



**INSTITUTO TECNOLÓGICO SUPERIOR DE ALAMO TEMAPACHE**

**CARRERA:**

INGENIERIA EN SISTEMAS COMPUTACIONALES

**ASIGNATURA:**

PROCESO PERSONAL DE SOFTWARE (PSP)

**NOMBRE DE LA PRÁCTICA:**

USO DE PSP0.1

**INTEGRANTES DEL EQUIPO:**

GONZALO MARTINEZ SILVERIO

**DOCENTE:**

DRA. TANIA TURRUBIATES LOPEZ

**SEMESTRE:**

7

**GRUPO:**

701A

---



## **INTRODUCCIÓN:**

PSP (Personal Software Process) es un enfoque desarrollado por el Instituto de Ingeniería de Software de la Universidad Carnegie Mellon que tiene como objetivo mejorar la calidad y la productividad en el desarrollo de software a nivel individual. Se compone de varios componentes clave, como la planificación detallada, el seguimiento y medición del progreso, el diseño y codificación estructurados, pruebas rigurosas y la documentación detallada de actividades. El PSP ofrece beneficios significativos, incluyendo una mayor calidad del software, estimaciones más precisas, mayor productividad y desarrollo profesional para los ingenieros de software. En resumen, el PSP es un marco disciplinado que ayuda a los desarrolladores a mejorar sus habilidades y a entregar software de alta calidad de manera eficiente.

## **OBJETIVO:**

- Mejora de la calidad del software: El PSP proporciona una estructura y una metodología sólida para que los desarrolladores mejoren la calidad de su código y reduzcan la cantidad de defectos en el software final.
- Mejora de la productividad: Al adoptar prácticas y procesos más eficientes, los desarrolladores pueden aumentar su productividad y la velocidad a la que entregan software funcional y de calidad.
- Estimación precisa: El PSP proporciona herramientas para estimar con precisión el tiempo y los recursos necesarios para completar proyectos de desarrollo de software.

## **COMPETENCIA A DESARROLLAR:**

- Especificaciones de PSP y TSP.
- Que el alumno conozca su ritmo de trabajo y pueda hacer una evaluación del tiempo que tarda y con respecto a ello conozca su ritmo de trabajo en cada etapa.

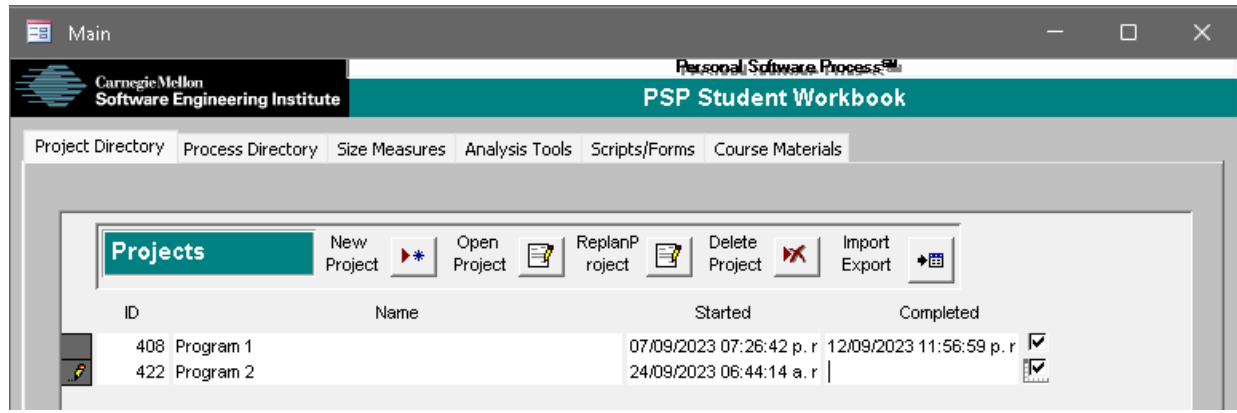
## **MATERIAL Y EQUIPO (REQUERIMIENTOS):**

- Computadora personal.
  - Dev C++
  - Material de práctica de PSP0.1.
  - Microsoft Office Word
-



