SWENET Module
PRO2 - Introduction to the Personal Software Process^{SM} (PSP^{SM})
Software Process Measurement
Exercise Booklet

Developed with support from the National Science Foundation

Table of Contents

Exercise: Software Process Measurement

Exercise Description

The exercise involves reading about the PSP and completing a scenario based exercise on tracking PSP work.

Exercise Objectives

Students completing this exercise will be able to:

* Describe the steps in using the PSP to develop a simple program.
* Use the PSP logs (Time, Defect and Plan Summary) to record data.

Reading Assignment

Read Chapters 3, 10 and 12 in [Humphrey 1997].

[Humphrey 1997] Humphrey, Watts S., *Introduction to the Personal Software Process*, Addison Wesley, 1997.

Scenario Assignment

Appendix A contains an exercise that requires reading a scenario (about a student completing a PSP programming assignment) and completing PSP logs and a summary form for the scenario.

Exercise Deliverables

* Completed PSP Time and Defect logs, and a Plan Summary form.

Appendix A

* Software Defects

The term software defect refers to something that is wrong in a program (or in a related artifact - like the design for a program). A defect might be as simple as a missing semicolon (a syntax error) or more complicated, as when an incorrect formula is used (a logic error). The primary measure of the quality of a program is the number of defects that it contains. In developing a program, there are several ways to find defects in a program: personal review of the program code, compiling the program code, and using test data in executing a program.

In order to improve the quality of programs, it is important to find and remove as many defects as possible. This is one of the most difficult parts of programming and requires a disciplined and focused effort by the programmer in order to be successful. The PSP incorporates a number of features that can help a programmer improve software quality. These include the following:

> classifying defects by type
> recording detailed information about each defect found and removed
> formal review of a program using a checklist
> summarizing defects found and removed on the Plan-Summary form
> computing and analyzing defect removal and software quality statistics

* Defect Types

The following table classifies different types of defects:

type	code	name	description
doc	10	Documentatio n	comments, messages
syn	**20**	**Syntax**	**spelling, punctuation, typos, instruction formats, etc.**
bld	30	Build, Package	change management, library, version control
asg	**40**	**Assignment**	**declaration, identifier names, scope, limits**
int	**50**	**Interface**	**procedure calls, context clauses, and references, I/O, user prompts, output labeling**
chk	60	Checking	error messages, inadequate checks or exception handling
dat	70	Data	structure, content
fun	**80**	**Function**	**logic, pointers, loops, recursion, computation, function defects**
sys	90	System	system configuration, timing, memory
env	100	Environment	tool support and operating system problems

The defect types in bold-face (syn, asg, int, and fun) are ones that you are most likely to find in your program and they are the ones you should concentrate on in this exercise. For example, the below table indicates some common defects and there types:

defect description **type**

leaving off a semicolon at the end of an instruction	syn
failing to declare a variable used in a program	asg
placing the parameters in the wrong order in a procedure call	int
using an incorrect Boolean expression as the loop condition in a while loop	fun

* **Instructions**

In the below scenario a student, Inez C. Light, develops a program using the PSP. Using the attached Time Recording Log, Defect Recording Log and Plan-Summary form, track and record Inez's work in each PSP phase.

Note:

* The time spent correcting errors is counted as part of the phase in which the error is discovered. For instance, if a logic error is discovered while testing a program, the time spent in finding the error and correcting it counted as "test" time.

* Estimate the fix time for defects.

* **Scenario**

A first year student, Inez C. Light, is enrolled in CS1 for the spring 2004 semester. She has studied the PSP and decides to use it in a CS1 programming assignment for calculating the mean and standard deviation for a set of input numbers, which are stored in one-dimensional arrays. On Thursday (2/12/04), Inez begins work on the assignment [0800] by reviewing the requirements in the assignment package, including the test requirements, to be sure she understands them. She copies the requirements to her engineering notebook. Then, based on the data presented on past student performance and Inez's feeling about her own performance, she estimates that the program will have about 120 LOC and the assignment will take about 220 minutes to complete. She writes the estimates on her Plan Summary and completes the rest of the planning part of the form. She then places this in her notebook. [0844].

After taking a break for some coffee, she starts to design the program [0850]. She sketches out an overall algorithm for solving the problem and identifies the routines she'll need for input of the data and for computing the mean and standard deviation. She copies the algorithms into her engineering notebook. Inez then goes off to her next class [0955].

