UNIVERSIDAD AUTÓNOMA DE YUCATÁN

FACULTAD DE MATEMATICAS

SEMESTRE: AGOSTO-DICIEMBRE

Desarrollo Y Mantenimiento de Software

Proyecto Final: parte 3

M. en C. Carlos Benito Mojica Ruiz.

Alumnos:

Jimena Nohemi Cruz Arreola

Luis Manuel Palma Pinto

Juan Pablo Rodríguez Falcon

Angel Mariel Osalde Salazar

Gabriel Precenda Valle

Contenido

Unidad 1 2
Estándar de Conteo 2
Consideraciones para el conteo de clases y métodos 5
Consideraciones para conteo de líneas, específicamente para control de versiones 6
Ejemplos básicos de cómo se aplica el Conteo de Líneas 6
Estándar de Codificación 7
Manual de Usuario 12
Manejo de excepciones, posibles errores comunes 15
Unidad 2 16
Casos de Prueba 16
Pruebas Unitarias 17
Pruebas de Integración 22
Pruebas de regresión 24
Unidad 3 27
Estimación de Tamaño del Software 27
Unidad 4. Mantenimiento 31
Decisiones y Modificaciones 32
Principales Decisiones y Modificaciones 32
Unidad 5: Aseguramiento de la calidad 35
Revisión de calidad 35
Metodología del aseguramiento de la calidad. 35
Enfoque 35
Roles y responsabilidades 36
Actividades del aseguramiento de la calidad 37
Calidad técnica 39

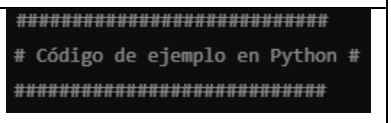
Unidad 1

Estándar de Conteo

El presente estándar de conteo tiene como objetivo proporcionar una guía clara y precisa para la identificación y clasificación de líneas de código en categorías lógicas o físicas, así como para definir la contabilización de clases y métodos dentro del programa. Además, establece criterios específicos para clasificar las líneas añadidas, eliminadas, movidas y modificadas. Solo se considerarán aquellas sentencias que estén marcadas con un 'Sí' en la columna 'Incluir'.

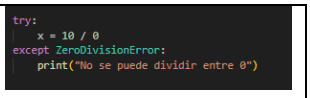
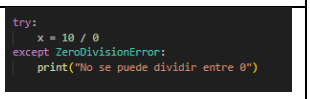
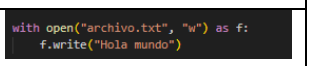
Estándar de conteo para códigos escritos en Python:

Tipo de conteo Físico / Lógico	**Tipo**	**Lenguaje**	
	Físico	Python	
Tipo de sentencia	**Incluir**	**Comentarios**	**Ejemplo (Opcional)**
Ejecutables	Sí		
No ejecutables:			
Declaraciones	Sí	Pueden ser declaraciones de funciones, variables, etc.	`x = 10  # Declaración de variable` `y = 20`
Declaraciones multilínea	Sí	Las declaraciones multilínea se tomarán en consideración como	`numeros = [` `  1, 2, 3, 4, 5,  # Primera fila` `  6, 7, 8, 9, 10,  # Segunda fila` `  11, 12, 13, 14, 15  # Tercera fila` `]`

		una línea física en toda su extensión. Es decir, si una declaración utiliza 4 líneas, las 4 serán consideradas como físicas.	
Comentarios		Se tomará en consideración que un comentario utiliza # (simple) o “”” “”” (multilínea); apenas se encuentre uno de estos indicadores se ignorará la línea del código.	 
- En una sola línea	No		
- Multilínea	No		
Líneas en blanco o vacías	No	Se considerará como línea vacía o en blanco si no tiene ni un solo carácter, incluso si son espacios en blanco o tabulaciones.	
Importaciones	Sí		
Decoradores	Sí		
Palabras reservadas	No	Las palabras reservadas como tal no se toman en consideración para agregar al conteo. Caso contrario que sucede en	

		el conteo de líneas lógicas.	

Tipo de conteo Físico / Lógico	**Tipo**	**Lenguaje**	
	Lógico	Python	
Tipo de sentencia	**Incluir**	**Comentarios**	**Ejemplo (Opcional)**
Ejecutables			
if	Sí		`x = 10` `if x > 5:` `  print("Es mayor que 5")`
elif	No	No se considerarán estas declaraciones porque son como casos. Similar a las sentencias Switch en otros lenguajes.	`x = 10` `if x > 15:` `  print("Mayor que 15")` `elif x > 5:` `  print("Mayor que 5 pero no mayor que 15")`
else	No		`x = 3` `if x > 5:` `  print("Mayor que 5")` `else:` `  print("No es mayor que 5")`
for	Sí		`for i in range(5):` `  print(i)`
while	Sí		`x = 0` `while x < 5:` `  print(x)` `  x += 1`
def	Sí		`def sumar(a, b):` `  return a + b` `print(sumar(3, 4))`
class	Sí		`class Persona:` `  def __init__(self, nombre):` `  self.nombre = nombre` `p = Persona("Juan")` `print(p.nombre)`

try	Sí		
lambda	No	Solo se consideran a las lambdas como líneas físicas	
except	No		
with	Sí		
Listas de comprensión	Sí	Estas declaraciones solo serán consideradas como una única línea lógica	
No ejecutables			
Comentarios			
- En una sola línea	No		
- Multilínea	No		
- Líneas en blanco o vacías	No		
Importaciones	No		
Decoradores	No		
Palabras reservadas	No	A excepción de las mencionadas anteriormente no se toman en consideración otro tipo de palabras reservadas	

Consideraciones para el conteo de clases y métodos

Las clases y métodos solo serán aquellas que sean definidas por la palabra reservada “class” y los métodos por la palabra reservada “def” que el lenguaje python posee. Además, los métodos deberán tener una indentación al estar dentro de su respectiva clase.

Consideraciones para conteo de líneas, específicamente para control de versiones

La siguiente sección habla específicamente sobre cómo se tomará en consideración el conteo de líneas entre versiones de un mismo tipo de archivo Python.

Es decir, se mencionará que se considera como una línea añadida, eliminada, sin cambios, movidas y con pequeños cambios.

1. Añadida: si una línea está en una versión, pero no en su versión previa es una línea.
2. Eliminada: si una línea está en una versión previa, pero no en la siguiente
3. Cambios
    a. Sin cambios: si una línea está contenida en ambas versiones, es una línea original y no ha sufrido cambio.
    b. Pequeños cambios: si una línea tuvo una mínima modificación o si realmente es totalmente nueva.
4. Consideraciones:
    a. Si las líneas se mueven de lugar, éstas se considerarán borradas y añadidas
    b. Si una línea ocupa más de 80 caracteres esta se debe de formatear para que no rebase ese límite. Por otro lado, esta modificación seguirá contando como una única línea de código.

