a patch of jungle. Such junglelike parts of programs can be caused by overly complex designs, multiple erroneous fixes, several different developers, defective enhancements, or other similar causes. To fix them, you will usually need to completely rip out and replace that junglelike part of the program.

The PMP strategy uses system-test and user-reported defect data to identify the junglelike patches of code to be replaced. Although the PMP script criteria for defect-free, defective, and replacement modules should be suitable as a starting point, gather data for your products and use these data to set your own classification criteria.

13.7 Process Development Considerations

There is nothing more annoying than a process that prescribes some complex procedure that you can't understand or remember. At the time you wrote it, you may have known exactly what you intended, but if you did not explain it in terms that would be clear later, it is not a fit process. You can also go too far in the other direction. There is no value in defining steps in great detail if a simple summary provides all of the information that you need to perform the task.

Design the process at just the level you need to guide the work. If it provides helpful reminders and useful guidance, you are more likely to use it. People get careless when they follow routine procedures. They need simple directions and checklists to remind them of all of the steps. Remember that *you* are defining a script that will help *you* to faithfully follow the process *you* have chosen to use. You thus must decide where you will need guidance, where reminders are adequate, and where you need no help. You will quickly become proficient at any repetitive process, so make sure that it is convenient during intensive use but also helpful when you need guidance and reminders. The key point to remember is that the process is *not* a tutorial. It is a guideline to help knowledgeable people follow a process that they know how to follow.

Start with a simple process and gradually enhance it. If you make the process too comprehensive, you will do a lot of unnecessary work. When you first use the process, you may find that it is overly detailed where you don't need it and that it doesn't help where it should. In defining processes, you will unconsciously tend to describe the steps that you understand and skip over those that you do not. Your needs, however, are the reverse. Until you make trial definitions, you will not know enough to provide detail where it will help the most.

One example is the Code Review Checklist. Without historical data, you would not know what to include. Although you could start with the checklist in the textbook, it might not fit your needs. From reviewing your PSP data, you could set preliminary review priorities and make a first-draft checklist. After using the checklist, you could then see where to refine it. For example, you could initially call for a general punctuation review and then later list specific punctuation marks.

Forms are more difficult to design than you might expect. To be convenient, a form must capture the essence of the task being performed. If you do not understand

the task, you cannot design a good form. Form design takes time, and your first pass will rarely be complete or convenient. The best advice is to produce a simplified form and then elaborate, revise, and enhance it as you use it.

This evolutionary principle applies to the entire process. Produce a simple initial process, and then elaborate, revise, and enhance it as you gain experience using it. View every process enactment as a test. Record all of your improvement ideas in PIPs and use these PIPs to guide your next process update.

You define a process to help you both understand and use it. Process definition is an experiment in which you almost know the answer. In other words, you generally know how the job should be done but have not defined the precise steps. Start with an experimental definition that extends the process in a direction you think would improve it. When you next perform it, you may find that some ideas worked as expected and that others need to be changed. Such learning is an inherent part of process definition. As you evolve the process, focus on your current needs. When your knowledge and skills improve, your process needs will change, and you should improve the process.

13.8 Summary

Software process development is much like software development. It has similar artifacts and requires many of the same disciplines and methods. The steps for designing a process are to define the needs, establish goals, and define quality criteria. Next, characterize the current and target processes and establish a development strategy. Finally, define and validate the initial process and establish means for its enhancement.

It is practically impossible to produce a usable process the first time you try. The problem is that the work of defining a process changes that process. This concept is best explained by considering the following four processes: The perceived process is what you think you do, the actual process is what you actually do, the official process is what management thinks you do, and the target process is the process to which you are evolving. A principal reason why the quality of our processes is so poor is that we often don't do what we know we should. However, you cannot evolve your process until it reasonably represents what you are doing.

Process development should be planned, measured, tracked, and improved. Guidelines for an effective process development process include planning and measuring the work, tracking the products produced, and recording the time spent by each major product type and development category. To plan process development, you must define the important measures for your work, use these measures to plan the work, and keep a development record. Finally, produce a summary report for each process development.

13.9 Exercises

The assignment for this chapter is to produce a final report on the programs that you wrote while learning the PSP. This work includes three tasks: updating the process that you developed for the midterm report R4, planning the effort to produce the final report, and producing the report. For the details of the assignment, consult the specifications for report R5 in the assignment kit. For the R5 assignment kit, see your instructor or get it at **www.sei.cmu.edu/tsp/psp**. The kit contains the R5 report specifications. In completing this assignment, produce the R5 report according to the specifications given in the assignment kit.

References

Humphrey, W. S. *Managing the Software Process.* Reading, MA: Addison-Wesley, 1989.

———. *A Discipline for Software Engineering.* Reading, MA: Addison-Wesley, 1995.