After a lunch, Inez begins coding [1332]. While working on coding, Inez is interrupted by a classmate who doesn't understand how to get started. She spends 5 minutes explaining some elements of the requirements and then gets back to coding. Inez finishes coding of all the routines [1417]. After a short break, she prints out a copy of her source code and begins a code review [1425]. Inez checks the printed copy against her code review checklist and finds that her design did not include initialization of a "Sum" variable. She changes the design and fixes the code [1447].

Next Inez compiles the program [1450] and gets an error message, missing semicolon. Looking at the compiler output, Inez sees where the missing semicolon belongs and fixes the source code. She recompiles the program and gets another error message, "undeclared identifier". Surprised, since she thought she declared this identifier, Inez searches through the source code and discovers that the identifier she declared had an '_' in it and this one didn't. She fixes the error, then quickly scans the rest of the source code and finds three more places where she left out the '_', and also fixes them. She again recompiles the program and gets another error message, incorrect parameter type. She studies the code for a couple of minutes, sees the error, and corrects her design and fixes the source code. Inez recompiles the program and gets another semicolon error. She fixes the code, recompiles the code, and gets no compile errors [1543].

After a short break, Inez loads the program and begins executing the first test case [1555]. The program executes and prints out the correct value for the mean of the numbers, but the value of the standard deviation is incorrect. Inez studies the source code for the standard deviation routine and but cannot find the problem. She then tries another test case, but still gets an incorrect answer. Inez then starts using input data to trace through the program, but she gets confused and lost in the trace, and still cannot spot the problem. After talking to some other students about her problem and trying some "trial and error" things, Inez decides to stop work on the problem for the day [1708]. The next day Inez goes to her instructor's office and meets with him to discuss the problem in her program [1000]. He looks through the code and says he thinks she has made a mistake coding the standard deviation formula. He suggests that Inez take a small set of input data (four numbers) and use it to trace through the standard deviation routine. Inez thanks him and heads off to her next class [1015]. That evening [1836] Inez goes back to work on her program. She creates a simple test case and begins tracing through her source code. In a short time, she finds the error in her code (parentheses were use incorrectly). She then corrects her design, changes the code for the standard deviation routine, and recompiles the code, with no compile errors. She then executes the program with all test cases and gets correct results for each case [1909].

*After a break, Inez begins completing her PSP Plan Summary form [1933].
She counts the LOC in her program and finds she has produced 104 LOC. Next, she fills in the actual time data and the actual defect data, and then makes the required calculations. Inez finishes the Plan Summary, prints out all program code, and assembles all assignment documents [1953].*

Time Log

Student	I.C. Light	Date	2/12/04
Instructor	U. R. Thayer	Class	CS 1
Program	Assignment # 3 - Statistics		

Date	Start	Stop	Int	Δ t	Activity	Comment

Defect Recording Log

Student	I.C. Light	Date	2/12/04
Instructor	U. R. Thayer	Class	CS1
Program	Assignment # 3 - Statistics		

Date	Def. Num.	Type	Phase Injected	Phase Removed	Fix Time	Description

type	code	name	description
doc	10	Documentation	comments, messages
syn	**20**	**Syntax**	**spelling, punctuation, typos, instruction formats, etc.**
bld	30	Build, Package	change management, library, version control
asg	**40**	**Assignment**	**declaration, identifier names, scope, limits**
int	**50**	**Interface**	**procedure calls, context clauses, and references, I/O, user prompts, output labeling**
chk	60	Checking	error messages, inadequate checks and exception handling
dat	70	Data	structure, content
fun	**80**	**Function**	**logic, pointers, loops, recursion, computation, function defects**
sys	90	System	system configuration, timing, memory
env	100	Environment	tool support and operating system problems

PSP Project Plan Summary Report

Student	I.C. Light	Date	2/12/04
Instructor	U. R. Thayer	Class	CS1
Program	Assignment # 3 - Statistics		

Program Size (LOC)	Plan	Actual	To Date
Total New & Changed	120		
Maximum Size	200		
Minimum Size	75		

Time in Phase (min.)	Plan	Actual	To Date	To Date %
Planning	20			
Design	60			
Design Review	0			
Code	60			
Code Review	30			
Compile	15			
Test	15			
Postmortem	20			
Total	220			
Maximum Time	330			
Minimum Time	120			

Defects Injected	Plan	Actual	To Date	To Date %
Design	1			
Design Review	0			
Code	4			
Code Review	0			
Compile	0			
Test	0			
Total	5			

Defects Removed	Plan	Actual	To Date	To Date %
Design	0			
Design Review	0			
Code	0			
Code Review	4			
Compile	1			
Test	0			
Total	5			

Summary	Plan	Actual	To Date
LOC/Hour	32.7		
Defects/KLOC	41.6		
Pre-Compile Yield	80%		

Comments: