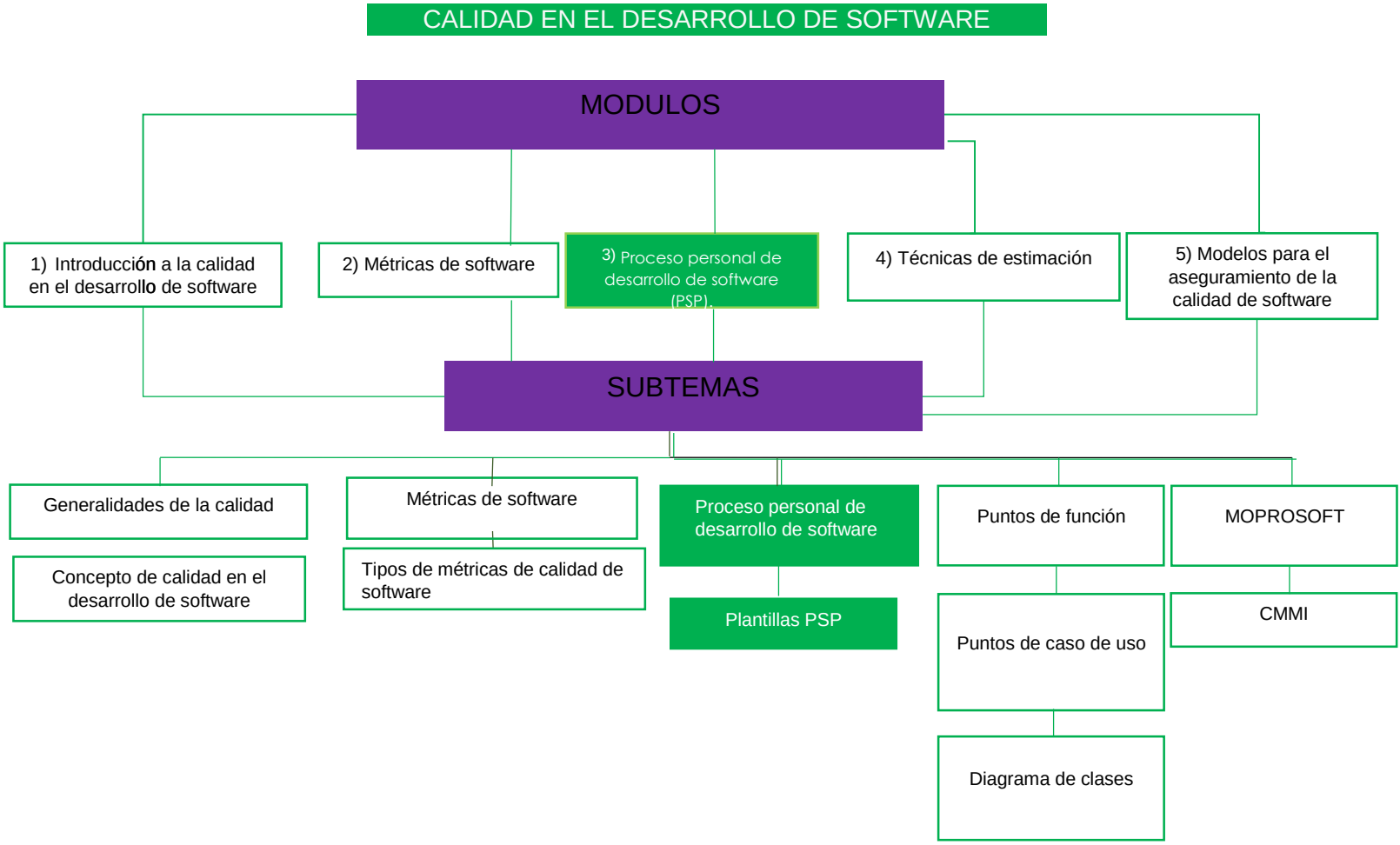
3. Metodología PSP

Calidad de softwarell

Brenda Juárez Santiago
PROFESORA
Quinto Cuatrimestre

Contenido

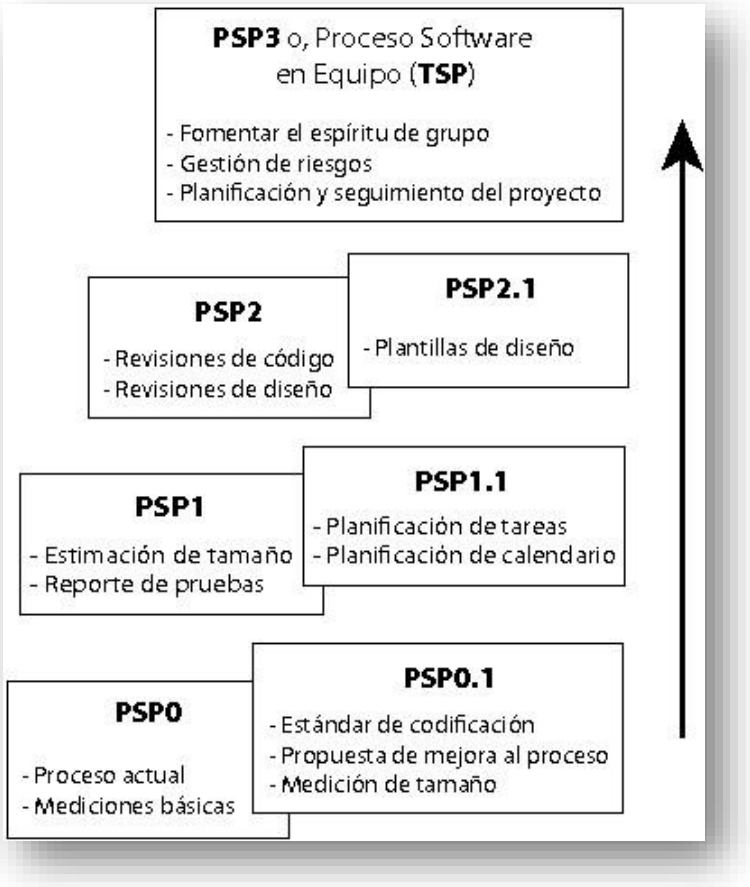
DIAGRAMA DE ASIGNATURA



CAPITULO 3. Proceso personal de desarrollo de software (PSP).

El PSP se desarrolló para ayudar a los ingenieros en software a hacer bien su trabajo. En éste se enseña al ingeniero a aplicar métodos avanzados de ingeniería a sus tareas diarias. Este proceso proporciona métodos detallados de planificación y estimación donde el ingeniero aprende a tener mayor control sobre su trabajo con respecto a planes que previamente establece, así como le ayuda a producir productos de calidad, reduciendo su número de defectos inyectados (Humphrey, 1995).

De acuerdo a los principios de planificación del PSP, para que los ingenieros sean eficaces deben seguir procesos definidos que sean medibles, así como planificar su trabajo. Por otra parte, los principios de calidad del PSP promueven que cada ingeniero realice trabajo de calidad. Para alcanzar esta calidad los ingenieros son responsables de la calidad de los productos que producen, previniendo defectos y haciendo su trabajo de manera correcta (Nichols y Salazar, 2009). Estas mejoras se logran a través de la introducción gradual de nuevos elementos o versiones a la línea base del proceso software personal (Abrahamsson y Kautz, 2002b). La progresión de las distintas versiones del PSP se muestra.



(Gómez, 2014)

Beneficios de PSP

- Los datos y su análisis permitirán determinar las fortalezas y debilidades.
- Los datos y su análisis posterior conducirán hacia nuevas ideas para la mejora del proceso.
- Se tendrá control total sobre el calendario, aceptando sólo aquellos compromisos que se puedan cumplir. Si se enfrenta con una presión no razonable, puede recurrir a la base de datos histórica de desempeño y demostrar que no es posible establecer el compromiso.
- Se gana un sentido de satisfacción personal.
- La parte de calidad ayudará a producir mejores productos de trabajo.
- El equipo de trabajo tendrá mayor confianza porque existe una disciplina para el desarrollo de los productos.

Desventajas de PSP

- El uso de LOC como métrica de estimación tiene sus desventajas, es dependiente del lenguaje, no todos los ingenieros están de acuerdo con lo que es una LOC lógica y son difíciles de visualizar desde la planeación y diseño
- PSP sólo requiere un estimado del tiempo de interrupción, en lugar de obligar al usuario a registrar el tiempo real. Esto hace que el tiempo de interrupción estimado está sujeto a las preferencias individuales.
- El método de estimación PROBE puede no ser efectivo si no existe suficiente correlación entre los datos históricos.
- Los formatos de diseño de PSP2.1 pueden ser redundantes para programadores que tienen acceso a otras herramientas de diseño.
- Es subjetivo determinar si una parte del software es reutilizable.
- No todos los ingenieros ven la definición de productividad de la misma manera.
- PSP esta especialmente enfocado al desarrollo de software y no toma en cuenta el tiempo empleado en la negociación de los requerimientos con el cliente. La fase de requerimientos es un componente clave en cualquier proyecto.
- Seguir PSP al pie de la letra no es viable para muchos ingenieros. Deben ver el método como una estructura para el desarrollo de una práctica de desarrollo de software con calidad. Cada uno de los métodos debe ser ajustado a la tecnología, práctica, fortalezas y debilidades de cada desarrollador. Es importante destacar que las métricas existen para evaluar el proceso no a las personas.

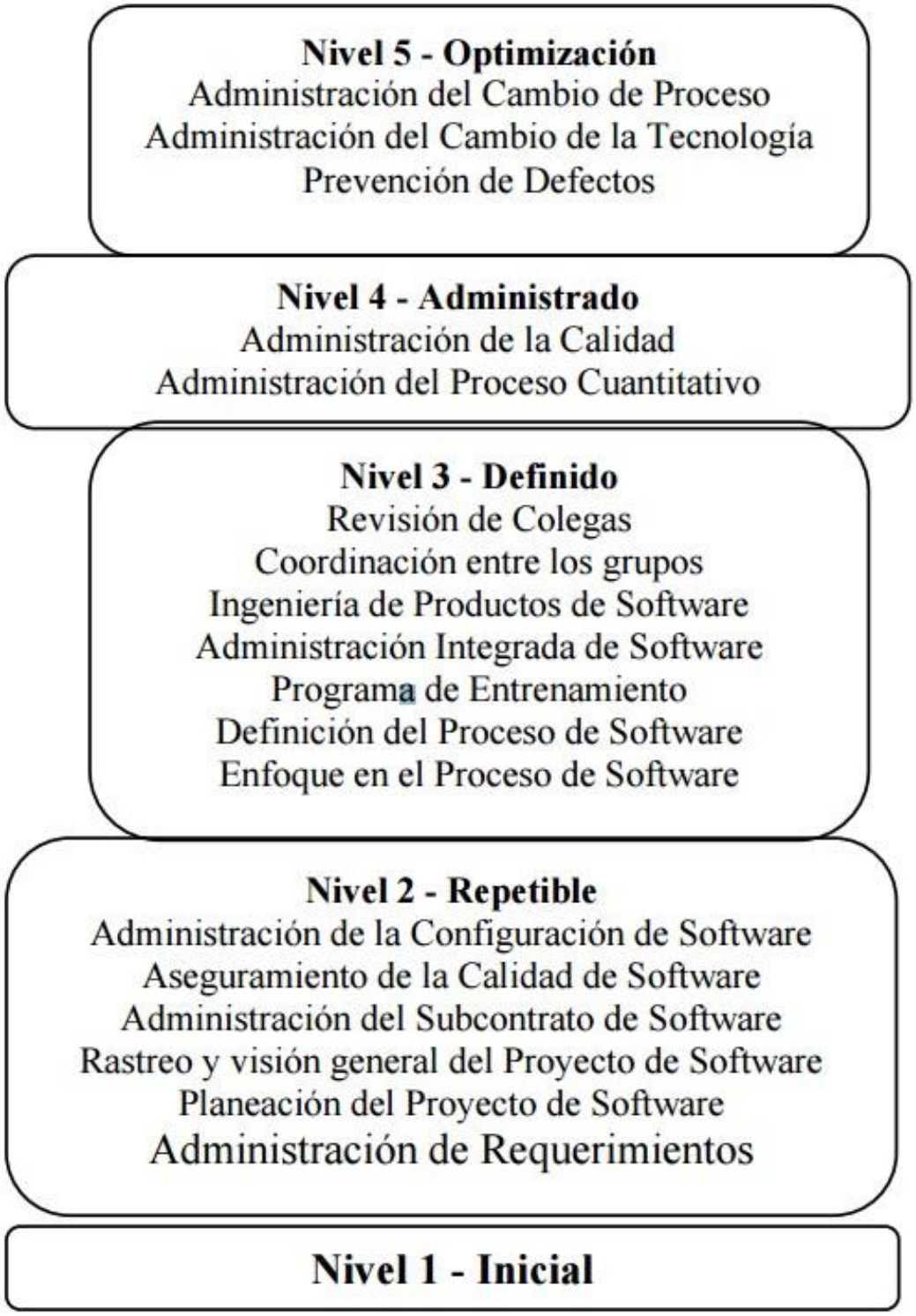
Herramientas automatizadas

PSP requiere de herramientas que permitan

- Simplificar el proceso de recolección de los datos de tiempo y defectos.
- Automatizar los cálculos requeridos.
- Facilitar el acceso a cada una de las formas.
- Llenar automáticamente cada una de las formas con los datos registrados, eliminando la necesidad de copiar la información a mano.
- El control de tiempo puede hacerse más preciso en segundos en lugar de minutos.
- Implementar una guía de tareas automatizada.
- Cada usuario pueda adecuar la herramienta a sus prácticas.
- Salvar automáticamente los datos históricos y dejarlos listos para su análisis.

(Escobar, 2010)

Figura 3.1 Diagrama de Elementos del PSP



Plantillas PSP

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

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.

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

PSP Defect Recording Log

Student ____________________ Date ____________
Program ____________________ Program # ____________
Instructor ____________________ Language ____________

Project | Date | Number | Type | Inject | Remove | Fix Time | Fix Ref.
Description:

Project | Date | Number | Type | Inject | Remove | Fix Time | Fix Ref.
Description:

Project | Date | Number | Type | Inject | Remove | Fix Time | Fix Ref.
Description:

Project | Date | Number | Type | Inject | Remove | Fix Time | Fix Ref.
Description:

Project | Date | Number | Type | Inject | Remove | Fix Time | Fix Ref.
Description:

Project | Date | Number | Type | Inject | Remove | Fix Time | Fix Ref.
Description:

Project | Date | Number | Type | Inject | Remove | Fix Time | Fix Ref.
Description:

Project | Date | Number | Type | Inject | Remove | Fix Time | Fix Ref.
Description:

PSP Defect Recording Log Instructions

Purpose	- Use this form to hold data on the defects that you find and correct. - These data are used to complete the Project Plan Summary form.
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 on which 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 that 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.

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

PSP Process Improvement Proposal (PIP)

Student _____ Date _____
Program _____
Program # _____
Instructor Language

Problem Description

Briefly describe the problems that you encountered.

Proposal Description

Briefly describe the process improvements that you propose.

Other Notes and Comments

Note any other comments or observations that describe your experiences or improvement ideas.

PSP Process Improvement Proposal (PIP) Instructions

Purpose	- To provide a way to record process problems and improvement ideas - To provide an orderly record of your process improvement ideas - To record any other noteworthy observations
General	Use the PIP form to - record process improvement ideas as they occur to you - establish priorities for your improvement plans - describe lessons learned and unusual conditions Keep PIP forms on hand while using the PSP. - Record process problems even without proposed solutions. - Submit a PIP with each PSP assignment report.
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.
Problem Description	Briefly describe any problems or experiences that led to this PIP.
Proposal Description	Describe the proposed improvement as explicitly as possible.
Other Notes and Comments	Briefly describe any other observations or facts that would later help you to - remember what you did while writing this program - remember an idea for a future improvement - explain to your instructor something you did and why you did it

C++ Coding Standard

Purpose	To guide implementation of C++ programs
Program Headers	Begin all programs with a descriptive header.
Header Format	/**/ /* Program Assignment: the program number */ /* Name: your name */ /* Date: the date you started developing the program */ /* Description: a short description of the program and what it does */ /**/
Listing Contents	Provide a summary of the listing contents
Contents Example	/**/ /* Listing Contents: */ /* Reuse instructions */ /* Modification instructions */ /* Compilation instructions */ /* Includes */ /* Class declarations: */ /* CData */ /* ASet */ /* Source code in c:/classes/CData.cpp: */ /* CData */ /* CData() */ /* Empty() */ /**/
Reuse Instructions	- Describe how the program is used: declaration format, parameter values, types, and formats. - Provide warnings of illegal values, overflow conditions, or other conditions that could potentially result in improper operation.
Reuse Instruction Example	/**/ /* Reuse instructions */ /* int PrintLine(char *line_of_character) */ /* Purpose: to print string, 'line_of_character', on one print line */ /* Limitations: the line length must not exceed LINE_LENGTH */ /* Return 0 if printer not ready to print, else 1 */ /**/
Identifiers	Use descriptive names for all variable, function names, constants, and other identifiers. Avoid abbreviations or single-letter variables.
Identifier Example	Int number_of_students; /* This is GOOD */ Float: x4, j, ftave; /* This is BAD */
Comments	- Document the code so the reader can understand its operation. - Comments should explain both the purpose and behavior of the code. - Comment variable declarations to indicate their purpose.
Good Comment	If(record_count > limit) /* have all records been processed? */
Bad Comment	If(record_count > limit) /* check if record count exceeds limit */
Major Sections	Precede major program sections by a block comment that describes the processing done in the next section.
Example	/**/ /* The program section examines the contents of the array 'grades' and calcu- */ /* lates the average class grade. */ /**/
Blank Spaces	- Write programs with sufficient spacing so they do not appear crowded. - Separate every program construct with at least one space.

(continued)

C++ Coding Standard (continued)

Indenting	- Indent each brace level from the preceding level. - Open and close braces should be on lines by themselves and aligned.

Indenting Example	while (miss_distance > threshold)
	{
	success_code = move_robot (target _location);
	if (success_code == MOVE_FAILED)
	{ printf("The robot move has
	failed.\n"); }
	}

Capitalization	- Capitalize all defines. - Lowercase all other identifiers and reserved words. - To make them readable, user messages may use mixed case.
Capitalization Examples	#define DEFAULT-NUMBER-OF-STUDENTS 15 int class-size = DEFAULT-NUMBER-OF-STUDENTS;

Size Counting Standard Template

Definition Name: _____ Language: _____

Author: _____ Date: _____

Count Type	**Type**	**Comments**
Physical/Logical		
Statement Type	**Included**	**Comments**
Executable		
Nonexecutable:		
Declarations		
Compiler Directives		
Comments		
Blank lines		
Other elements		
Clarifications		**Examples/Cases**

Note		

Note		
Note		

Test Report Template

Student		Date	
Program			
Program #			
Instructor		Language	

Test Name/Number Test Objective Test Description   Test Conditions   Expected Results   Actual Results
Test Name/Number Test Objective Test Description   Test Conditions   Expected Results   Actual Results

Test Report Template Instructions

Purpose	- To maintain a record of the tests run and the results obtained - To be sufficiently complete so that you can later re-run the same tests and get the same results - To facilitate regression testing of modified or reused programs
General	- Expand this table or use multiple copies as needed. - Report all the tests that were successfully run. - Be as brief and concise as possible.
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.
Test Name/Number	Uniquely identify each test for each program. - the same tests with different data - the same data with different tests
Test Objective	Briefly describe the objective of the test.
Test Description	Describe each test's data and procedures in sufficient detail to facilitate its later use as a regression test.
Test Conditions	- List any special configuration, timing, fix, or other conditions of the test. - When multiple tests are run with different parameters or under varying conditions, separately list each.
Expected Results	List the results that the test should produce if it runs properly.
Actual Results	List the results that were actually produced.

Size Estimating Template

Student _____ Date _____
Program _____ Program # _____
_____ Language _____
Instructor _____

Size Measure

	Estimated			
Total				BA
Base Parts	**Base**	**Deleted**	**Modified**	**Added**
	B	**D**	**M**	

	Actual			
Base Parts	**Base**	**Deleted**	**Modified**	**Added**
Total				

	Estimated				Actual	
Parts Additions	**Type**	**Items**	**Rel. Size**	**Size***	**Size***	**Items**
Total				PA		

	Estimated	Actual
Reused Parts	**Size**	**Size**

Total		

R

(continued)

Size Estimating Template (continued)

Student _____ Program _____

	Size	Time

PROBE Calculation Worksheet (Added and Modified)

Added size (A): A = BA+PA

Estimated Proxy Size (E): E = BA+PA+M

PROBE estimating basis used: (A, B, C, or D)

Correlation: (R^2)

Regression Parameters: β_0 Size and Time

Regression Parameters: β_1 Size and Time

Projected Added and Modified Size (P): $P = \beta 0_{size} + \beta 1_{size} * E$

Estimated Total Size (T): T = P + B - D - M + R

Estimated Total New Reusable (NR): sum of * items

Estimated Total Development Time: $Time = \beta 0_{time} + \beta 1_{time} * E$

Prediction Range: Range

Upper Prediction Interval: UPI = P + Range

Lower Prediction Interval: LPI = P - Range

Prediction Interval Percent:

Size Estimating Template Instructions

Purpose	Use this form with the PROBE method to make size estimates.
General	- A part could be a module, component, product, or system. - Where parts have a substructure of methods, procedures, functions, or similar elements, these lowest-level elements are called items. - Size values are assumed to be in the unit specified in size measure. - Avoid confusing base size with reuse size. - Reuse parts must be used without modification. - Use base size if additions, modifications, or deletions are planned. - If a part is estimated but not produced, enter its actual values as zero. - If a part is produced that was not estimated, enter it using zero for its planned values.
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. - Enter the size measure you are using.
Base Parts	If this is a modification or enhancement of an existing product - measure and enter the base size (more than one product may be entered as base) - estimate and enter the size of the deleted, modified, and added size to the base program After development, measure and enter the actual size of the base program and any deletions, modifications, or additions.
Parts Additions	If you plan to add newly developed parts - enter the part name, type, number of items (or methods), and relative size - for each part, get the size per item from the appropriate relative size table, multiply this value by the number of items, and enter in estimated size - put an asterisk next to the estimated size of any new-reusable additions After development, measure and enter - the actual size of each new part or new part items - the number of items for each new part
Reused Parts	If you plan to include reused parts, enter the - name of each unmodified reused part - size of each unmodified reused part After development, enter the actual size of each unmodified reused part.

PROBE Calculation Worksheet Instructions

Purpose	Use this form with the PROBE method to make size and resource estimate calculations.
General	The PROBE method can be used for many kinds of estimates. Where development time correlates with added and modified size - use the Added and Modified Calculation Worksheet - enter the resulting estimates in the Project Plan Summary - enter the projected added and modified value (P) in the added and modified plan space in the Project Plan Summary If development time correlates with some other combination of sizeaccounting types - define and use a new PROBE Calculation Worksheet - enter the resulting estimates in the Project Plan Summary - use the selected combination of size accounting types to calculated the projected size value (P) - enter this P value in the Project Plan Summary for the appropriate plan size for the size-accounting types being used

PROBE Calculations: Size (Added and Modified)	- Added Size (A): Total the added base code (BA) and Parts Additions (PA) to get Added Size (A). - Estimated Proxy Size (E): Total the added (A) and modified (M) sizes and enter as (E). - PROBE Estimating Basis Used: Analyze the available historical data and select the appropriate PROBE estimating basis (A, B, C, or D). - Correlation: If PROBE estimating basis A or B is selected, enter the correlation value (R^2) for both size and time. - Regression Parameters: Follow the procedure in the PROBE script to calculate the size and time regression parameters (β_0 and β_1), and enter them in the indicated fields. - Projected Added and Modified Size (P): Using the size regression parameters and estimated proxy size (E), calculate the projected added and modified size (P) as $P = \beta_{0Size} + \beta_{1Size} * E$. - Estimated Total Size (T): Calculate the estimated total size as T = P+B-D-M+R. - Estimated Total New Reusable (NR): Total and enter the new reusable items marked with *.
PROBE Calculations: Time (Added and Modified)	- PROBE Estimating Basis Used: Analyze the available historical data and select the appropriate PROBE estimating basis (A, B, C, or D). - Estimated Total Development Time: Using the time regression parameters and estimated proxy size (E), calculate the estimated development time as $Time = \beta_{0Time} + \beta_{1Time} * E$.
PROBE Calculations: Prediction Range	- Calculate and enter the prediction range for both the size and time estimates. - Calculate the upper (UPI) and lower (LPI) prediction intervals for both the size and time estimates. - Prediction Interval Percent: List the probability percent used to calculate the prediction intervals (70% or 90%).
After Development (Added and Modified)	Enter the actual sizes for base (B), deleted (D), modified (M), and added base code (BA), parts additions (PA), and reused parts (R).

Task Planning Template

Student _____ Date _____

Program _____ _____

Program # _____ _____

Instructor Language

Task			Plan							Actual		
Program/Part	Phase	Task Name	Task Hours	Cumulative Task Hours	Week Due	Week	Week Predicted	Planned Value PV	Cumulative PV	Task Hours	Cumulative EV	Week

Totals												

Task Planning Template Instructions

Purpose	- To estimate the development time for each project task - To compute the planned value for each project task - To estimate the planned completion date for each task - To provide a basis for tracking schedule progress even when the tasks are not completed in the planned order
General	- Complete the Schedule Planning and Task Planning templates together. - Select tasks that have explicit completion criteria, i.e., plan completed, program compiled and defects corrected, etc. - Expand this template or use multiple pages as needed. - Include every significant task. - Use task names and numbers that support the activity and are consistent with the project work breakdown structure. - Note that most support tools will do the earned-value calculations.
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.
Program/Part	Enter the program or part to which the task relates.
Phase	Enter the phase for each task.
Task Name	Enter task names and/or numbers in the order in which you expect to complete them.
Plan: Task Hours	Enter the total planned hours for each task.
Plan: Cumulative Task Hours	Enter the cumulative sum of the total planned task hours.
Plan: Week Due	If the task has a specific due date, enter the week due here.
Plan: Week	- On the Schedule template, find the plan cumulative schedule hours entry that equals or just exceeds each cumulative task hours entry on this form. - The week number in that row of the Schedule template is the plan week number for the task on Task template. - If several weeks on the Schedule template have the same cumulative value, enter the earliest week number.
Plan: Week Predicted	- On the Schedule template, find the predicted cumulative earned value entry that equals or just exceeds each cumulative planned value entry on this form. - The week number in that row of the Schedule template is the predicted week number for the task on the Task template. - If several weeks on the Schedule template have the same cumulative value, enter the earliest week number.
Plan: Planned Value (PV)	- Total the planned hours for all tasks. - Find the percentage each task's planned hours is of total hours. - Enter this percentage as the planned value for each task.
Plan: Cumulative PV	Enter the cumulative sum of the planned values.
Actual: Task Hours	When a task is completed, enter the hours spent on the task.
Actual: Cumulative Earned Value (EV)	- Each week, total the EV for all completed tasks and enter that total beside the latest completed task. - Also enter the weekly and cumulative total EV on the Schedule template.
Actual: Week	As a task is completed, enter the week number it was completed.

Schedule Planning Template

Student _____ Date _____

Program _____ _____

Program # _____ _____

Instructor _____ Language _____

		Plan			Actual				Predicted
Week No.	Date	Schedule Hours	Cumulative Schedule Hours	Cumulative Planned Value	Schedule Hours	Cumulative Schedule Hours	Week Earned Value	Cumulative Earned Value	Cumulative Predicted Earned Value

Schedule Planning Template Instructions

Purpose	- To record the estimated and actual hours expended by calendar period - To relate the task planned value to the calendar schedule
General	- Expand this template or use multiple pages as needed. - Complete in conjunction with the Task Planning template.
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.
Week No.	- From the project start, enter a week number, typically starting with 1. - For very small projects, it may be more convenient to use days instead of weeks.
Date	- Enter the calendar date for each week. - Pick a standard day in the week (for example, Monday).
Plan: Schedule Hours	- Enter the planned number of schedule hours that you expect to spend working on the project each week. - Consider non-work time such as vacations, holidays, etc. - Consider other committed activities such as e-mail, courses, meetings, and other projects.
Plan: Cumulative Schedule Hours	Enter the planned cumulative schedule hours through each week.
Plan: Cumulative Planned Value	For each week - take the plan cumulative schedule hours from the Schedule template - on the Task template, find the task with nearest equal or lower plan cumulative task hours and note its plan cumulative value - enter this cumulative value in the Schedule template for that week - if the cumulative value for the prior week still applies, enter it again
Actual: Schedule Hours	At the end of each week, enter the actual schedule hours worked in that week.
Actual: Cumulative Schedule Hours	At the end of each week, calculate and enter the actual cumulative schedule hours for the week.
Actual: Week Earned Value	At the end of each week, calculate the total earned value for each task completed during the week and enter here.
Actual: Cumulative Earned Value	At the end of each week, calculate the cumulative earned value for the week.
Predicted: Cumulative Predicted Earned Value	At the end of each week, recalculate the cumulative predicted earned value for the current week through to the end of the schedule. - Enter the current week's actual cumulative earned value as the current week's cumulative predicted earned value. - Calculate the average actual earned value per hour worked on the job to date (Actual Cumulative EV/Actual Cumulative Schedule Hours). - For each week n, starting with the next week, multiply the average earned value per planned hour by the planned hours for week n. Add the result to the cumulative predicted earned value for the preceding week and enter in the cumulative predicted earned value for that week. Repeat for each week until the cumulative predicted earned value reaches 100.

Code Review Checklist

Student Date

Program # _____

Program _____

Instructor _____

Language C++

Purpose	To guide you in conducting an effective code review
General	- Review the entire program for each checklist category; do not attempt to review for more than one category at a time! - As you complete each review step, check off that item in the box at the right. - Complete the checklist for one program or program unit before reviewing the next.

Complete	Verify that the code covers all of the design.				
Includes	Verify that the includes are complete.				
Initialization	Check variable and parameter initialization. - at program initiation - at start of every loop - at class/function/procedure entry				
Calls	Check function call formats. - pointers - parameters - use of '&'				
Names	Check name spelling and use. - Is it consistent? - Is it within the declared scope? - Do all structures and classes use '.' reference?				
Strings	Check that all strings are - identified by pointers - terminated by NULL				
Pointers	Check that all - pointers are initialized NULL - pointers are deleted only after new - new pointers are always deleted after use				
Output Format	Check the output format. - Line stepping is proper. - Spacing is proper.				
() Pairs	Ensure that () are proper and matched.				
Logic Operators	- Verify the proper use of ==, =, ||, and so on. - Check every logic function for ().				

Line-by-line check	Check every line of code for - instruction syntax - proper punctuation				
Standards	Ensure that the code conforms to the coding standards.				
File Open and Close	Verify that all files are - properly declared - opened - closed				

PSP2.1 Project Plan Summary

Student _____ Date _____

Program _____ Program # _____

Instructor _____ Language _____

Summary	**Plan**	**Actual**	**To Date**
Size/Hour			
Planned Time			
Actual Time			
CPI (Cost-Performance Index)			
			(Planned/Actual)
% Reuse			

Program Size	**Plan**	**Actual**	**To Date**
Base (B)			

% New Reusable			

Test Defects/KLOC or equivalent

Total Defects/KLOC or equivalent

Yield %

% Appraisal COQ

% Failure COQ

COQ A/F Ratio

PQI

	(Measured)	(Measured)	
Deleted (D)			
	(Estimated)	(Counted)	
Modified (M)			
	(Estimated)	(Counted)	
Added (A)			
	(A+M □ M)	(T □ B + D □ R)	
Reused (R)			
	(Estimated)	(Counted)	
Added and Modified (A+M)			
	(Projected)	(A + M)	
Total Size (T)			

	(A+M + B □ M □ D + R)	(Measured)	
Total New Reusable			

Estimated Proxy Size (E)

Upper Prediction Interval (70%)

Lower Prediction Interval (70%)

(continued)

PSP2.1 Project Plan Summary (continued)

Student _____ Program # _____

Time in Phase (min.)	**Plan**	**Actual**	**To Date**	**To Date %**
Planning				

Defects Injected	**Plan**	**Actual**	**To Date**	**To Date %**
Planning				

Design				
Design Review				
Code				
Code Review				
Compile				
Test				
Postmortem				
Total				

Total Time UPI (70%) ***Total Time LPI (70%)***

Defects Removed	**Plan**	**Actual**	**To Date**	**To Date %**

Planning				
Design				
Design Review				
Code				
Code Review				
Compile				
Test				
Total Development				
Design				
Design Review				
Code				
Code Review				
Compile				
Test				
Total Development				
After Development				

Defect Removal Efficiency	**Plan**	**Actual**	**To Date**
Defects/Hour Design Review			

Defects/Hour Code Review

Defects/Hour Compile

Defects/Hour Test

DRL (DLDR/UT)

DRL (Code Review/UT)

DRL (Compile/UT)

PSP2.1 Plan Summary Instructions

Purpose	To hold the plan and actual data for programs or program parts
General	- Use the most appropriate size measure, either LOC or element count. - "To Date" is the total actual to-date values for all products developed. - A part could be a module, component, product, or system.
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.
Summary	- Enter the added and modified size per hour planned, actual, and to-date. - Enter the planned and actual times for this program and prior programs. - For planned time to date, use the sum of the current planned time and the to-date planned time for the most recent prior program. - CPI = (To Date Planned Time)/(To Date Actual Time). - Reuse % is reused size as a percentage of total program size. - New Reusable % is new reusable size as a percentage of added and modified size. - Enter the test and total defects/KLOC or other appropriate measure. - Enter the planned, actual, and to-date yield before compile.
Quality Indicators	- ***Appraisal COQ: the percentage of development time in reviews.*** - ***Failure COQ: the percentage of development time in compile and test.*** - ***A/FR: the ratio of appraisal to failure COQ.*** - ***Enter the planned, actual, and to-date PQI (the process quality index)***

Program Size	- Enter plan base, deleted, modified, reused, new reusable, and total size from the Size Estimating template. - Enter the plan added and modified size value (A+M) from projected added and modified size (P) on the Size Estimating template. - Calculate plan added size as A+M – M. - Enter estimated proxy size (E) from the Size Estimating template. - Enter actual base, deleted, modified, reused, total, and new reusable size from the Size Estimating template. - Calculate actual added size as T-B+D-R and actual added and modified size as A+M. - Enter to-date reused, added and modified, total, and new reusable size.
Time in Phase	- Enter plan total time in phase from the estimated total development time on the Size Estimating template. - Distribute the estimated total time across the development phases according to the To Date % for the most recently developed program. - Enter the actual time by phase and the total time. - To Date: Enter the sum of the actual times for this program plus the todate times from the most recently developed program. - To Date %: Enter the percentage of to-date time in each phase.
Prediction Interval	***- Enter the 70% UPI and LPI total size and time ranges.***

(continued)

PSP2.1 Plan Summary Instructions (continued)

Defects Injected	- Enter the total estimated defects injected. - Distribute the estimated total defects across the development phases according to the To Date % for the most recently developed program. - 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 todate values for the most recent previously developed program. - To Date %: Enter the percentage of the to-date defects injected by phase.
Defects Removed	- Enter the estimated total defects removed. - Distribute the estimated total defects across the development phases according to the To Date % for the most recently developed program. - 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.
Defect-Removal Efficiency	- Calculate and enter the defects removed per hour in design review, code review, compile, and test. - For DRL, take the ratio of the review and compile rates with test. - Where there were no test defects, use the to-date test defect/hour value.

Design Review Checklist

Student _____ Date _____

Program _____ Program # _____

Instructor _____ Language _____

Purpose	To guide you in conducting an effective design review
General	- Review the entire program for each checklist category; do not attempt to review for more than one category at a time! - As you complete each review step, check off that item in the box at the right. - Complete the checklist for one program or program unit before reviewing the next.

Complete	Verify that the design covers all of the applicable requirements. - All specified outputs are produced. - All needed inputs are furnished. - All required includes are stated.				
External Limits	Where the design assumes or relies upon external limits, determine if behavior is correct at nominal values, at limits, and beyond limits.				
Logic	***Use a trace table, mathematical proof, or similar method to verify the logic.*** - Verify that program sequencing is proper. Stacks, lists, and so on are in the proper order. Recursion unwinds properly. - Verify that all loops are properly initiated, incremented, and terminated. - Examine each conditional statement and verify all cases.				
State Analysis	***For each state machine, verify that the state transitions are all complete and orthogonal.***				
Internal Limits	Where the design assumes or relies upon internal limits, determine if behavior is correct at nominal values, at limits, and beyond limits.				
Special Cases	- Check all special cases. - Ensure proper operation with empty, full, minimum, maximum, negative, and zero values for all variables. - Protect against out-of-limits, overflow, and underflow conditions. - Ensure "impossible" conditions are absolutely impossible. - Handle all possible incorrect or error conditions.				
Functional Use	- Verify that all functions, procedures, or methods are fully understood and properly used. - Verify that all externally referenced abstractions are precisely defined.				

