PSP0.1

Estándar de codificación
Mediciones de tamaño
Propuesta de mejora del proceso

El proceso de Planeación I y II

Carlos Mojica

Introducción

Planeación es el primer paso en PSP por 3 razones:

1. **Sin buenos planes no se puede administrar ni siquiera proyectos de software de tamaño modesto.**

2. **Planear es una habilidad que se puede aprender y mejorar con la práctica.**

3. **Una buena planeación ayudará a hacer un trabajo mejor.**

El método de planeación de PSP es para uso personal, sin embargo, ayuda para la planeación futura de proyectos más complejos y grandes.

¿Por qué hacer planes?

Lo que se busca es encontrar soluciones económicas y en tiempo a las necesidades planteadas considerando los costos y los tiempos.

Los planes ayudan a contraer compromisos para soportar el negocio (identificar tareas, dar seguimiento al tamaño de los productos, registrar el tiempo que toma desarrollarlos - FRPP).

¿Por qué hacer planes?

Los planes para los proyectos grandes serán más realistas cuando estén compuestos de múltiples planes personales realizados por los individuos o equipos que realizarán el trabajo.

Para planear hay que:

1 Identificar necesidades.
2 Identificar las tareas.
3 Darle seguimiento al tamaño de los productos.
4 Darle seguimiento al tiempo de desarrollo.

Contenido de un plan de software

1 Tamaño del trabajo

Qué se va a hacer (tareas), qué tan grande es, cuánto tiempo tomará realizarlo.

2 Estructura del trabajo

Cómo se va a realizar el trabajo (tareas), en qué orden (que tarea primero y cual después), quién lo va a hacer, qué será entregado, cuándo, a qué precio

Seguimiento de un plan

3 Estado del trabajo

Cómo está evolucionando (el trabajo), ¿estamos adelantados, atrasados, en los costos?, ¿qué hacer para determinarlo?, ¿qué se hará si hay problemas?

4 Evaluación

Qué tan bueno es, fue el trabajo, qué errores se cometieron (de planeación y seguimiento), cómo se pueden evitar, qué se puede hacer para realizar un mejor trabajo.

Planeación de un Proyecto de Software [1/2]

1. Comenzar con una descripción explícita del trabajo que se requiere.

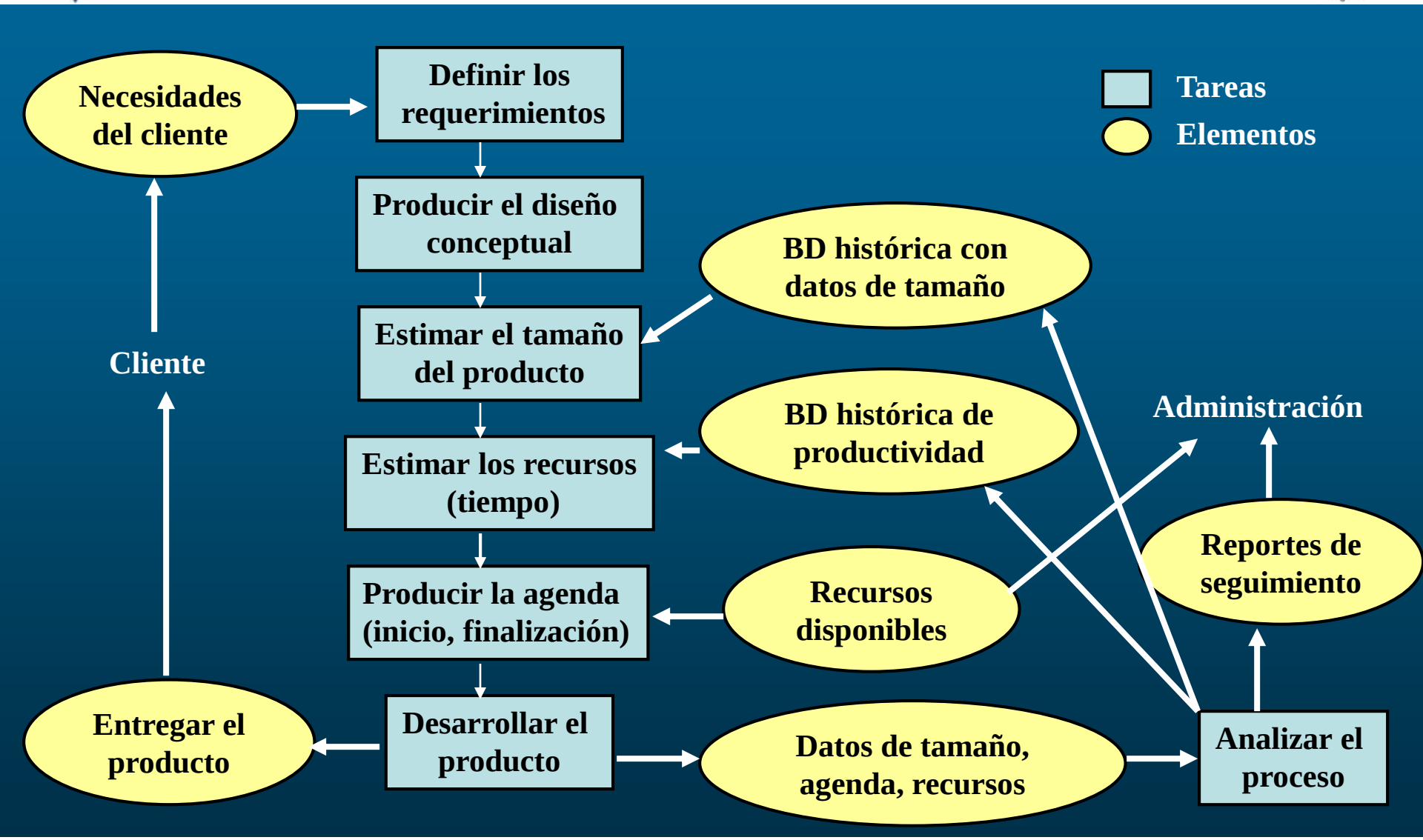
2. Para proyectos que tomen más de *pocos días*, *descomponerlo* en múltiples tareas pequeñas y estimar cada tarea por separado.