Ejemplos básicos de cómo se aplica el Conteo de Líneas

Los siguientes ejemplos utilizan ambas tablas definidas anteriormente para poder definir qué se considera línea física y línea lógica.

- Archivo con líneas vacías y comentarios:

```
if(a)
    a = a + 1
else
    a = a - 1
```

    - Líneas físicas: 4
    - Líneas lógicas: 1

- Archivo con comentarios:

        - def suma (a, b):
        - # Línea física
        - return a + b
        - # Línea física
        - # Comentario de una línea

- Líneas físicas: 2
- Líneas lógicas: 1

– Conteo de clases y métodos
    - Class clase1:
        - Def metodo1():
            - Return
- Clases 1
- Métodos 1

Estándar de Codificación

Este estándar tiene como objetivo garantizar la calidad, mantenibilidad y uniformidad del código fuente en todos los proyectos realizados por el equipo, promoviendo prácticas de desarrollo consistentes y eficientes.

1. Convenciones de Nombres

A continuación, se especifica las convenciones que se deben seguir para nombrar las clases, métodos y variables:

- **Clases**

o Se utiliza la convención Upper Camel Case para nombrar las clases.

Ejemplo

Buen uso

```
class MiClase:
    #codigo de la clase
    Pass
```

Mal uso

```
class miclase:
    #codigo de la clase
    Pass
```

- **Métodos**

o Se emplea la convención snake_case para nombrar los métodos dentro de las clases. Ejemplo

Buen uso

```
class MiClase:
    def mi_primera_funcion(self):
    #codigo de la clase
    Pass
```

Mal uso

```
class miclase:
    def miPrimeraFuncion(self):
    #codigo de la clase
    Pass
```

- **Variables**

o Para las variables, se aplican la convención snake_case, para toda aquella variable que requiera ser declarada con más de una palabra, solo si esto ayuda al correcto entendimiento de su propósito.

Buen uso
Variable con un nombre claro y descriptivo mi_variable = 1000
Mal uso
Variable que no sigue la convención **MiVariable = 1000**

- **Archivos y Carpetas:**
    - **Archivos:** Los archivos deben de seguir un formato snake_case con mayúscula en cada inicio de palabra.

Buen uso
Mi_Archivo_Principal.py
Mal uso
miArchivo_principal.py

    - **Carpetas:** El nombre de las carpetas que se creen deben de utilizar un formato Upper Camel Case.
    -

Buen uso
MiCarpeta/
Mal uso
miCarpeta/

2. Comentarios en el Código

Las especificaciones que se deben seguir para documentar el código son las siguientes:

- **Comentarios en Línea**

- Los comentarios dentro del código se indicarán utilizando el símbolo # seguido de un texto explicativo, el cual, deberá ser breve y relevante y deberá evitarse poner comentarios en línea con el codigo.
-

Buen uso
def calcular_suma(a, b):   # Esta función calcula el total de sumar a con b   return a + b
Mal uso
def calcular_suma(a, b):   return a + b # comentario que se mezcla con el codigo

- **Docstrings**

  Documentar cada función al inicio con una breve descripción de su propósito, los parámetros que recibe y los valores que retorna.
- Se utilizan docstrings para documentar las clases y los métodos, describiendo su propósito y/o funcionalidad de manera clara y concisa.
- Los docstrings deben usarse al inicio de **clases**, **funciones** y **módulos**. Ejemplo para funciones:
-

Buen uso
def calcular_suma(a, b):   “””   Esta funcion calcula la suma de dos números   Parámetros: a(int): El primer numero   b(int): El segundo numero   Retorna: int: La suma de los 2 numeros   “””

```
    return a + b
```

Mal uso

```
def calcular_suma(a, b):
    “””
    La función realiza la resta de dos números
    Contexto:
    Cuando hice esta función estaba lloviendo
    El cielo estaba …
    “””
    return a + b
```

3. Formato y Estructura

Indica cómo debe organizarse el código.

- **Indentación**
  - Usar 4 espacios para la indentación o 1 tabulación por nivel de indentación.
  - Las estructuras de control (por ejemplo: *if, for, while, etc*) y definiciones (*class, def*) deben estar correctamente alineadas para garantizar la legibilidad.
- **Líneas de código**

  Limitar la longitud de una línea a 80 caracteres, excepto cuando sea completamente inevitable. Por ejemplo, cuando se presenten líneas con cadenas de texto largas o declaraciones complejas.
- **Separación entre secciones**

  Dejar una línea en blanco entre:
  - Métodos dentro de una clase.
  - La definición de clases y otros bloques principales.
  - Comentarios de bloque y el código al que hacen referencia.
- **Bloques Lógicos y de Control**

  En estructuras de control como if y for, emplear una línea por cada declaración, incluso cuando sea breve, para mayor claridad.

  Ejemplo:

Buen uso
if not línea_sin_espacios: continue
Mal uso
if not línea_sin_espacios: continue

- **Importaciones**

  Colocar las importaciones al inicio del archivo y en el siguiente orden:

  - Librerías estándar.
  - Librerías externas.
  - Módulos locales.

- **4. Manejo de errores**

  Utilizar el manejo de errores para ciertas circunstancias que puedan hacer que el programa falle.

  - **FileNotFoundError:** Esta excepción ocurrirá si el usuario ha introducido mal el nombre del archivo que desea analizar o si no existe en la carpeta designada para que el analizador pueda acceder a ella.
  - **IOError:** Esta excepción ocurrirá si ocurre un problema al leer el archivo, es decir, permisos insuficientes para acceder a la ubicación del archivo o si ocurre un error en el disco.
  - **UnicodeDecodeError:** Esta excepción ocurrirá si el archivo a leer no esta en un formado utf-8 es decir si no es un archivo valido para el sistema.

Manual de Usuario

Introducción

El programa **"Comparador_De_Versiones.exe"** está diseñado para analizar y comparar dos archivos de código fuente escritos en Python, proporcionando información detallada sobre los cambios realizados entre ambas versiones. Además de comparar versiones, el programa cuenta con funcionalidades adicionales para

analizar y reportar detalles sobre líneas de código (LOC), clases y métodos presentes en los archivos procesados.

Requisitos Previos

- **Archivos de entrada:** Dos archivos Python (.py) colocados en la carpeta Analizador junto al ejecutable.

Estructura de Archivos y Carpetas

1. **Comparador_De_Versiones.exe:** Archivo ejecutable principal.
2. **Carpeta Analizador:**
    a. Aquí deben colocarse los archivos .py que se desean comparar.
    b. Asegúrese de nombrar los archivos correctamente y utilizar sus nombres exactos al ejecutar el programa.
3. **Archivos de Salida:** Los resultados del análisis se guardan automáticamente en la misma carpeta Analizador.

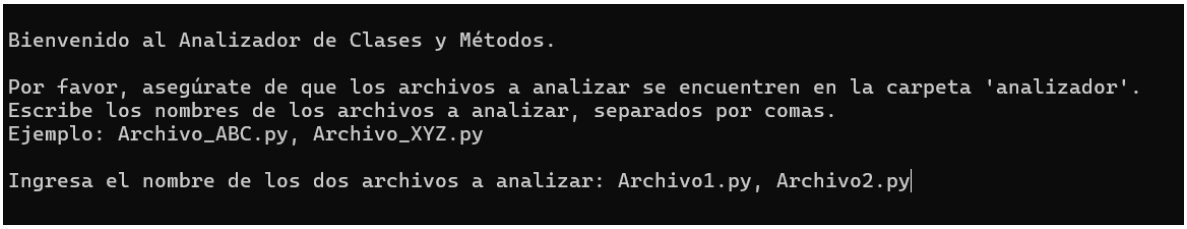
Pasos para Usar el Programa

1. Preparar los Archivos a Comparar

1. Cree una carpeta llamada Analizador en el mismo directorio que el ejecutable si no lo hace el programa la creara.
2. Coloque los dos archivos Python (Archivo1.py y Archivo2.py) dentro de esta carpeta.
3. Asegurate de tener python 3.8 o superior en tu computadora.

2. Ejecutar el Programa

1. Abra una terminal o haga doble clic en el ejecutable.
2. Siga las instrucciones en pantalla. Se le pedirá que ingrese los nombres de los archivos a analizar.
    a. Ejemplo:



3. Funcionalidades del Programa

Al introducir los archivos al programa, este realiza lo siguiente:

A. Comparación de Versiones

- Detecta y clasifica los cambios realizados entre los dos archivos:
    - **Líneas sin cambios:** Líneas idénticas en ambos archivos.
    - **Líneas añadidas:** Líneas presentes únicamente en el archivo modificado.
    - **Líneas eliminadas:** Líneas presentes únicamente en el archivo original.
    - **Líneas modificadas:** Líneas con pequeños cambios detectados a nivel de caracteres.
- El programa genera dos nuevos archivos etiquetados como Version_Original.py y Version_Modificada.py donde:
    - Se etiquetan los cambios realizados con comentarios en linea.
    - Se formatean las líneas que exceden los 80 caracteres.

B. Análisis de Código

1. **Cálculo de Líneas de Código (LOC):**
    - **Líneas físicas: Total de líneas de código sin contar comentarios ni líneas en blanco.**
    - **Líneas lógicas: Bloques de código como clases, métodos, bucles o condicionales.**
2. **Análisis de Clases y Métodos:**
    - **Número total de clases presentes en cada archivo.**
    - **Número total de métodos dentro de cada clase.**
    - **Distribución de líneas dentro de cada clase.**

C.- Formato del Reporte:

El análisis produce una salida en pantalla como la siguiente:

```
Bienvenido al Analizador de Clases y Métodos.

Por favor, asegúrate de que los archivos a analizar se encuentren en la carpeta 'analizador'.
Escribe los nombres de los archivos a analizar, separados por comas.
Ejemplo: Archivo_ABC.py, Archivo_XYZ.py

Ingresa el nombre de los dos archivos a analizar: Prueba1.py, Prueba2.py

Analizando archivo: Prueba1.py

Analizando archivo: Prueba2.py

Comparando archivos...
Líneas añadidas: 1
Líneas eliminadas: 1
Líneas sin cambios: 9
Líneas con cambios pequeños: 0

Informe del archivo :Prueba1.py
------------------------------------------------------------
Clases                         | Métodos     | Líneas
------------------------------------------------------------
MiClaseCompleja:               | 4           | 9
------------------------------------------------------------
Total líneas físicas           | 20
Total líneas lógicas           | 10
Total de clases                | 1

Informe del archivo :Prueba2.py
------------------------------------------------------------
Clases                         | Métodos     | Líneas
------------------------------------------------------------
MiClaseCompleja:               | 4           | 9
------------------------------------------------------------
Total líneas físicas           | 20
Total líneas lógicas           | 10
Total de clases                | 1

¿Deseas analizar más archivos? (si/no):
```

Consideraciones Importantes

- Verifique que los archivos tengan extensión .py
- Si encuentra errores, asegúrese de que los nombres de archivo sean correctos y estén en la carpeta Analizador.
- Las líneas comentadas o vacías no son analizadas ni reportadas en los resultados.

Manejo de excepciones, posibles errores comunes

Al ejecutar el programa pueden darse ciertos escenarios en los que el programa tenga que finalizar debido a un incorrecto uso de este. A continuación, se mencionará cuando el programa se verá forzado a terminar su ejecución, cada tipo de excepción (error/fallo) vendrá acompañado de un mensaje dando la razón de por qué finalizo.

1. **Abrir un archivo que no existe**: por ejemplo, si el programa intenta leer un(os) archivo(s) que no está en el directorio especificado, se generará este error.
2. **Cantidad de archivos:** Esta excepción en particular no detendrá el programa, pero se mostrará en pantalla que se tiene que ingresar exactamente 2 archivos. Así como se especificó anteriormente en el manual de cómo ejecutar el programa.
3. **Error de entrada / salida:** esto puede ocurrir por problemas al intentar leer o escribir un archivo.
4. **Problema al decodificar:** cuando el programa intenta decodificar una cadena de bytes en texto Unicode. Esto puede suceder normalmente cuando contiene caracteres que no son compatibles con la codificación especificada.
5. **Archivo mal estructurado (con código fuera de clases): e**n caso de introducir un archivo fuera del paradigma POO el analizador invocara una excepción y le mandara un aviso.

Unidad 2

Casos de Prueba

Los casos de prueba se diseñan para verificar que el código funcione como se espera en diferentes situaciones. Así como al usuario para la verificación del funcionamiento del programa, así como la garantía de revisar cada procedimiento que se esté realizando por el equipo:

Para esta versión de pruebas se han implementado pruebas para el módulo designado como Comparador_De_Versiones.py que será el responsable de resolver la sección del problema de comparar las líneas añadidas, eliminadas, movidas y con cambios de los archivos.

Instrucciones para el llenado del formato de Registro de Pruebas

Registro de Pruebas **(Tipo de prueba)**	
Id/Nombre	Identificador único de las pruebas
Objetivo	Mencionar brevemente la razón por la cual se está realizando la prueba
Descripción	Describir los datos (las entradas) y el procesamiento que va a tener el programa sobre esas entradas.

Condiciones (ambiente de la prueba)	Mencionar el ambiente el que se está ejecutando la prueba. Es decir, las herramientas utilizadas con las que se llevó a cabo la prueba.
Resultados esperados	Listar los resultados que la prueba debe producir si corre apropiadamente
Resultados Obtenidos	Listar los resultados que produjo la prueba. Las salidas obtenidas, así como los posibles defectos encontrados.
Comentarios	Escribir posibles comentarios que puedan ayudar al mejor entendimiento de la prueba.
Estado	Señalar el cómo se considera la prueba: • Exitosa • Fallida

Pruebas Unitarias

Las pruebas unitarias se diseñaron para cubrir las funcionalidades del nuevo módulo antes de ser integrado al sistema. Con la finalidad de validar el correcto funcionamiento del mismo en diferentes situaciones.

Objetivo: Validar la correcta identificación y etiquetado de las líneas en distintos escenarios de código.

Funcionalidad: Comparar líneas

Registro de Pruebas **(Unitaria)**	
Id/Nombre	
Objetivo	Validar la capacidad del Comparador de identificar líneas añadidas, eliminadas, modificadas, con cambios de 2 archivos python.
Descripción	Esta prueba implementa distintos escenarios en los que el comparador deberá identificar correctamente y distinguir las líneas añadidas, eliminadas, movidas y con cambios. Esta prueba presenta escenarios donde se tienen distintos casos de códigos que respetan nuestro estándar de codificación y están en distintos contextos, es decir ningún código de prueba es igual.

	• **test_caso1_identico**s: Prueba para archivos idénticos. • **test_caso2_linea_añadida:** Prueba para detectar líneas añadidas. • **test_caso3_linea_borrada**: Prueba para detectar líneas eliminadas. • **test_caso4_linea_movida:** Prueba para detectar líneas movidas. • **test_caso5_cambios_pequeños:** Prueba para detectar pequeños cambios en las líneas. • **test_formato_80_caracteres:** Prueba para validar el formato de 80 caracteres por línea.
Condiciones (ambiente de la prueba)	Configuración del ambiente: • Se utiliza la clase TestComparadorArchivos que ejecuta pruebas sobre los archivos en un directorio denominado Test y dentro de este se encuentran sub carpetas de nombre "Caso" y un identificador numérico para así tener una mejor organización. Esta clase automatiza las pruebas sin utilizar librerías externas, asegurando que el comparador tenga acceso a ellas y las realice. Los archivos fuente deben cumplir con las siguientes condiciones para que la prueba sea efectiva: • Indentación valida: Uso consistente de espacios y tabulaciones (ver estándar de codificación) • Comentarios: comentarios en línea (#) y docstrings("""""). Pueden o no incluirlos • Nombres de clases y métodos: Estructurados de manera correcta para ser identificados por la herramienta. • Uso correcto de los bloques de control if,for,while,class. Como se establece en el estándar de codificación. • Los archivos no deben tener errores de sintaxis. • No deben tener código comentado

Resultados esperados	Se espera recibir por resultados los números de líneas previamente contados y establecidos como valores esperados en la función que automatiza las pruebas por cada caso (5 en total). Siendo los resultados esperados: • **test_caso1_identicos** ○ Líneas sin cambios: **5** • **test_caso2_linea_añadida** ○ Líneas añadidas: **2** ○ Líneas sin cambios: **5** • **test_caso3_linea_borrada** ○ Líneas eliminadas: **9** ○ Líneas sin cambios: **11** • **test_caso4_linea_movida** ○ Líneas añadidas: **1** ○ Líneas eliminadas: **1** ○ Líneas sin cambios: **9** • **test_caso5_cambios_pequeños** ○ Líneas añadidas: **1** ○ Líneas eliminadas: **1** ○ Líneas sin cambios: **9**
Resultados Obtenidos	Los resultados obtenidos fueron aquellos que se calcularon utilizando el módulo con los distintos casos. En los que se obtuvieron los resultados : • test_caso1_identicos ○ Líneas sin cambios: 5 • test_caso2_linea_añadida ○ Líneas añadidas: 2 ○ Líneas sin cambios: 5 • test_caso3_linea_borrada ○ Líneas eliminadas: 9 ○ Líneas sin cambios: 11

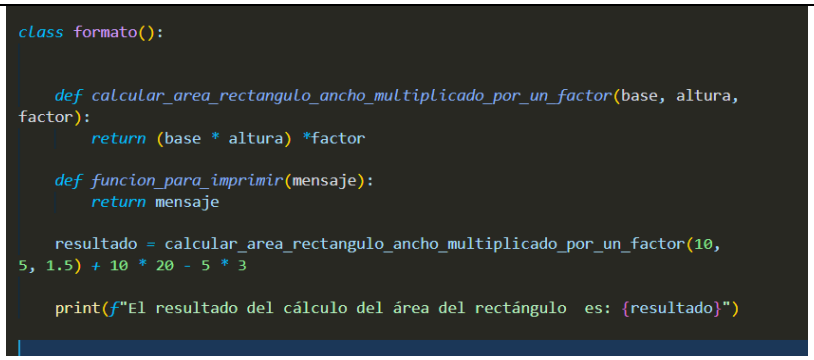
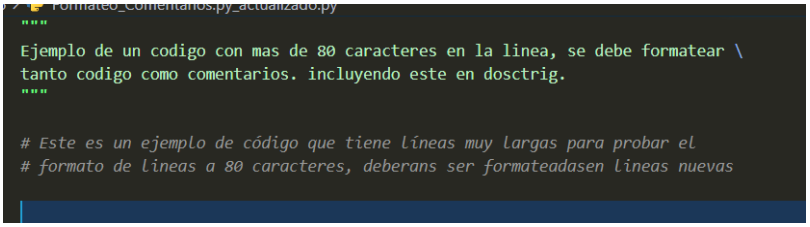
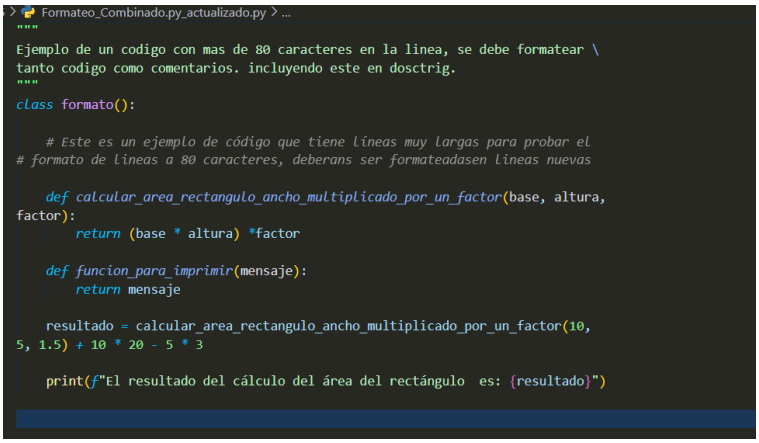
	• test_caso4_linea_movida ○ Líneas añadidas: 1 ○ Líneas eliminadas: 1 ○ Líneas sin cambios: 9 • test_caso5_cambios_pequeños ○ Líneas añadidas: 1 ○ Líneas eliminadas: 1 ○ Líneas sin cambios: 9
Comentarios	Teniendo la comparación, se obtuvo que la prueba fue exitosa, demostrando que el Comparador es capaz de identificar líneas añadidas, eliminadas, movidas y con cambios.
Estado	Exitosa.

Objetivo: Validar que el módulo formatee correctamente las líneas del archivo, siendo comentarios y código a 80 caracteres

Funcionalidad: Formateo de Líneas y comentarios

Registro de Pruebas **(Unitaria)**	
Id/Nombre	
Objetivo	Validar que el módulo formatea correctamente comentarios y código añadiendo nuevas líneas donde se continúen aquellos que sobrepasen el límite.
Descripción	Se ejecutará una prueba que automatizará la la ejecución del módulo para que este compare archivos con líneas de más de 80 caracteres y las formatee añadiendo un # si es un comentario o un \ si es código para las nuevas líneas creadas. Los casos a cubrir son: • test para formatear comentarios • test para formatear líneas de código

	• test para formatear ambos en un solo archivo
Condiciones (ambiente de la prueba)	Configuración del ambiente: • Se utiliza la clase TestComparadorArchivos que ejecuta pruebas sobre los archivos en un directorio denominado Test y dentro de este se encuentran sub carpetas de nombre “Caso” y un identificador numérico para así tener una mejor organización. Esta clase automatiza las pruebas sin utilizar librerías externas, asegurando que el comparador tenga acceso a ellas y las realice. Los archivos fuente deben cumplir con las siguientes condiciones para que la prueba sea efectiva: • Indentación valida: Uso consistente de espacios y tabulaciones (ver estándar de codificación) • Comentarios: comentarios en línea (#) y docstrings(“””””). Pueden o no incluirlos • Nombres de clases y métodos: Estructurados de manera correcta para ser identificados por la herramienta. • Uso correcto de los bloques de control if,for,while,class. Como se establece en el estándar de codificación. • Los archivos no deben tener errores de sintaxis. • No deben tener código comentado
Resultados esperados	Se espera que en los 3 casos el comparador formatee las líneas en 80n caracteres, añadiendo # si son comentarios o \ si son codigos para continuar en una nueva línea.
Resultados Obtenidos	Resultado de la prueba para formatear código.

	 Resultado de la prueba de formateo de comentarios  Resultado de la prueba de formateo de código con comentarios 
Comentarios	Las pruebas demuestran que el comparador es capaz de formatear las líneas con código y comentarios de manera efectiva.
Estado	Exitosa

Pruebas de Integración

Propósito: Verificar que el nuevo módulo interactúe correctamente con el resto del sistema para asegurar que el flujo completo de datos, entradas y salidas funcionen bien.

Registro de Pruebas

(Integración)	
Id/Nombre	Prueba Integración
Objetivo	Probar que el sistema que utiliza el nuevo módulo, formatee correctamente los archivos y señale las líneas modificadas, y que el resto de funcionalidades de `app.py` sigan funcionando correctamente.
Descripción	Se diseño una prueba que imita el uso completo del flujo del sistema, introduciendo archivos al sistema, enviándoselos a los módulos para que se procesen los mismos y devuelvan cada uno su respectivo informe.
Condiciones (ambiente de la prueba)	Configuración del ambiente: • Se utiliza una clase llamada TestIntegracion que ejecuta pruebas sobre el archivo app.py que es el archivo principal que invoca al resto del código. Los archivos fuente deben cumplir con las siguientes condiciones para que la prueba sea efectiva: • Indentación valida: Uso consistente de espacios y tabulaciones (ver estándar de codificación) • Comentarios: comentarios en línea (#) y docstrings(“””””). Pueden o no incluirlos • Nombres de clases y métodos: Estructurados de manera correcta para ser identificados por la herramienta. • Uso correcto de los bloques de control if,for,while,class. Como se establece en el estándar de codificación. • Los archivos no deben tener errores de sintaxis. • No deben tener código comentado
Resultados esperados	Se espera la correcta ejecucion de todos los procesos que involucra comparar un archivo, analizando comparando, formateando los archivos y desplegando el informe.

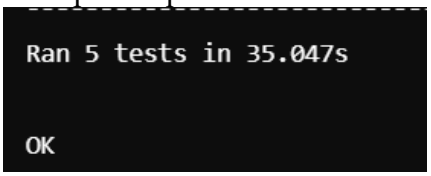
Resultados Obtenidos	Se simuló una entrada por parte de la prueba para que se invoquen los métodos y validar si no se crean conflictos o errores entre módulos. La prueba devuelve un ok si todo salió como se espera. 
Comentarios	Por lo que se tiene que el módulo Comparador_De_versiones ha sido integrado satisfactoriamente al sistema.
Estado	Exitosa

Pruebas de regresión

Propósito: Asegurarse de que los cambios realizados no han afectado negativamente el comportamiento de las funcionalidades previas del sistema.

Registro de Pruebas (Regresión)	
Id/Nombre	Prueba de regresión
Objetivo	Confirmar que las partes del sistema que no han cambiado sigan funcionando como antes. Verificar que el nuevo módulo no haya introducido errores o comportamientos inesperados en otras áreas del sistema que no se ven afectadas directamente por el nuevo código.
Descripción	Se diseño una prueba para validar que el flujo principal de la aplicación (la interacción con el usuario, análisis de archivos) funcione como se espere en los distintos casos de prueba a los que se someterá el programa. Los casos son : - **Verificación de inicialización y bienvenida:** o Asegurarse de que la carpeta `./analizador` se crea automáticamente si no existe.

	o Validar que el mensaje de bienvenida se despliega correctamente al inicio del programa. - **Validación de entrada del usuario:** o Probar con entradas válidas de dos archivos existentes (`archivo1.py, archivo2.py`). o Probar con entradas inválidas, como: ▪ Nombres no separados por comas. ▪ Archivos inexistentes en la carpeta. ▪ Archivos que no tienen extensión `.py`. - Verificar los mensajes de error para cada caso. - **Análisis de archivos (módulo `AnalizadorEstructural`):** o Validar que el análisis estructural de cada archivo produce los resultados esperados. o Confirmar que no hay errores si el análisis falla en un archivo mientras otro es válido. - **Comparación de archivos (módulo `ComparadorArchivos`):** o Confirmar que el módulo compara correctamente dos archivos válidos y despliega el informe esperado. o Probar con archivos idénticos y archivos diferentes, verificando los resultados. - **Flujo de interacción completo:** o Simular el flujo completo con diferentes combinaciones de entradas válidas e inválidas, asegurándose de que el programa maneja cada situación correctamente: ▪ Análisis de múltiples pares de archivos en una sola sesión. ▪ Terminación correcta del programa cuando el usuario elige no continuar. - **Manejo de excepciones:** o Probar la aplicación simulando errores como permisos insuficientes para leer un archivo o el uso de un archivo corrupto, verificando que el programa maneja estos errores adecuadamente sin detenerse inesperadamente.
Condiciones (ambiente de la prueba)	Configuración del ambiente: • Se utiliza una clase llamada TestRegresion que ejecuta pruebas sobre el archivo app.py que es el archivo principal que invoca al resto del código.

	• Los archivos fuente deben cumplir con las siguientes condiciones para que la prueba sea efectiva: • Indentación valida: Uso consistente de espacios y tabulaciones (ver estándar de codificación) • Comentarios: comentarios en línea (#) y docstrings(“””””). Pueden o no incluirlos • Nombres de clases y métodos: Estructurados de manera correcta para ser identificados por la herramienta. • Uso correcto de los bloques de control if,for,while,class. Como se establece en el estándar de codificación. • Los archivos no deben tener errores de sintaxis. • No deben tener código comentado
Resultados esperados	Se espera recibir por resultados la una confirmación por parte del código de pruebas para saber que las pruebas se ejecutaron correctamente en todos los casos mencionados.
Resultados Obtenidos	Los resultados obtenidos muestran que las pruebas no tuvieron ningun error por lo que todas fueron exitosas. 
Comentarios	De este modo se tiene que el sistema cumple con las funcionalidades aun después de integrar el nuevo módulo y es capaza de realizar las funcionalidades que se tenían antes de implementar el mismo.
Estado	Exitosa.

Unidad 3

Estimación de Tamaño del Software

Se utilizará como métrica Puntos Funcionales para obtener el tamaño y complejidad del sistema descrito.

Se tomará en consideración el conteo de los siguientes elementos funcionales:

- Entradas lógicas
- Salidas
- Consulta (Query)
- Archivos Lógicos Internos
- Archivos Lógicos Externos

Para la asignación del grado de complejidad se usará como base la siguiente tabla:

Elemento Funcional	**Factor de Ponderación**		
	Simple	Promedio	Complejo
Entradas Externas	3	4	6
Salidas Externas	4	5	7
Consultas Externas	3	4	6
Archivos Lógicos Externos	7	10	15
Archivos Lógicos Internos	5	7	10

Ecuación para el conteo de Puntos Funcionales sin Ajuste

$$UFC = \sum Cantidad_{elemento} \cdot Peso_{elemento}$$

Se utilizará la siguiente plantilla para calcular el factor de complejidad técnica para los puntos funcionales sin ajustar

Componentes del factor de complejidad técnica

F_1	Fiabilidad de la copia de seguridad y recuperación	
F_2	Funciones distribuidas	
F_3	Configuración utilizada	
F_4	Facilidad operativa	
F_5	Complejidad de interfaz	
F_6	Reutilización	
F_7	Instalaciones múltiples	
F_8	Comunicaciones de datos	
F_9	Desempeño	
F_{10}	Entrada de datos en línea	
F_{11}	Actualización en línea	
F_{12}	Procesamiento complejo	
F_{13}	Facilidad de instalación	
F_{14}	Facilidad de cambio	
	Total	

0-Irrelevante o sin influencia

1-Incidental

2-Moderado

3-Medio

4-Significativo

5-Esencial

Ecuación para el factor de complejidad técnica

$$TFC = 0.65 + 0.01 \sum_{i=1}^{14} F_i$$

Ecuación para el conteo de Puntos Funcionales Ajustado

$$FP = UFC \cdot TFC$$

Por cada punto funcional encontrado se considerará:

$$1pf\ tarda\ aprox\ 11.35\ hrs$$

Descripción del sistema

Escribir un programa para contar el numero de cambios que sufre entre versiones, al compararlo con su versión previa.

1. Comparar
    a. Si una línea está contenida en ambas versiones, es una línea original y no ha sufrido cambio.
    b. Si una línea está en una versión, pero no en su versión previa es una línea añadida.
    c. Si una línea está en una versión previa, pero no en la siguiente es una línea borrada.
2. Contar los cambios de líneas añadidas y borradas.
3. Por cada línea añadida
    a. Comparar si es una pequeña modificación o si realmente es totalmente nueva.
    b. Si las líneas se mueven de lugar, éstas se considerarán borradas y añadidas.
4. Cada línea no debe ocupar más allá de 80 caracteres
    a. Por lo que se tiene que formatear aquellas líneas que rebasen los 80 caracteres en dos o más líneas.
    b. Esta línea seguirá contando como una única línea de código
    c. Etiquetar cada línea añadida en la nueva versión al final de la misma con un comentario
    d. Etiquetar cada línea borrada en la versión anterior al final de la misma con un comentario
5. Realizar el conteo como lo indica el ejercicio 2

Requisitos identificados:

1. Entrada del archivo Python original
2. Entrada del mismo archivo Python modificado

3. Entrada de querer continuar el programa o terminarlo.
4. Salida de recuento de los tipos de cambios que se realizaron entre versiones
5. Salida de conteo de líneas físicas y lógicas utilizadas del proyecto 2
6. Salida de los archivos etiquetados

Conteo de los puntos de Función sin ajuste

Elemento	Cantidad	Peso	Total
Entradas Externas	1	3	3
	2	6	12
Salidas Externas	1	5	5
	2	7	14
Consultas Externas	0	0	0
Archivos Lógicos Externos	0	0	0
Archivos Lógicos Internos	0	0	0
Total			34

$$UFC = (1 * 3) + (2 * 6) + (1 * 5) + (2 * 7) = 34$$

$$UFC = 16$$

Componentes del factor de complejidad técnica		
F_1	Fiabilidad de la copia de seguridad y recuperación	0
F_2	Funciones distribuidas	4
F_3	Configuración utilizada	0
F_4	Facilidad operativa	1
F_5	Complejidad de interfaz	0
F_6	Reutilización	4
F_7	Instalaciones múltiples	0

F_8	Comunicaciones de datos	0
F_9	Desempeño	3
F_{10}	Entrada de datos en línea	0
F_{11}	Actualización en línea	0
F_{12}	Procesamiento complejo	3
F_{13}	Facilidad de instalación	1
F_{14}	Facilidad de cambio	0
	Total	16

$$TFC = 0.65 + 0.01(16) = 0.81$$

$$FP = 34 \cdot\ 0.81 = 27.54\ \approx 28$$

Tiempo esperado que puede tomar realizar el sistema:

$$Tiempo_{hrs} = 28 \cdot 11.35\ hrs = 317.8\ hrs\ \approx 318\ hrs$$

$$Tiempo_{dias} = \frac{318}{24} = 13.25\ días\ \approx 14\ días$$

Considerando que se tendrán al menos 2 personas encargadas del desarrollo del código este valor puede reducirse al siguiente dato

$$Tiempo_{dias} = \frac{14}{2}\ días/desarrollador =$$

$$7\ \ días/desarrollador$$

Unidad 4. Mantenimiento

La transición del Proyecto 2 al Proyecto 3 ha representado un avance significativo en el sistema, con un enfoque en la integración de nuevas funcionalidades y la optimización de herramientas existentes. Este proceso incluyó decisiones técnicas clave para abordar nuevas necesidades de mantenimiento, mejorar la experiencia del usuario y garantizar la precisión del análisis.