System Considerations	- Verify that the program does not cause system limits to be exceeded. - Verify that all security-sensitive data are from trusted sources.				
	- Verify that all safety conditions conform to the safety specifications.				
Names	Verify that - all special names are clear, defined, and authenticated - the scopes of all variables and parameters are self-evident or defined - all named items are used within their declared scopes				
Standards	Ensure that the design conforms to all applicable design standards.				

Operational Specification Template

Student _____ **Date** _____

Program _____ **Program** _____

_____

Instructor _____ **Language** _____

Scenario Number		**User Objective**	
Scenario Objective			
Source	**Step**	**Action**	**Comments**

Operational Specification Template Instructions

Purpose	- To hold descriptions of the likely operational scenarios followed during program use - To ensure that all significant usage issues are considered during program design - To specify test scenarios
General	- Use this template for complete programs, subsystems, or systems. - Group multiple small scenarios on a single template, as long as they are clearly distinguished and have related objectives. - List the major scenarios and reference other exception, error, or special cases under comments. - Use this template to document the operational specifications during planning, design, test development, implementation, and test. - After implementation and testing, update the template to reflect the actual implemented product.
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.
Scenario Number	Where several scenarios are involved, reference numbers are needed.
User Objective	List the users' likely purpose for the scenario, for example, to log onto the system or to handle an error condition.
Scenario Objective	List the designer's purpose for the scenario, for example, to define common user errors or to detail a test scenario.
Source	- Enter the source of the scenario action. - Example sources could be user, program, and system.
Step	Provide sequence numbers for the scenario steps. These facilitate reviews and inspections.
Action	Describe the action taken, such as - Enter incorrect mode selection. - Provide error message.
Comments	List significant information relating to the action, such as - User enters an incorrect value. - An error is possible with this action.

Functional Specification Template

Student _____ **Date** _____

Program _____ **Program #** _____

_____

Instructor **Language**

Class Name	
Parent Class	

Attributes

Declaration	**Description**

Items

Declaration	**Description**

Functional Specification Template Instructions

Purpose	- To hold a part's functional specifications - To describe classes, program modules, or entire programs
General	- Use this template for complete programs, subsystems, or systems. - Use this template to document the functional specifications during planning, design, test development, implementation, and test. - After implementation and testing, update the template to reflect the actual implemented product.
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.
Class Name	- Enter the part or class name and the classes from which it directly inherits. - List the class names starting with the most immediate. - Where practical, list the full inheritance hierarchy.
Attributes	- Provide the declaration and description for each global or externally visible variable or parameter with any constraints. - List pertinent relationships of this part with other parts together with the multiplicity and constraints.
Items	- Provide the declaration and description for each item. - Precisely describe the conditions that govern each item's return values. - Describe any initialization or other key item responsibilities.

Example Items	An item could be a class method, procedure, function, or database query, for example.

State Specification Template

Student _____ Date _____

Program _____ _____

_____ Program _____

#

Instructor Language

State Name	Description
Function/Parameter	**Description**

States/Next States	Transition Condition	Action

State Specification Template Instructions

Purpose	- To hold the state and state transition specifications for a system, class, or program - To support state-machine analysis during design, design reviews, and design inspections

General	- This form shows each system, program, or routine state, the attributes of that state, and the transition conditions among the states. - Use this template to document the state specifications during planning, design, test development, implementation, and test. - After implementation and testing, update the template to reflect the actual implemented product.
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.
State Name	- Name all of the program's states. - Also enter each state name in the header space at the top of each "States/Next States" section of the template.
State Name Description	- Describe each state and any parameter values that characterize it. - For example, if a state is described by SetSize=10 and SetPosition=3, list SetSize=10 and SetPosition=3.
Function/Parameter	- List the principal functions and parameters. - Include all key variables or methods used to define state transitions or actions.
Function/Parameter Description	- For each function, provide its declaration, parameters, and returns. - For each parameter, define its type and significant values.
Next State	- For each state, list the names of all possible next states. - Include the state itself.
Transition Condition	List the conditions for transition to each next state. - Use a mathematical or otherwise precise notation. - If the transition is impossible, list "impossible," with a note saying why.
Action	List the actions taken with each state transition.

Logic Specification Template

Student _____ Date _____

Program _____ Program # _____

Instructo Languag r e _____

Design

References _____

_____

_____

_____

Parameters

Logic Specification Template Instructions

Purpose	- To contain the pseudocode for a program, component, or system - To enable precise and complete program implementation - To facilitate thorough design and implementation reviews and inspections
General	- Use this template to document the program's detailed logic. - After implementation and testing, update the template to reflect the actual implemented product. - During detailed design, write the pseudocode needed to describe all of the program's logic. - Use plain language and avoid using programming instructions wherever practical.
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.
Design References	List the references used to produce the program's logical design. - the Operational, Functional, and State templates - the program's requirements - any other pertinent source
Parameters	- Where needed, define any parameters or abbreviations used. - Avoid duplicating definitions on other templates and reference these other definitions where they are needed.

(PSP, s.f.)