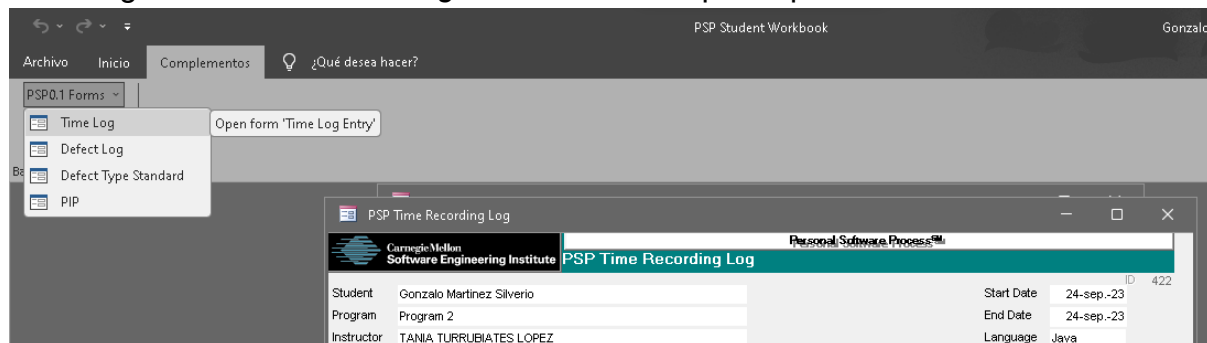
## DESARROLLO:

1. Registramos el inicio de la etapa de planeación dando doble *clic* en la barra blanca debajo de *Started* en la herramienta *PSP Student Workbook.mde*:



## ETAPA DE PLANEACION:

2. Registramos en el *time log* el inicio de la etapa de planeación:



3. El programa a realizar las siguientes condiciones:

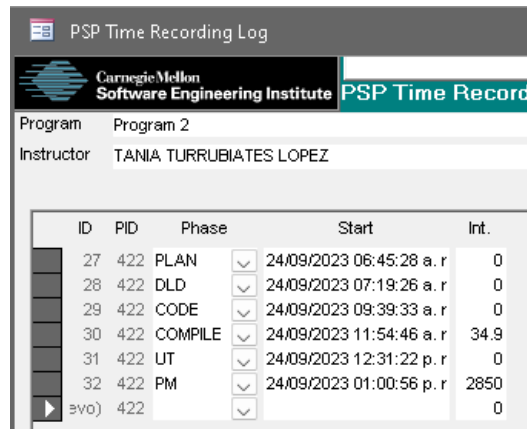
### 1.-Requerimientos del programa:

Usando PSP0.1, escribir un programa para contar (en LOC) lo siguiente:

- Tamaño total del programa.
- Tamaño total de cada una de las partes del programa (clases, funciones o procedimientos).
- El número de elementos (o métodos) en cada parte.
- 

Producir e imprimir:

- Un único conteo para el programa completo.
- Tamaño y conteo de elementos para cada parte junto con el nombre de la parte.

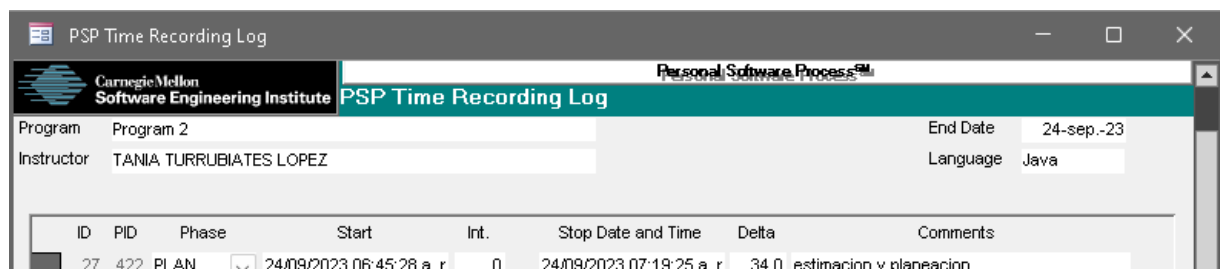


ID	PID	Phase	Start	Int.
27	422	PLAN	24/09/2023 06:45:28 a. r	0
28	422	DLD	24/09/2023 07:19:26 a. r	0
29	422	CODE	24/09/2023 09:39:33 a. r	0
30	422	COMPILE	24/09/2023 11:54:46 a. r	34.9
31	422	UT	24/09/2023 12:31:22 p. r	0
32	422	PM	24/09/2023 01:00:56 p. r	2850
33	422			0

## 2.-Estimación de tiempo:

El tiempo para este proyecto en todas sus etapas puesto que los requerimientos son bastante específicos será de 4 horas y 0 minutos.

## 3.- Registramos el cierre de la etapa de planeación registrándolo en el Time Log:

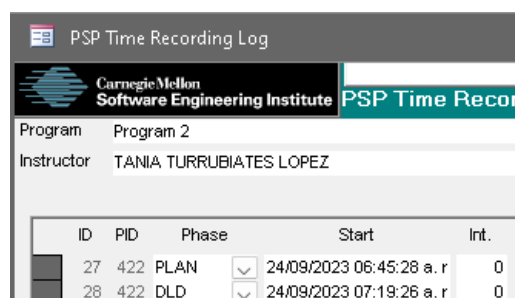


ID	PID	Phase	Start	Int.	Stop Date and Time	Delta	Comments
27	422	PLAN	24/09/2023 06:45:28 a. r	0	24/09/2023 07:19:25 a. r	34.0	estimacion y planeacion

## ETAPA DE DISEÑO:

### 1.Registramos el inicio de la etapa de diseño en el Time Log

:



ID	PID	Phase	Start	Int.
27	422	PLAN	24/09/2023 06:45:28 a. r	0
28	422	DLD	24/09/2023 07:19:26 a. r	0



## 2.- El diseño nos presentará lo siguiente:

Este diagrama de clases muestra la clase *Programa2* representa la lógica principal del programa. Contiene variables miembros para almacenar diversas métricas y realiza el análisis del archivo de código fuente en su método *main*.

El método *main* se encarga de leer el archivo de código fuente línea por línea, analizar cada línea y realizar el seguimiento de las métricas.

El método *contarMetricas* (que podría ser un método adicional) podría contener la lógica para contar las métricas, aunque en el código proporcionado, esta lógica se encuentra principalmente en el método *main*.

Las variables miembros representan las métricas que se están contando, como el número de funciones, procedimientos, clases, métodos y líneas de código. También hay una variable booleana para rastrear si se está procesando un comentario multilineal.

```
+-----+
| Programa2 |
+-----+
| - filename: String |
| - NumeroDeFunciones: |
| int |
| - NumeroDeProcedimie- |
| ntos: int |
| - NumeroDeClases: int |
| - NumeroDeMetodos: int |
| - NumeroDeLineasLOC: |
| int |
| - Comentarios: boolean |
+-----+
| + main(args: String[]): |
| void |
| + constructor() |
| + contarMetricas() |
+-----+
```

## 3.-Finalizamos la etapa de Diseño y la registramos en el Time Log:



ID	PID	Phase	Start	Int.	Stop Date and Time	Delta	Comments
27	422	PLAN	24/09/2023 06:45:28 a. r	0	24/09/2023 07:19:25 a. r	34.0	estimacion y planeacion
28	422	DLD	24/09/2023 07:19:26 a. r	0	24/09/2023 09:39:32 a. r	140.1	Diseño

## ETAPA DE CODIFICACION:

Registramos el inicio de la etapa de codificación en nuestro Time Log:

ID	PID	Phase	Start	Int.
27	422	PLAN	24/09/2023 06:45:28 a. r	0
28	422	DLD	24/09/2023 07:19:26 a. r	0
29	422	CODE	24/09/2023 09:39:33 a. r	0

2. Este código es un programa en Java que realiza un análisis simple de un archivo de código fuente Java para contar varios elementos, como clases, métodos, funciones, procedimientos y líneas de código. Aquí tienes una breve explicación de lo que hace el código:

Se define una serie de variables para llevar un registro de diferentes métricas, como el número de clases, métodos, funciones, procedimientos y líneas de código. También se define una variable booleana llamada Comentarios para controlar si el código está dentro de un comentario multilineal (`/* ... */`).

Se utiliza un bloque try-with-resources para abrir y leer un archivo de código fuente Java especificado en la variable filename. Se crea un objeto BufferedReader para leer el archivo línea por línea.

Se inicia un bucle while que recorre cada línea del archivo de código fuente.

Se eliminan los espacios en blanco al principio y al final de cada línea utilizando `Lines.trim()`.

Se verifica si la línea comienza con "import" o "package" y se omite, ya que son líneas de importación y no se cuentan como líneas de código real.

Se verifica si la variable Comentarios es verdadera, lo que significa que estamos dentro de un comentario multilineal. Si es así, se verifica si la línea contiene `*/` para determinar si hemos salido del comentario. Si hemos salido, se actualiza Comentarios a falso y se



## INSTITUTO TECNOLÓGICO SUPERIOR DE ALAMO TEMAPACHE

elimina el texto del comentario de la línea actual. Si aún estamos dentro del comentario, se continúa ignorando la línea.

Se verifica si la línea comienza con "/\*", lo que indica un comentario de una sola línea. En ese caso, se omite la línea.

Si la línea contiene "/\*", se verifica si también contiene "\*/" en la misma línea. Si es así, se elimina todo el comentario de la línea actual. Si no contiene "\*/", se marca que estamos dentro de un comentario (Comentarios se establece en verdadero) y se elimina el texto del comentario de la línea.

Se verifica si la línea no está vacía y no consiste solo en "{" o "}". En ese caso, se incrementa la variable NumeroDeLineasLOC para contar las líneas de código efectivas.

Se verifican diferentes condiciones para determinar si la línea contiene una declaración de clase, método, función o procedimiento, y se incrementan las variables correspondientes. Además, se imprime la línea actual junto con información sobre qué tipo de elemento se encontró y su número correspondiente.

El programa finaliza con la impresión de la cantidad total de clases, métodos y líneas de código.

```
Start Page x Programa2.java x
Source History
1 package PROGRAMA2;
2 /*
3 Programa para contar lineas de codigo(en LOC) lo siguiente:
4 Tamaño total del programa
5 Tamaño total de cada una de las partes del programa (clases,
6 funciones y procedimientos)
7 El número de elementos y métodos en cada parte
8
9 Produce e imprime:
10 Un único recuento para todo el programa
11 Tamaño y recuentos de elementos para cada parte, junto con el
12 nombre de la parte (clases, funciones y procedimientos)
13 */
14 /**
15  * @author Gonzalo
16  */
17 import java.io.BufferedReader;
18 import java.io.FileReader;
19 import java.io.IOException;
20
21 public class Programa2 {
22     public static void main(String[] args) {
23         String filename = "C:/Users/Gonzalo/Desktop/RelojDigital.java";
24
25         int NumeroDeFunciones = 0;
26         int NumeroDeProcedimientos = 0;
27         int NumeroDeClases = 0;
28         int NumeroDeMetodos = 0;
29         int NumeroDeLineasLOC = 0;
30         boolean Comentarios = false;
```



```
31
32     try (BufferedReader br = new BufferedReader(new FileReader(filename))) {
33         String Lineas;
34         while ((Lineas = br.readLine()) != null) {
35             Lineas = Lineas.trim();
36             // Comentarios líneas de importación y package
37             if (Lineas.startsWith("import") || Lineas.startsWith("package")) {
38                 continue;
39             }
40             if (Comentarios) {
41                 if (Lineas.contains("*/")) {
42                     Comentarios = false;
43                     Lineas = Lineas.substring(Lineas.indexOf("*/") + 2);
44                 } else {
45                     continue; // Ignorar líneas dentro del comentario
46                 }
47             }
48
49             if (Lineas.startsWith("//")) {
50                 continue; // Ignorar líneas de comentarios de una sola línea
51             }
52
53             if (Lineas.contains("/*")) {
54                 if (Lineas.contains("*/")) {
55                     Lineas = Lineas.substring(0, Lineas.indexOf("/*"))
56                     + Lineas.substring(Lineas.indexOf("*/") + 2);
57                 } else {
58                     Comentarios = true;
59                     Lineas = Lineas.substring(0, Lineas.indexOf("/*"));
60                 }
61             }
62
63             // No contar líneas que solo contienen " "
64             if (!Lineas.isEmpty() && !Lineas.equals("(") && !Lineas.equals(")")) {
65                 NumeroDeLineasLOG++;
66                 //System.out.println("Línea de código: " + Lineas);
67             }
68             if (Lineas.startsWith("public class") || Lineas.startsWith("class")) {
69                 NumeroDeClases++;
70
71                 System.out.println("Clase: \n" + Lineas);
72                 System.out.println("Numero de clase: " + NumeroDeClases);
73             }
74
75             else if (Lineas.startsWith("public ")
76                     || Lineas.startsWith("private")
77                     || Lineas.startsWith("protected")
78                     || Lineas.startsWith("void")) {
79                 NumeroDeMetodos++;
80                 System.out.println("Método: \n" + Lineas);
81                 System.out.println("Numero de método: " + NumeroDeMetodos);
82             }
83
84             if (Lineas.contains("public void")
85                     || Lineas.contains("public int")
86                     || Lineas.contains("public double")
87                     || Lineas.contains("public String")) {
88                 NumeroDeFunciones++;
89             }
90         }
91     }
92 }
```





```
90      System.out.println("Función: " + Lineas);
91      System.out.println("Numero de funcion: " + NumeroDeFunciones );
92  }
93  else if ( Lineas.contains("public static int")
94      || Lineas.contains("public static double")
95      || Lineas.contains("public static String")) {
96      NumeroDeProcedimientos++;
97      System.out.println("Procedimiento: \n" + Lineas);
98      System.out.println("Numero de procedimiento: " + NumeroDeProcedimientos);
99  }
100 }
101 } catch (IOException e) {
102     e.printStackTrace();
103 }
104 System.out.println("\nCantidad total de clases: " + NumeroDeClases);
105 System.out.println("Cantidad total de métodos: " + NumeroDeMetodos);
106 System.out.println("Cantidad total de líneas de código (LOC): " +
107 );
108 }
109
110
```

3.-Marcamos como finalizada la etapa de codificación en el Time Log:

PSP Time Recording Log

Carnegie Mellon Software Engineering Institute

Personal Software Process<sup>SM</sup>

Program: Program 2

Instructor: TANIA TURRUBIATES LOPEZ

End Date: 24-sep.-23

Language: Java

ID	PID	Phase	Start	Int.	Stop Date and Time	Delta	Comments
27	422	PLAN	24/09/2023 06:45:28 a. r	0	24/09/2023 07:19:25 a. r	34.0	estimacion y planeacion
28	422	DLD	24/09/2023 07:19:26 a. r	0	24/09/2023 09:39:32 a. r	140.1	Diseño
29	422	CODE	24/09/2023 09:39:33 a. r	0	24/09/2023 10:54:44 a. r	75.2	Codificacion en netbeans 8.2

## ETAPA DE COMPILACION:

PSP Time Recording Log

Carnegie Mellon Software Engineering Institute

Personal Software Process<sup>SM</sup>

Program: Program 2

Instructor: TANIA TURRUBIATES LOPEZ

End Date: 24-sep.-23

Language: Java

ID	PID	Phase	Start	Int.	Stop Date and Time	Delta	Comments
27	422	PLAN	24/09/2023 06:45:28 a. r	0	24/09/2023 07:19:25 a. r	34.0	estimacion y planeacion
28	422	DLD	24/09/2023 07:19:26 a. r	0	24/09/2023 09:39:32 a. r	140.1	Diseño
29	422	CODE	24/09/2023 09:39:33 a. r	0	24/09/2023 10:54:44 a. r	75.2	Codificacion en netbeans 8.2
30	422	COMPILE	24/09/2023 11:54:46 a. r	34.9	24/09/2023 12:30:21 p. r	0.7	Copilacion del programa, int almuerzo



## INSTITUTO TECNOLÓGICO SUPERIOR DE ALAMO TEMAPACHE

Se compila:

BUILD SUCCESSFUL (total time: 2 seconds)

Compilación antes finalizar esta etapa se decidió hacer una prueba ejecutando el programa e introduciéndole datos y no se encontró algún error el cual procederé a solucionar en esta siguiente etapa de pruebas:

### ETAPA DE TESTEO:

Registramos el inicio de nuestra etapa de Testeo:

ID	PID	Phase	Start	Int.
27	422	PLAN	24/09/2023 06:45:28 a. r	0
28	422	DLD	24/09/2023 07:19:26 a. r	0
29	422	CODE	24/09/2023 09:39:33 a. r	0
30	422	COMPILE	24/09/2023 11:54:46 a. r	34.9
31	422	UT	24/09/2023 12:31:22 p. r	0

### **AHORA PROCEDEMOS A EJECUTAR EL PROGRAMA:**

Nuestro programa leerá el siguiente programa y nos mostrará el número de métodos, clases y líneas de código LOC.

```
RelojDigital.java
Archivo  Editar  Ver

package horas;
/**
 *
 * @author Gonzalo
 */
import javax.swing.*;
import java.awt.*;
import java.text.SimpleDateFormat;
import java.util.Date;

public class RelojDigital extends JPanel implements Runnable {
    private Thread thread;

    public RelojDigital() {
        thread = new Thread(this);
        thread.start();
    }

    public void run() {
        while (true) {
            repaint();
            try {
```



## INSTITUTO TECNOLÓGICO SUPERIOR DE ALAMO TEMAPACHE

```
        Thread.sleep(1000); // Actualizar cada segundo
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
@Override
protected void paintComponent(Graphics g) {
    super.paintComponent(g);
    SimpleDateFormat sdf = new SimpleDateFormat("HH:mm:ss");
    String time = sdf.format(new Date());

    String time2 = sdf.format(new Date());
    Font font = new Font("Arial", Font.BOLD, 58);





    g.setColor(Color.BLACK);
    g.fillRect(0, 0, getWidth(), getHeight());

    g.setColor(Color.WHITE);
    g.setFont(font);
    g.drawString(time, 100, 100);
}

public static void main(String[] args) {
    SwingUtilities.invokeLater(() -> {
        JFrame frame = new JFrame("Reloj Digital");
        frame.setSize(400, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.add(new RelojDigital());
        frame.setVisible(true);
    });
}
```

Resultados:

**Output - PSP (run)** X

```
run:
Clase:
public class RelojDigital extends JPanel implements Runnable {
Numero de clase: 1
Método:
private Thread thread;
Numero de método: 1
Método:
public RelojDigital() {
Numero de método: 2
Método:
public void run() {
Numero de método: 3
Función: public void run() {
Numero de funcion: 1
Método:
protected void paintComponent(Graphics g) {
Numero de método: 4
Método:
public static void main(String[] args) {
Numero de método: 5

Cantidad total de clases: 1
Cantidad total de métodos: 5
Cantidad total de líneas de código (LOC): 31
BUILD SUCCESSFUL (total time: 0 seconds)
```



## INSTITUTO TECNOLÓGICO SUPERIOR DE ALAMO TEMAPACHE

Los resultados fueron verificados y son correctos, por lo tanto, marcarse como finalizada la etapa de pruebas:

PSP Time Recording Log

Program: Program 2  
Instructor: TANIA TURRUBIATES LOPEZ  
End Date: 24-sep.-23  
Language: Java

ID	PID	Phase	Start	Int.	Stop Date and Time	Delta	Comments
27	422	PLAN	24/09/2023 06:45:28 a. r	0	24/09/2023 07:19:25 a. r	34.0	estimacion y planeacion
28	422	DLD	24/09/2023 07:19:26 a. r	0	24/09/2023 09:39:32 a. r	140.1	Diseño
29	422	CODE	24/09/2023 09:39:33 a. r	0	24/09/2023 10:54:44 a. r	75.2	Codificacion en netbeans 8.2
30	422	COMPILE	24/09/2023 11:54:46 a. r	34.9	24/09/2023 12:30:21 p. r	0.7	Copilacion del programa, int almuerzo
31	422	UT	24/09/2023 12:31:22 p. r	0	24/09/2023 01:00:55 p. r	29.6	Pruebas y correccion de errores

Iniciamos la etapa del POSMORTEM

PSP Time Recording Log

Student: Gonzalo Martinez Silverio  
Program: Program 1  
Instructor: TANIA TURRUBIATES LOPEZ

ID	PID	Phase	Start	Int.
16	408	PLAN	07/09/2023 07:27:42 p. r	0
17	408	DLD	07/09/2023 07:40:25 p. r	20
18	408	CODE	07/09/2023 08:59:34 p. r	10
25	408	PM	07/09/2023 09:59:51 p. r	2689
22	408	COMPILE	09/09/2023 09:59:51 p. r	0
23	408	UT	09/09/2023 10:00:16 p. r	31
26	408	PM	09/09/2023 10:55:36 p. r	4330

### **ETAPA POSMORTEM:**

En el contexto del desarrollo de este programa haremos una revisión o análisis retrospectivo de nuestro proyecto, con el objetivo de aprender de los éxitos y fracasos y mejorar en futuros proyectos.

Haremos un análisis profundo de nuestro proceso de desarrollo que realizamos anteriormente enfocado en cada etapa, evaluando nuestro desempeño y los resultados obtenidos:



## **ETAPA DE PLANEACION:**

Inicie con el análisis de los requerimientos del programa y comprendí lo que se nos pedía desarrollar una herramienta simple para analizar archivos de código fuente Java y contar métricas básicas, como clases, métodos, funciones, procedimientos y líneas de código., en base a esto elabore una estimación del tiempo que tardare en realizar el programa.

## **ETAPA DE DISEÑO:**

Gracias a lo comprendido anteriormente procedí a elaborar el diagrama de clases ya que permiten representar de manera visual los objetos que forman parte de un sistema de software. Cada clase en el diagrama representa un tipo de objeto, y las instancias de esas clases son los objetos reales que se crearán y utilizarán en el programa, esto anterior me permite tener una idea de cómo resolver el problema.

## **ETAPA DE CODIFICACION:**

La simplicidad del programa lo hace adecuado para un análisis rápido de código fuente, pero no para análisis en profundidad.

Para proyectos futuros, considerare la posibilidad de diseñar una estructura de datos que almacene los resultados para su posterior procesamiento o exportación.

## **ETAPA DE COMPILACION:**

En esta etapa el programa no tuvo complicaciones.

## **ETAPA DE PRUEBAS:**

En esta etapa de pruebas solo se verifíco si existía algún tipo de errores de los cuales solo fueron de documentación y de sintaxis lo que no causo errores graves, pero fueron corregidas con éxito y se agregó comentarios acerca de cómo funciona mi proyecto.

## **EVALUACION DE RESULTADOS:**

### **OBJETIVO DEL PROYECTO:**

El objetivo del proyecto Programa2 era desarrollar una herramienta simple para analizar archivos de código fuente Java y contar métricas básicas, como clases, métodos, funciones, procedimientos y líneas de código.

---



### ÉXITOS:

1. Funcionalidad Básica: El programa pudo contar con éxito las métricas básicas en archivos de código fuente Java proporcionados como entrada.
2. Manejo de Comentarios: El programa manejó de manera efectiva los comentarios de una línea y los comentarios multilínea dentro del código fuente.
3. Interfaz de Consola Clara: La interfaz de consola proporcionó información detallada sobre las métricas y el proceso de análisis.

### DESAFÍOS Y APRENDIZAJES:

1. Limitaciones en el Análisis: El programa tenía limitaciones en su capacidad para analizar de manera exhaustiva archivos de código fuente Java más complejos. No manejaba clases anidadas ni otros aspectos avanzados de la estructura de Java.
2. Almacenamiento de Resultados: Aunque mostraba información en la consola, no almacenaba los resultados en una estructura de datos que pudiera utilizarse para análisis posteriores o exportación.
3. Dependencia del Formato del Código: El programa dependía en gran medida del formato del código fuente Java, lo que podría llevar a resultados inexactos si el código no sigue un formato estándar.

### LECCIONES APRENDIDAS:

1. La simplicidad del programa lo hace adecuado para un análisis rápido de código fuente, pero no para análisis en profundidad.
2. Para proyectos futuros, considerar la posibilidad de diseñar una estructura de datos que almacene los resultados para su posterior procesamiento o exportación.
3. Es importante tener en cuenta las limitaciones y dependencias en el formato del código cuando se desarrollan herramientas de análisis de código fuente.

### ACCIONES FUTURAS:

1. Explorar bibliotecas o herramientas más robustas de análisis de código fuente de Java para casos de uso más avanzados.
  2. Implementar una opción para exportar los resultados a un archivo o base de datos para un análisis posterior.
  3. Mejorar la capacidad de manejo de excepciones para garantizar que el programa sea más robusto ante errores en archivos de entrada inesperados.
-



## FINAL DEL POSMORTEM

PSP Time Recording Log									
Carnegie Mellon Software Engineering Institute									
PSP Time Recording Log									
Student		Gonzalo Martinez Silverio					Start Date		24-sep.-23
Program		Program 2					End Date		26-sep.-23
Instructor		TANIA TURRUBIATES LOPEZ					Language		Java
ID	PID	Phase	Start	Int.	Stop Date and Time	Delta	Comments		
27	422	PLAN	24/09/2023 06:45:28 a. r	0	24/09/2023 07:19:25 a. r	34.0	estimacion y planeacion		
28	422	DLD	24/09/2023 07:19:26 a. r	0	24/09/2023 09:39:32 a. r	140.1	Diseño		
29	422	CODE	24/09/2023 09:39:33 a. r	0	24/09/2023 10:54:44 a. r	75.2	Codificacion en netbeans 8.2		
30	422	COMPILE	24/09/2023 11:54:46 a. r	34.9	24/09/2023 12:30:21 p. r	0.7	Copilacion del programa, int almuerzo		
31	422	UT	24/09/2023 12:31:22 p. r	0	24/09/2023 01:00:55 p. r	29.6	Pruebas y correccion de errores		
32	422	PM	24/09/2023 01:00:56 p. r	3150	26/09/2023 07:14:37 p. r	103.7	int para asuntos personales y escolares		
33	422	PM	26/09/2023 07:14:37 p. r	0	26/09/2023 07:14:37 p. r	0.0			

## RESULTADOS:

Los datos mostrados son confiables ya que el programa respeta cada operación de las fórmulas correspondientes.

PSP Time Recording Log									
Carnegie Mellon Software Engineering Institute									
PSP Time Recording Log									
Student		Gonzalo Martinez Silverio					Start Date		24-sep.-23
Program		Program 2					End Date		26-sep.-23
Instructor		TANIA TURRUBIATES LOPEZ					Language		Java
ID	PID	Phase	Start	Int.	Stop Date and Time	Delta	Comments		
27	422	PLAN	24/09/2023 06:45:28 a. r	0	24/09/2023 07:19:25 a. r	34.0	estimacion y planeacion		
28	422	DLD	24/09/2023 07:19:26 a. r	0	24/09/2023 09:39:32 a. r	140.1	Diseño		
29	422	CODE	24/09/2023 09:39:33 a. r	0	24/09/2023 10:54:44 a. r	75.2	Codificacion en netbeans 8.2		
30	422	COMPILE	24/09/2023 11:54:46 a. r	34.9	24/09/2023 12:30:21 p. r	0.7	Copilacion del programa, int almuerzo		
31	422	UT	24/09/2023 12:31:22 p. r	0	24/09/2023 01:00:55 p. r	29.6	Pruebas y correccion de errores		
32	422	PM	24/09/2023 01:00:56 p. r	3150	26/09/2023 07:14:37 p. r	103.7	int para asuntos personales y escolares		
33	422	PM	26/09/2023 07:14:37 p. r	0	26/09/2023 07:14:37 p. r	0.0			