3. Basar las estimaciones en datos históricos.

4. Registrar las estimaciones y luego compararlas con los resultados reales.

Ver técnica de Estructura de Descomposición del Trabajo.

Marco de Trabajo [2/2]



Cualidades de un Plan [1/2]

1 Completo

La definición de un proceso de planeación facilita verificar este punto, ya que dice qué se tiene que hacer y qué formas se tienen que usar.

2 Accesible

Debe estar en un lugar disponible para todos aquellos que lo van a utilizar. En un formato adecuado y los datos en orden para que puedan ser localizados rápidamente.

Cualidades de un Plan [2/2]

3 Claro

Limpio, los datos correctos en el lugar correcto.

4 Específico

Qué se tiene que hacer, cuándo, por quién y a qué costo.

5 Preciso

Se refiere a relacionar la unidad de medida a la magnitud total de la medición.

6 Exacto

Para poder medir, predecir y mejorar la planeación.

Medición del tamaño del Producto

El proceso de planeación comienza con la estimación del tamaño del trabajo a realizar, de esta manera se descubre la cantidad de trabajo que se requiere.

Para poder estimar el tamaño, se requiere de un mecanismo consistente y repetible.

Medición de tamaño

Las mediciones deben ser :

1 Útiles para la planeación.

2 Deben ser precisas.

3 Factibles de ser automatizadas.

Medición de tamaño
Útiles para la planeación [1/5]

Se requieren más recursos para desarrollar programas grandes que para programas pequeños.

Se debe encontrar la correlación entre los datos que vamos a utilizar para la planeación.

Correlación es el grado en el cual 2 conjuntos de datos están relacionados.

r entre -1.0 y +1.0

$r^2 > 0.5$ para ser útil en la planeación (durante las estimaciones)

Medición de tamaño
Útiles para la planeación [2/5]

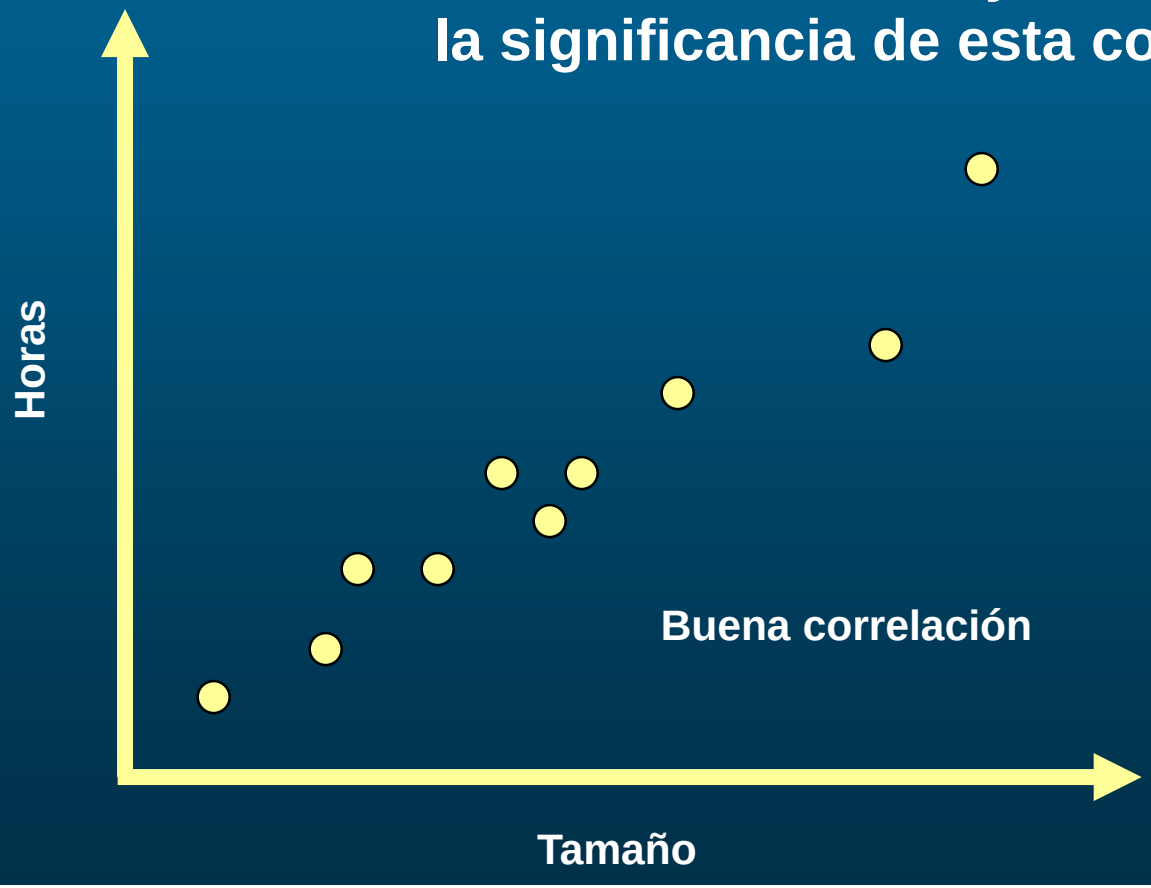
El grado en el cual la correlación es significativa también es importante, a esto se le llama significancia y es la probabilidad de que se hayan obtenido los resultados por casualidad.

Una significancia de 0.25 quiere decir que una cuarta parte del tiempo el resultado puede ocurrir por variaciones. Por otro lado, una significancia de 0.005 quiere decir que este resultado ocurriría 1 en 200 veces (1/0.005). Esta es una significancia muy alta.

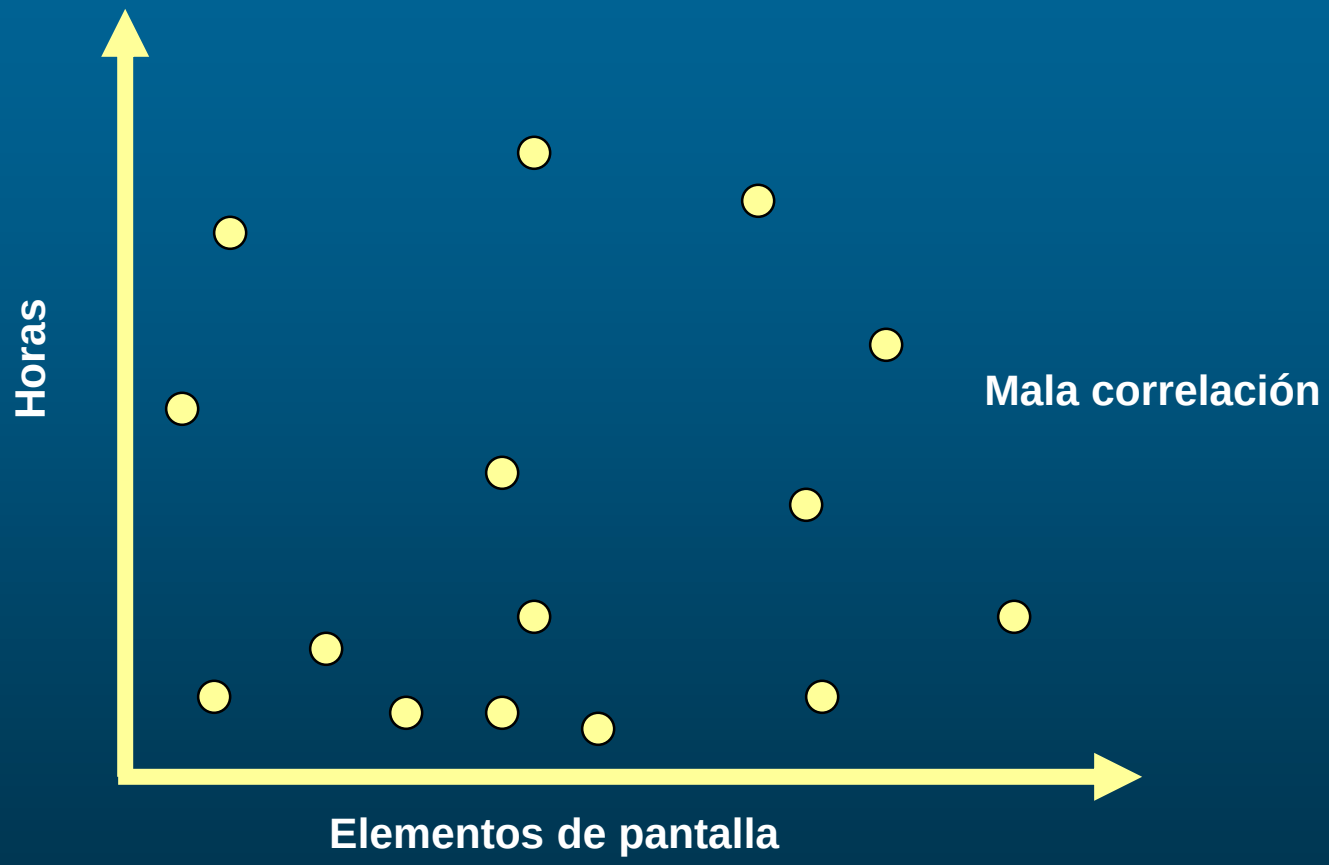
Una significancia de 0.05 es buena una de 0.20 es mala.

Medición de tamaño
Útiles para la planeación [3/5]

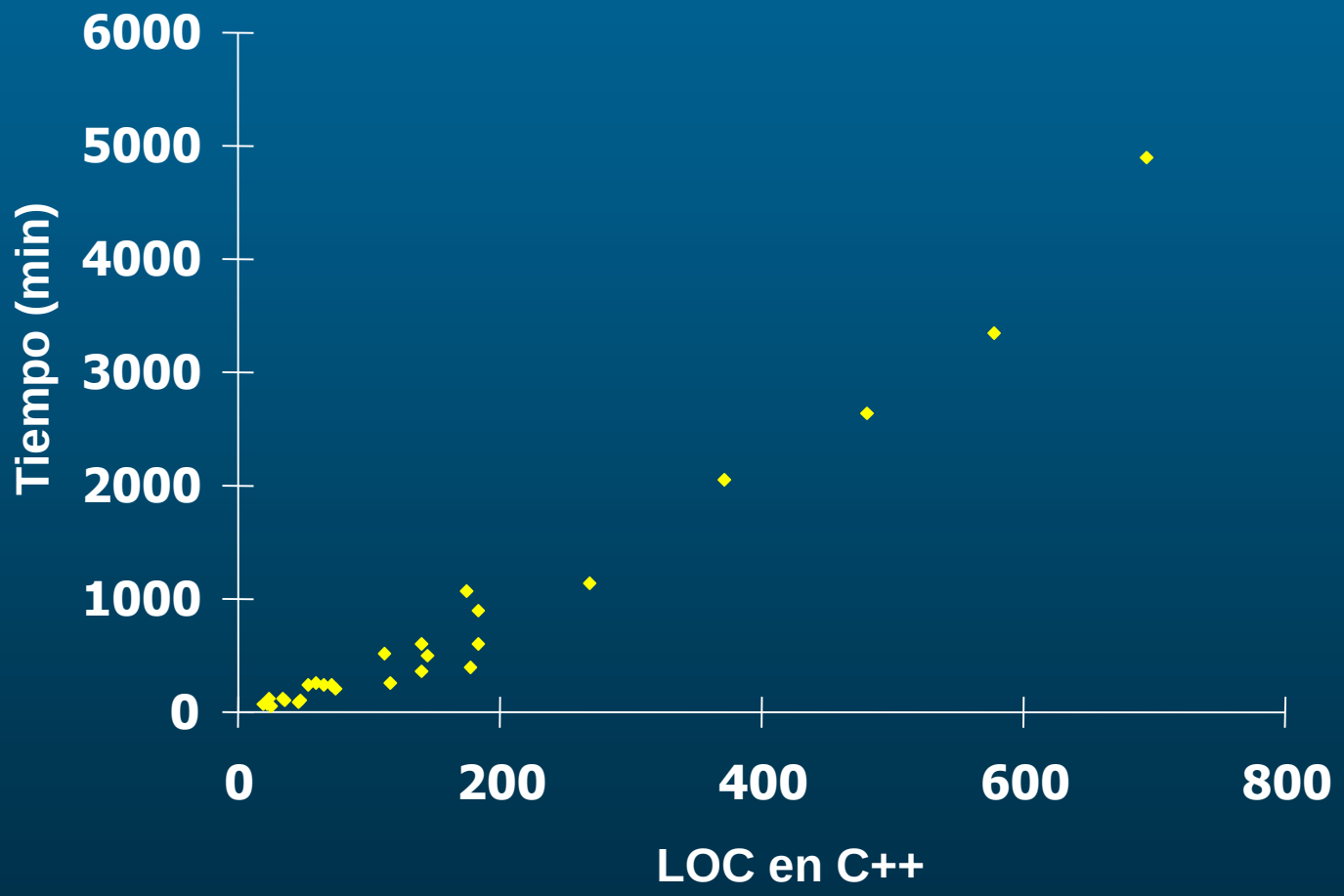
En la evaluación de las medidas de tamaño, estamos interesados en la correlación entre la medida del tamaño y las horas de desarrollo, y la significancia de esta correlación.



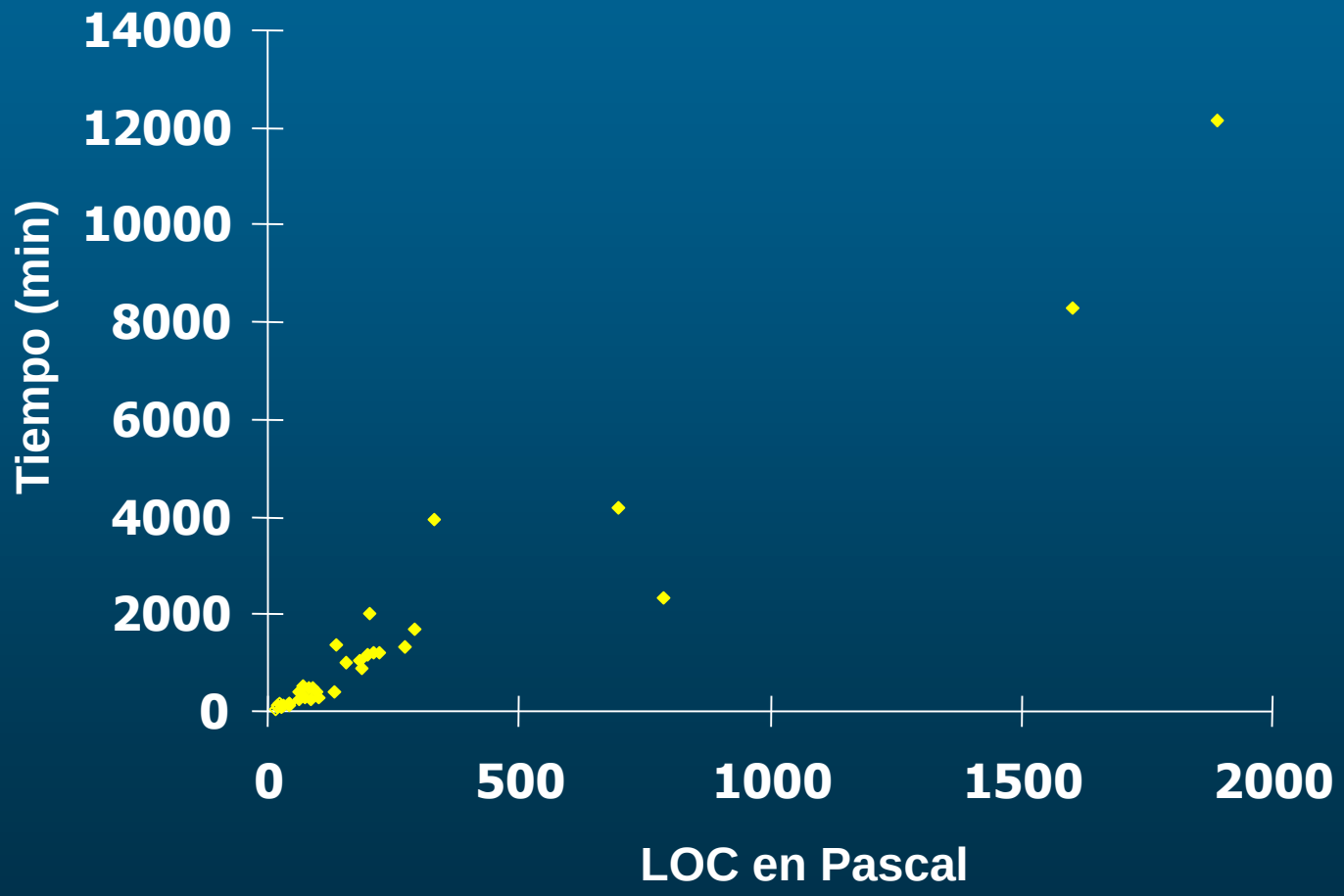
Medición de tamaño
Útiles para la planeación [4/5]



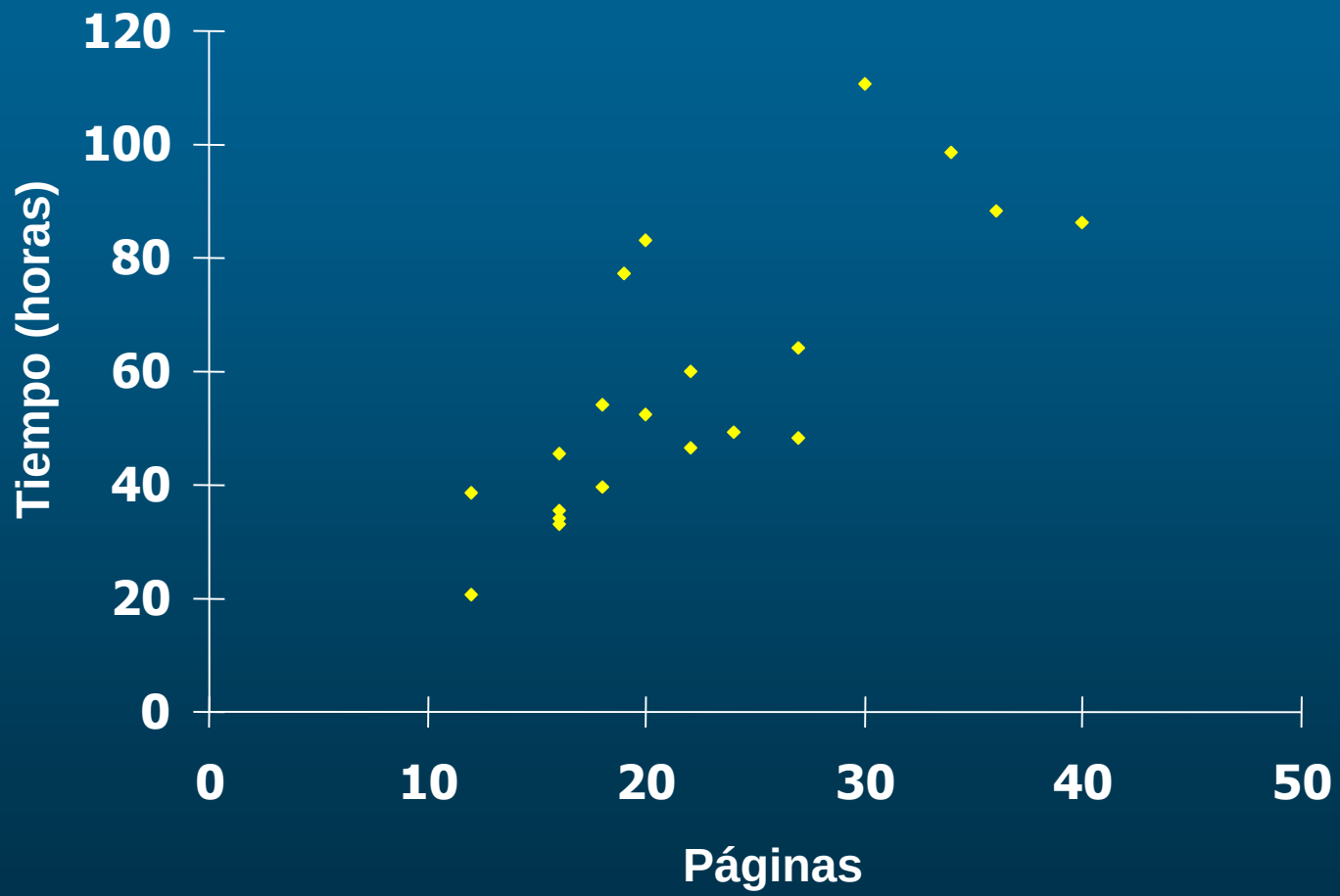
LOC en C++ vs. Tiempo de Desarrollo



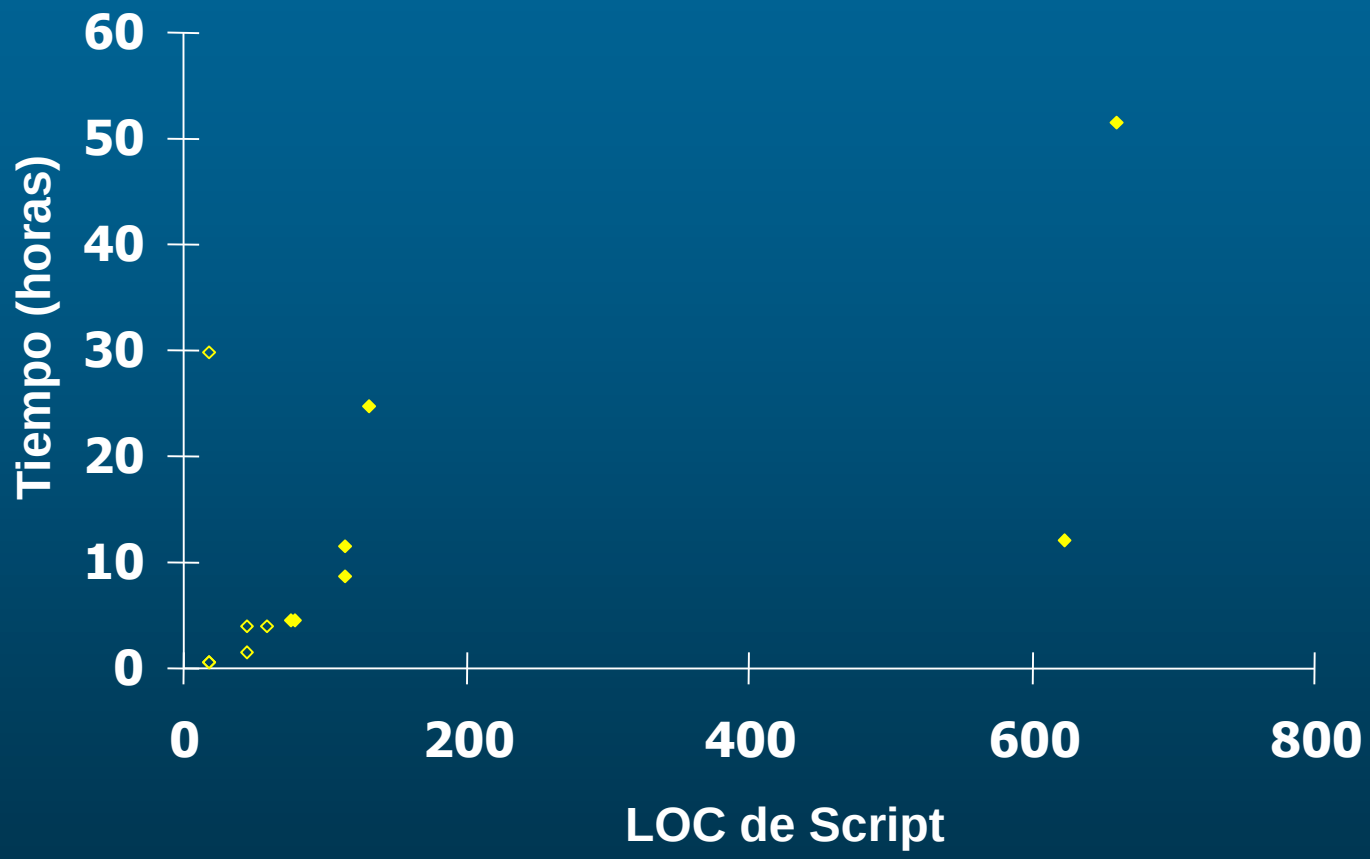
LOC en Pascal vs. Tiempo de Desarrollo



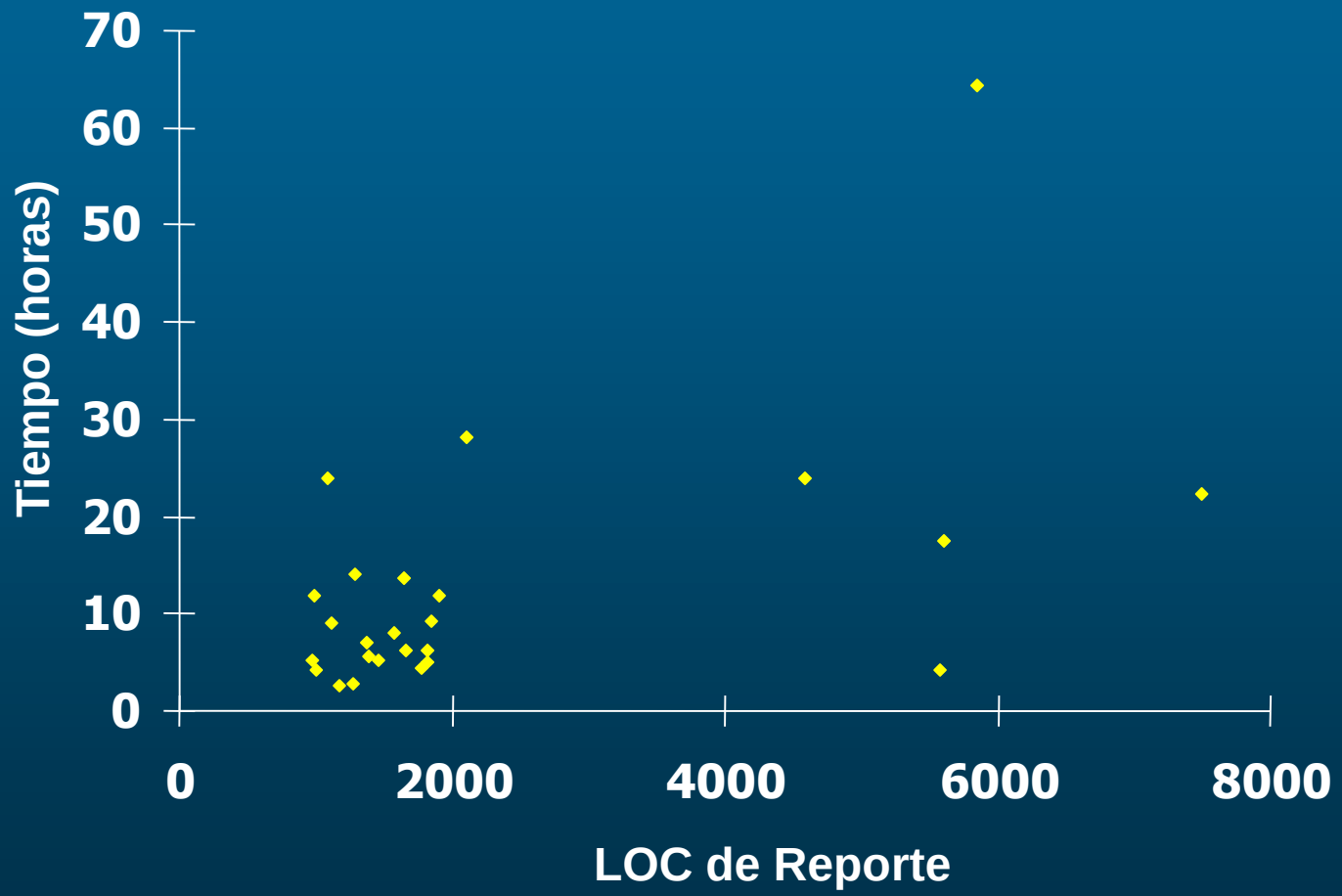
Páginas de Texto vs. Tiempo



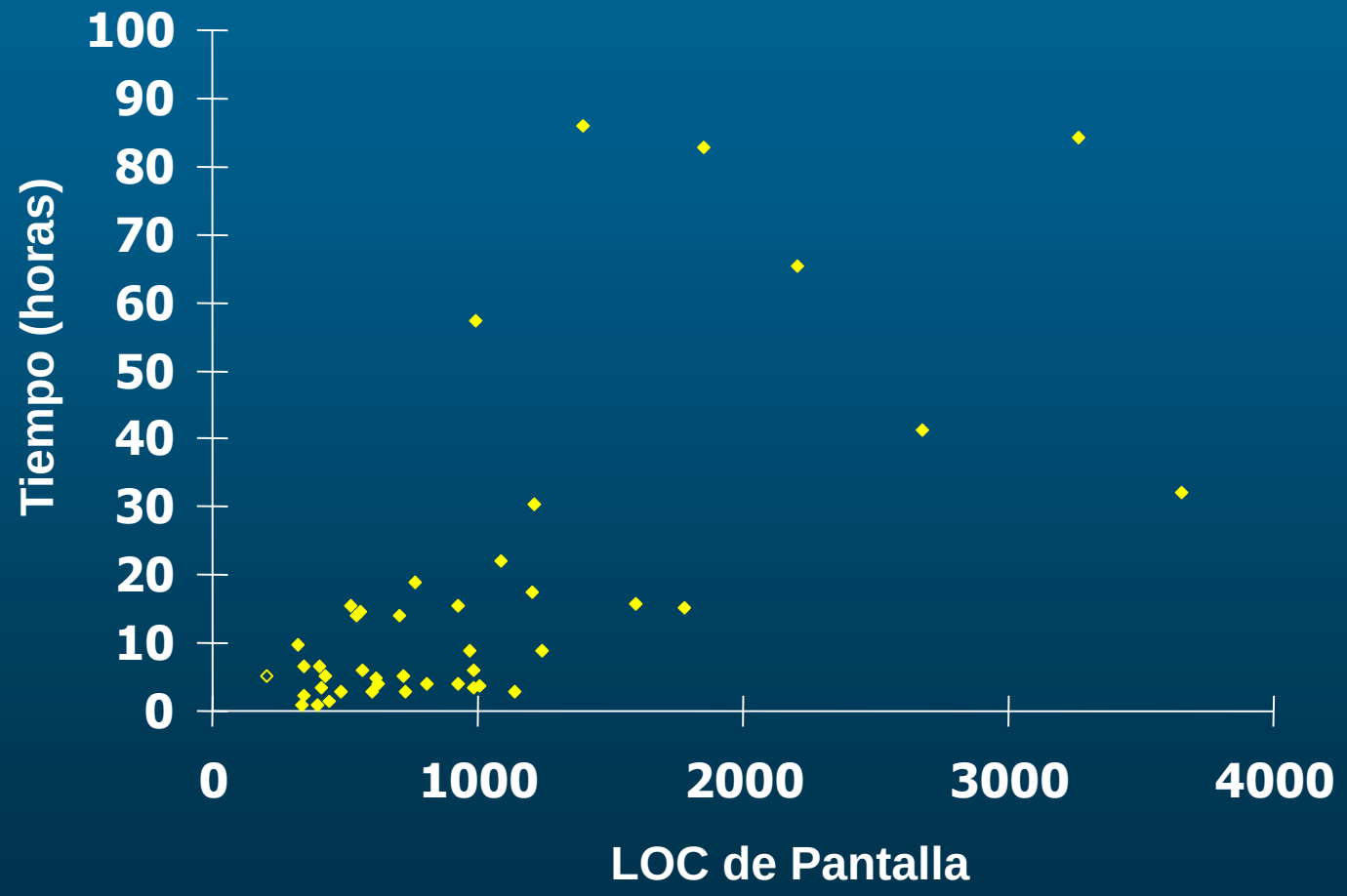
LOC de Script vs. Tiempo



LOC de Reporte vs. Tiempo



LOC de Pantalla vs. Tiempo



Medición de tamaño
Útiles para la planeación [5/5]

El grado por el cual es significativa la correlación también es importante, la significancia, esencialmente, la probabilidad de que se haya obtenido por casualidad.

Una significancia de 0.25 dice que una cuarta parte del tiempo se obtiene por fluctuaciones. Sin embargo una significancia de 0.005 dice que se obtiene por casualidad 1 en 200 veces. Esta última tiene una significancia alta.

Medición de tamaño
Precisión

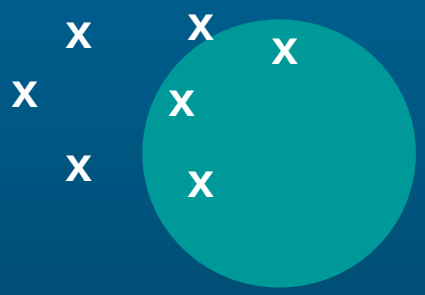
La precisión tiene que ver con el nivel de detalle (granularidad) en la medición.

La exactitud encierra la relación de una aseveración y el hecho real.

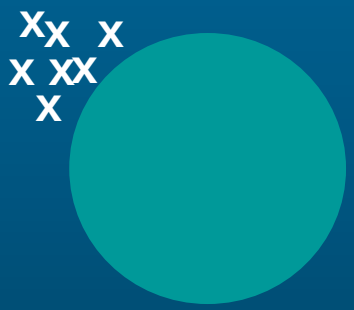
Cita a las 8:16 am (cita muy precisa)
Llegada a las 9:06 am (una llegada no muy exacta)

Precisión vs. Exactitud

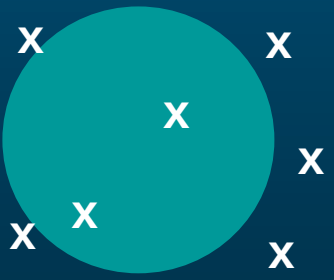
Impreciso e inexacto



Preciso e inexacto



Impreciso y exacto



Preciso y exacto

Tamaño vs. Esfuerzo de Desarrollo

- El requerimiento principal: si la medida del tamaño no está directamente relacionada al costo del desarrollo, no vale la pena usarla.

- Hay muchas posibles unidades de medida:
  - Líneas de código (LOC).
  - Puntos Funcionales.
  - Páginas, pantallas, scripts, reportes.

- La medida del tamaño debe ser sensible al lenguaje, diseño y prácticas de desarrollo.

Medición de tamaño
Conteo automático

Una vez seleccionada una medición de tamaño precisa, se requieren medios automáticos, económicos y exactos de conteo.

Marco de trabajo para la medición del tamaño

Hay que describir la forma de conteo, es personal y por cada lenguaje de programación.

Fácil de comunicar

Si se utiliza el mismo método, ¿podrán otros saber precisamente qué ha sido medido, incluido y excluido?

Repetible

¿Podrá alguien repetir la medición y obtener el mismo resultado?

Plantilla de estándar de conteo [1/2]

Ver.

Plantilla de estándar de conteo [2/2]

Ver ejemplo 1.

Establecimiento de un Estándar de Conteo

Existen diversos tipos de código:

- **Código de pruebas.**
- **Código de apoyo.**
- **Código de producto.**

Por lo que hay que separar cada uno de ellos al momento de contar.

Línea Física

Cada una de las líneas terminadas por un retorno de carro.

```
If A < B then
        C := C + 1
Else
        C := C – 1
```

4 LOC

Línea lógica

Cada construcción lógica

```
If A < B then
        C := C + 1
Else
        C := C – 1
```

1 LOC

Conteo de Programas

- Líneas lógicas.
  - Invariable a los cambios de edición.
  - Correlacionada con el esfuerzo de desarrollo.
  - Definición única.
  - Compleja de contar.

- Líneas Físicas.
  - Fácil de contar.
  - Variable.
  - No existe una definición única.

Anomalías en el Conteo

El incremento de datos en la base de datos históricos permite la compensación de las variaciones de tamaño de la LOC.

```
main()
ThisPage.PrintL(MyList.GetString(ThisData,1));
```

Estándar de Codificación [1/2]

Purpose:	To guide the development of C++ programs
Program Headers	All programs begin with a descriptive header.
Header Format	/********** /* Program Assignment: the program number */ /* Name: your name */ /* Date: the date program development started */ /* Description: a short description of the program */ /* function */ **********/
Listing Contents	Provide a summary of the listing contents.
Contents Example	/********** /* Listing Contents: */ /* Reuse instructions */ /* Includes */ /* Class declarations: */ /* CData */ /* ASet */ /* Source code in c:\classes\CData.cpp: */ /* CData */ /* CData() */ /* Empty() */ **********/
Reuse Instructions	Describe how the program is used. Provide the declaration format, parameter values and types, and parameter limits. Provide warnings of illegal values, overflow conditions, or other conditions that could potentially result in improper operation.
Example	/********** /* Reuse Instructions */ /* int PrintLine(char *line_of_character) */ /* Purpose: to print string, 'line_of_character', on one print line */ /* Limitations: the maximum line length is LINE_LENGTH */ /* Return: 0 if printer not ready to print, else 1 */ **********/

Estándar de Codificación [2/2]

Identifiers	Use descriptive names for all variables, 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; /* These are BAD */
Comments	Sufficiently 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 the records been processed? */
Bad Comment	if(record_count > limit) /* check if record_count is greater than limit */
Major Sections	Major program sections should be preceded by a block comment that describes the processing that is done in the next section
Example	/***/ /* This program section will examine the contents of the array "grades" */ /* and will calculate the average grade for the class. */ /***/

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	All defines are capitalized. All other identifiers and reserved words are lower case. Messages being output to the user can be mixed-case so as to make a clean user presentation.
Capitalization Example	#define DEFAULT_NUMBER_OF_STUDENTS 15 int class_size = DEFAULT_NUMBER_OF_STUDENTS;

No hay que comparar la productividad entre Ings. de Software

Ing. A	Ing. B
1322 LOC	743 LOC
Más legible	Menos tiempo
No necesariamente mejor programa	No necesariamente más rápido

Para que sirve el conteo de LOC

1. El empacado.

2. Medir, evaluar y/o predecir el trabajo.

3. Evaluar la calidad del producto.
    1. Determinar la calidad en todo o alguna parte del proceso de desarrollo.
    2. Para estimar el mantenimiento y soporte.

Tipos de LOC

1. Base (B)
2. Nuevo y Cambiado (N)
3. Reutilizable (R)
4. Borrado (D)
5. Modificado (M)
6. Total de LOC (T)
7. Añadido = A = T- B + D - R

Forma de Resumen del Plan del Proyecto PSP0.1

Ver.

Malas prácticas en el conteo

- Mezclar LOC de código nuevo, de pruebas de apoyo.
- Mezclar el conteo de diferentes tipos de lenguajes.
- Mezclar dos o más tipos de código.

Reglas de Conteo [1/4]

		Añadido	Substraído	Cambio Neto	Base
Base V0					0
	Borrado		0		
	Modificado	0	0		
	Añadido	350			
	Reusado	0			
	Total V0	350 -	0 =	350 + =	350
Base V1					350
	Borrado		0		
	Modificado	25	25		
	Añadido	100			
	Reusado	0			
	Total V1	125 -	25 =	100 + =	450
Base V2					450
	Borrado		200		
	Modificado	75	75		
	Añadido	50			
	Reusado	600			
	Total V2	725 -	275 =	450 + =	900
Producto Final					900

Reglas de Conteo [2/4]

1. Cada versión de programa comienza con algún número de LOC **base** que vienen de la versión anterior. La versión 0 tiene 0 LOC como base. Las LOC base de la versión 1, son las LOC totales de la versión 0.

2. Todas las adiciones a esta versión son registradas en los espacios marcados en la columna Añadido. LOC Reutilizadas, Añadidas y Modificadas.

Reglas de Conteo [3/4]

3. Todas las substracciones son registradas en los espacios de la columna Substracciones. Líneas modificadas y borradas

4. Sumar la cantidad de cada versión en las columnas Añadidas y substraídas.

5. Registrar el total añadido menos el total substraído en la columna cambio neto.

Reglas de Conteo [4/4]

6. Añadir este cambio neto a las LOC base de la versión para obtener el total.

7. Copiar este total de esta versión a la columna base de la siguiente versión o a la columna total de la fila producto finalizado

Productividad

Productividad = Cantidad de trabajo generado / hrs invertidas

		Añadido	Substraído	Cambio Neto	Base
Base V0					0
	Reusado	600			
	Modificado	100	100		
	Añadido	500			
	Borrado		200		
	Total V0	1200	- 300	= 900	+ = 900
Producto Final					900

	LOC	Productividad (60 hrs)
Añadido	500	8.33
Añadido + Modificado	600	10.00
Añadido + Borrado	800	13.33
Añadido + Modificado + Borrado	1200	20.00
Añadido + Modificado + Reusado	1400	23.33
Producto final	900	15.0

Scripts del Proceso PSP0.1

Ver.

Forma de Propuesta de Mejora del Proceso

Por cada ejercicio completado una forma de PIP debe ser completada e incluir la siguiente información:

- Descripción del problema.- Cualquier problema que hayas encontrado usando el proceso.

- Descripción de la propuesta.- Cualquier sugerencia que tengas para mejorar el proceso.

- Notas y comentarios.- Tus observaciones y hallazgos al hacer los ejercicios.

Ver.

Reporte R1

Producir un estándar de conteo de código.

Objetivos:

- Definir el estándar de conteo de líneas de código que sea apropiado al lenguaje de programación que uses.
- Proveer una base para el desarrollo de un estándar de codificación.
- Estar preparado para desarrollar un programa de conteo de líneas de código.

Usar la plantilla de estándar de conteo de LOC.

Reporte R2

Producir un estándar de codificación para ser usado en la escritura de los programas ejercicios de PSP.

Ver ejemplo de estándar de codificación.

Ejercicio 2A
Usar PSP0.1

Usar PSP0.1 para escribir el programa 2A para contar las líneas físicas de código de tus programas omitiendo los comentarios y las líneas en blanco.

Producir un solo resultado para el archivo fuente.

Prueba del ejercicio 2A

Cuenta manualmente las líneas de código de tus programas 1A y 2A y compáralos con el resultado de tu programa 2A. Utiliza la siguiente tabla para reportar tus resultados.

Número de programa	LOC
1A	
2A	

Orden de entrega de las formas

Resumen del Plan del Proyecto nivel PSP0.1.
FRPP Anterior.
Formas de PIPs.
Forma de Registro de Tiempos.
Forma de Registro de Defectos.
Código Fuente del Programa.
Reporte R1.
Reporte R2.
Pantallas de la interfase gráfica.
Pantallas de los resultados.
Tabla de resultados.