Decisiones y Modificaciones

El Proyecto 3 reutilizó y adaptó el código del Proyecto 2, ajustándolo a nuevos requerimientos relacionados con el mantenimiento de software y la comparación entre versiones. Además de conservar las funcionalidades de análisis de POO, se implementaron mejoras clave en la detección de cambios, manejo de errores, y en la presentación de informes detallados.

El sistema ahora incluye una funcionalidad que permite comparar dos versiones consecutivas del programa para identificar los cambios realizados, tanto en el código como en la documentación.

Tipos de Mantenimiento Realizados

Mantenimiento Perfectivo:

- Optimización del análisis de POO y enriquecimiento de resultados mediante una presentación tabular más detallada.
- Inclusión de nuevas características, como el conteo y clasificación de líneas movidas.
- Mejora del flujo interactivo y del menú de usuario para cubrir escenarios más complejos.

Mantenimiento Correctivo:

- Corrección en la lógica para identificar métodos externos y manejar casos de anidación compleja.
- Ajuste en el manejo de errores que ahora permite al usuario leer los mensajes cuando una entrada no cumple con los principios de POO.
- Resolución de inconsistencias en la validación de archivos para evitar interrupciones por datos corruptos.

Principales Decisiones y Modificaciones

Procesos Realizados

1. **Comparación de Versiones**

- Decisión: Introducir una herramienta dedicada para analizar diferencias entre versiones consecutivas.

- Modificaciones
    - Implementación de un contador para clasificar líneas cambiadas, añadidas, eliminadas y movidas.
    - Ajuste en las etiquetas para diferenciar líneas movidas de las borradas y añadidas.
- Justificación: Facilita el seguimiento de cambios, resalta las modificaciones realizadas, y evita confusiones en los reportes del usuario.

2. Optimización del Análisis de POO

- Decisión: Refinar las validaciones de clases y métodos POO para mayor precisión.
- Modificaciones
    - Mejora en la exclusión de métodos externos en el análisis de POO.
    - Ajuste en la lógica para manejar estructuras de anidación compleja.
- Justificación: Garantizar que el análisis de POO sea preciso y cumpla con los estándares establecidos

3. Presentación Enriquecida de Resultados

- Decisión: Mejorar el formato tabular existente para reportar métricas y cambios.
- Modificaciones
    - Inclusión de columnas adicionales para líneas añadidas, borradas y movidas.
    - Creación de un resumen separado que facilite el análisis posterior.
- Justificación: Ofrecer datos claros y detallados, adaptados a las necesidades de los desarrolladores.

4. Flujo Continuo para el Usuario

- Decisión: Extender el ciclo interactivo para incluir análisis y comparaciones entre versiones.
- Modificaciones
    - Expansión del menú para manejar análisis individuales o comparaciones de versiones consecutivas.
    - Implementación de mecanismos que visualicen diferencias específicas entre archivos.
    - Aumentar el tiempo de espera antes de que el programa finalice.
- Justificación: Incrementar la flexibilidad del sistema y mejorar la experiencia del usuario.

5. **Mejora del Manejo de Errores**

- Decisión: Extender el manejo de errores para incluir validaciones en comparaciones.
- Modificaciones
  - Validación de entradas para garantizar que ambas versiones existan antes de compararlas.
  - Captura de excepciones para manejar archivos corruptos o inconsistencias detectadas.
- Justificación: Incrementar la robustez del sistema y prevenir interrupciones inesperadas.

Procesos Realizados

- Actualización de los comentarios en el código para reflejar las nuevas funcionalidades.
- Inclusión de explicaciones claras sobre la comparación de versiones, validaciones adicionales y manejo de errores.
- Expansión del manual de usuario con instrucciones para usar las nuevas herramientas.

Modularización

- Introducción de módulos independientes para análisis de POO, diferencias entre versiones, y generación de reportes.

Documentación

- Mantenimiento: Desglose detallado de las mejoras realizadas entre el Proyecto 2 y el Proyecto 3.
- Documentación del Código: Inclusión de comentarios descriptivos en las nuevas funciones relacionadas con la comparación de versiones y la presentación de resultados.

Interacción del Programa con el Usuario

El sistema ahora ofrece las siguientes mejoras para facilitar su uso:

1. **Ciclo Interactivo Mejorado:**

Opciones para analizar archivos individuales, comparar versiones y visualizar cambios detectados en una sola sesión.

Informes Enriquecidos

Formato tabular con métricas detalladas y resultados de comparación de versiones.

Gestión de Errores Avanzada

- Mensajes claros para errores relacionados con archivos inexistentes, datos corruptos o inconsistencias en la comparación.
- Pausas para garantizar que los usuarios puedan leer los mensajes antes de terminar la ejecución.

Unidad 5: Aseguramiento de la calidad

Revisión de calidad

El objetivo principal del aseguramiento de la calidad es garantizar que el sistema:

- Cumpla con los requisitos funcionales y no funcionales definidos.
- Entregue resultados precisos y consistentes en diferentes entornos de uso.
- Sea confiable, mantenible y eficiente.

Además, este apartado tiene como propósito detallar las actividades, roles, procesos y herramientas que se utilizaron para asegurar tanto la calidad técnica del producto como la calidad administrativa del proyecto.

Metodología del aseguramiento de la calidad.

Esta sección describe cómo se implementará el aseguramiento de la calidad en el proyecto.

Enfoque

El enfoque de calidad en este proyecto se abordará en dos niveles: administrativo y técnico, con el objetivo de asegurar que tanto el proceso de desarrollo como el producto final cumplan con los estándares establecidos.

Calidad administrativa

En el nivel administrativo, se priorizará la planificación adecuada, la gestión eficiente de recursos y el seguimiento constante del progreso del proyecto. Esto incluirá la definición

clara de roles y responsabilidades, la organización de inspecciones periódicas del proyecto, y la supervisión de los plazos y recursos. Las revisiones formales se llevarán a cabo para evaluar el cumplimiento de los estándares, asegurando que el proyecto avance de acuerdo con lo planificado.

Calidad técnica

A nivel técnico, se implementarán dos principales estrategias para garantizar la calidad del software:

- **Inspecciones**: Las inspecciones formales del código serán realizadas de manera regular para identificar defectos en el diseño, la lógica del software y la calidad del código fuente. Estas inspecciones ayudarán a detectar problemas tempranos en el ciclo de desarrollo, permitiendo que se corrijan.
- **Pruebas**: El sistema será sometido a una serie de pruebas técnicas, que incluyen:
    - **Pruebas unitarias** para validar el correcto funcionamiento de cada componente individual del sistema.
    - **Pruebas de integración** para asegurar que las distintas partes del sistema trabajen juntas de manera coherente.
    - **Pruebas de regresión:** las cuales nos aseguran que modificaciones realizadas a métodos añadidos (y modificados) no vayan a afectar al funcionamiento general del sistema.
    - **Pruebas de aceptación** para verificar que el sistema cumpla con los requisitos establecidos y entregue los resultados esperados.