## BITACORA DE DEFECTOS

PSP Defect Recording Log

Carnegie Mellon Software Engineering Institute Personal Software Process<sup>SM</sup>

Student Gonzalo Martinez Silverio Start Date 24-sep.-23 ID 422

Program Program 2 End Date 26-sep.-23

Instructor TANIA TURRUBIATES LOPEZ Language Java

ID	PID	Date	Type	Phase Injected	Phase Removed	Fix Time	FixDefect
10	422	24/09/2023	10-Documentation	CODE	UT	5	2
Defect Description		Error de no colocar comentarios acerca de las funciones de cada parte del programa					
11	422	24/09/2023	20-Syntax	CODE	UT	5	2
Defect Description		Ortografia					

## PROPUESTAS DE MEJORA DE PROCESOS

PSP Process Improvement Proposal

Carnegie Mellon Software Engineering Institute Personal Software Process<sup>SM</sup>

UID #422Nor Date 24-sep.-23

**Problem Description**  
Briefly describe the problems you encountered.

Encontre algunos problemas los cuales no fueron bien escritas algunas declaraciones de variables y no haber agregado comentarios describiendo mi código.

**Proposal Description**  
Briefly describe the process improvements that you propose.

Propongo estudiar y observar mas detalle lo que se escribe ya que esto dificulta el aprendizaje y entendimiento del código.

**Other Notes and Comments**  
Note any other comments or observations that describe your experiences or improvement ideas.

Es importante tener en cuenta que se puede mejorar siempre y cuando la persona misma tenga ese objetivo.

Registro: 1 de 1 Sin filtro Buscar





PSP0.1 Project Plan Summary

Carnegie Mellon

Software Engineering Institute

PSP0.1 Project Plan Summary

Personal Software Process<sup>SM</sup>

StudentGonzalo Martinez Silverio

ProgramProgram 2

InstructorTANIA TURRUBIATES LOPEZ

Start Date24-sep.-23

End Date26-sep.-23

LanguageJava

ID 422

Program Size Summary

LOC-Lines of code

	Plan Size	Actual Size	To Date
Base (B)		0.00	
Deleted (D)		0.00	
Modified (M)		0.00	
Added (A)		0.00	
Reused (R)		0.00	0.00
Added & Modified (A&M)	0.00	0.00	0.00
Total (T)		0.00	0.00
New Reusable (NR)		0.00	0.00

Time in Phase

Phase	Plan	Actual	To-Date	To-Date%
PLAN	13	34	13	5.2%
DLD	61	140	59	24.3%
CODE	52	75	50	20.7%
COMPILE	0	1	0	0.2%
UT	25	30	24	10.0%
PM	99	104	96	39.6%
Total	250	383	243	

Defects Injected in Phase

Phase	Plan	Actual	To-Date	To-Date%
PLAN		0	0	0.0%
DLD		0	1	50.0%
CODE		2	1	50.0%
COMPILE		0	0	0.0%
UT		0	0	0.0%
PM		0	0	0.0%
Total		2	2	

Defects Removed in Phase

Phase	Plan	Actual	To-Date	To-Date%
PLAN		0	0	0.0%
DLD		0	0	0.0%
CODE		0	0	0.0%
COMPILE		0	0	0.0%
UT		2	2	100.0%
PM		0	0	0.0%
Total		2	2	

Registro: 1 de 1

Filtrado

Buscar



## **CONCLUSIONES:**

PSP me ayuda a mejorar mi habilidad individual como desarrollador de software y en la gestión de proyectos de desarrollo de software de manera más eficiente. Ahora puedo concluir que debemos mejorar la calidad del software y la productividad de los desarrolladores mediante la adopción de procesos más disciplinados y la medición continua del rendimiento personal. Esto implica el uso de técnicas y herramientas específicas para seguir y mejorar el proceso de desarrollo de software a nivel individual.

## **BIBLIOGRAFÍA (Formato APA):**

- Watts S. Humphrey (2005). PSP A Self-Improvement Process for Software Engineers. Addison-Wesley Professional.
- PSP Academic Material (2016). Acceso el 20 de Agosto de 2016 desde Team Software Process. Software Engineering Institute, Carnegie Mellon University. Sitio Web: <http://www.sei.cmu.edu/tsp/tools/academic/index.cfm>