Ambas estrategias, inspecciones y pruebas, estarán interrelacionadas y se realizarán a lo largo de todo el ciclo de vida del desarrollo, garantizando la detección y corrección temprana de defectos, así como la validación continua de que el producto cumple con los estándares de calidad.

Roles y responsabilidades

A continuación, se muestra los roles que tendrá cada persona del equipo y sus responsabilidades en el proceso del aseguramiento de la calidad.

- **Líder de calidad**: Jimena Nohemí Cruz Arreola

Sera el responsable de planificar y supervisar las actividades de QA.

- **Moderador**: Se encargará de coordinar las inspecciones de calidad.
- **Inspectores**: Serán los que revisaran el código fuente y reportaran defectos.
- **Autor**: Persona que desarrolló el módulo que se está inspeccionando.
- **Escriba**: Documenta los hallazgos durante las inspecciones.

Actividades del aseguramiento de la calidad

En este apartado se mostrará las actividades/procesos que se llevaron a cabo durante todo el proceso de desarrollo del proyecto.

Num. de revisión	Fecha de la revisión	Participantes de la revisión	Tema revisado	Problema detectado	Impacto potencial	Acción propuesta
1	2/12/2024	Líder de Calidad: Jimena Cruz Arreola Revisor: Luis Palma Pinto Autor: Angel Osalde	Manual de usuario	Falta especificar más la funcionalidad del código, como llama la información, como la procesa, como arroja resultados, que puede pasar cuando se	El usuario puede no entender en su totalidad como hace cada cosa el sistema.	Actualizar el manual de usuario especificando de manera clara y concisa cada uno de los puntos señalados.

				presenta un error, etc.		
2	3/12/2024	Líder de Calidad: Jimena Cruz Arreola Moderador: Gabriel Precenda Revisor: Luis Palma Pinto Autor: Angel Osalde	Manual de usuario.	Falta agregar que si el usuario ingresa un archivo que contenga código comentado, esto sera considerado como mala práctica.	El usuario debe tener pleno conocimiento de lo que el sistema considera correcto y lo que no, porque si no podría haber confusiones.	Agrega este punto de manera detallada.
3	4/12/2024	Líder de Calidad: Jimena Cruz Arreola Moderador: Gabriel Precenda	Documentación de Mantenimiento.	La redacción es algo confuso y no es completamente clara y hace falta mencionar ciertos	No ser claros con lo que se quiere expresar y/o dar a entender al lector puede generar	Actualizar todo aquel punto de la documentación de mantenimiento que no sea totalmente claro y

		Autor: Juan Rodriguez		detalles importantes.	mucha confusion y poco entendimiento de lo importante.	comprensible.

Calidad técnica

Inspección 1

- Artefacto inspeccionado: Código fuente.
- Tipo de inspección: Revisión del codigo fuente.
- Objetivo de la inspección: Identificar defectos en el codigo fuente que pueda afectar su funcionalidad o resultados.

Participantes y roles:

- Moderador: Jimena Cruz Arreola Gabriel Precenda Valle.
- Autor: Ángel Osalde Salazar.
- Inspectores: Luis Palma Pinto y Jimena Cruz Arreola.
- Escriba: Juan Rodríguez Falcon.

Resumen de la Inspección

- Número total de defectos encontrados: 2
- Detalles de los Defectos
    1. Defecto #1:
        - Descripción: El código etiqueta a las líneas movidas como líneas eliminadas y añadidas.
        - Impacto: Puede generar confusiones.
        - Acción propuesta: Actualizar la parte donde nombra a estas líneas, para que ahora sean etiquetadas como líneas movidas.
        - Responsable: Ángel Osalde Salazar.
- Plazo estimado para las correcciones: (1 día).

Resultados de Seguimiento

- Estado del defecto: Resuelto.
- Notas adicionales: Se realizó una segunda revisión y el problema fue corregido con éxito.

Inspección 2

- Artefacto inspeccionado: Pruebas
- Tipo de inspección: Revisión de las pruebas realizadas.
- Objetivo de la inspección: Identificar defectos en la ejecución de las pruebas correspondientes del sistema.

Participantes y roles:

- Moderador: Jimena Cruz Arreola.
- Autor: Gabriel Precenda Valle.
- Inspectores: Ángel Osalde Salazar y Juan Rodríguez Falcon.
- Escriba: Luis Palma Pinto.

Resumen de la Inspección

- Número total de defectos encontrados: 2
- Detalles de los Defectos
    1. Defecto #1:
        - Descripción: Hace falta hacer algunas nuevas pruebas debido a la última actualización del código fuente.
        - Impacto: Puede creerse que el sistema y cada una de sus partes funcionan bien con la ultima actualización cuando realmente no es así.
        - Acción propuesta: Rehacer las pruebas faltantes.
        - Responsable: Gabriel Precenda Valle
- Plazo estimado para las correcciones: (1 día).

Resultados de Seguimiento

- Estado de los defectos: Resuelto.

- Notas adicionales: Se realizó una segunda revisión y el problema fue corregido con éxito.

Inspección 3

- Artefacto inspeccionado: Código fuente.
- Tipo de inspección: Revisión de código
- Objetivo de la inspección: Identificar defectos en el codigo fuente que puuedan afectar sus resultados y funcionalidad.

Participantes y roles:

- Moderador: Jimena Cruz Arreola.
- Autor: Ángel Osalde Salazar.
- Inspectores: Luis Palma Pinto y Gabriel Precenda Valle.
- Escriba: Juan Rodríguez Falcon.

Resumen de la Inspección

- Número total de defectos encontrados: 2
- Detalles de los Defectos
    1. Defecto #1:
        - Descripción: La forma en que el sistema muestra los resultados no es la mejor ni las más visiblemente entendible.
        - Impacto: Puede hacer que los resultados no sean comprensibles debido a que no son visiblemente claros.
        - Acción propuesta: Hacer que el sistema genere una versión analizada de cada archivo analizado, donde se muestre cuáles son las líneas añadidas, movidas o eliminadas.
        - Responsable: Ángel Osalde Salazar.

    2. Defecto #2:
        - Descripción: El sistema no da la opotunidad de ver el mensaje de error generado porque el archivo introducido no esta codificado usando POO, cierra la ventana y no se alcanza a leer.

        - Impacto: El usuario no entendera el porque no le permite usar dicho archivo.
        - Acción propuesta: Escoger el mejor metodo para hacer que la ventana no se cierre de inmediato y permita leer el error..
        - Responsable: Ángel Osalde Salazar.

- Plazo estimado para las correcciones: (1 día).
- Revisión de seguimiento:

Resultados de Seguimiento

- Estado de los defectos: Resuelto.
- Notas adicionales: Se realizó una segunda revisión y los problemas fueron corregidos con éxito.