UNIVERSIDAD AUTÓNOMA DE YUCATÁN

FACULTAD DE MATEMATICAS

Semestre:

Agosto-diciembre

Materia:

Desarrollo Y Mantenimiento de Software

Proyecto Final:

Parte 3

Docente:

M. en C. Carlos Benito Mojica Ruiz.

Alumnos:

Santiago Efrain Itzincab Poot

Rogerio Emmanuel Canto Romero

Eduardo Alberto Gonzalez Ortega

Jafet Andree Mena Solis

Contenido

Unidad #1 3
Manual de usuario 3
Manual Técnico 9
Unidad #2 14
Casos de prueba 14
Pruebas Unitarias: 15
Pruebas de integración: 16
Unidad #3 18
Estimación de tamaño 18
Unidad #4 23
Mantenimiento, documentación de las decisiones y modificaciones 23
Unidad #5 25
Aseguramiento de la calidad 25
Inspecciones 25

Unidad #1

Manual de usuario

Este programa está diseñado como una herramienta para el análisis y comparación de código fuente. Su objetivo principal es ayudar a los desarrolladores a rastrear cambios entre dos versiones de un archivo y generar un informe detallado de las diferencias, mientras analiza las líneas físicas y lógicas del archivo resultante. Además, verifica que el código cumpla con estándares de codificación establecidos para mantener la calidad del software.

Requisitos Previos

1. Sistema y Herramientas:

- Sistema operativo con Python 3.6 o superior instalado.
- Biblioteca tabulate instalada (puede instalarla usando: pip install tabulate).
- tkinter (preinstalada en distribuciones de Python).

2. Archivos de Entrada:

- Dos archivos Python (.py) que serán analizados.

3. Funcionalidades del Script:

- Verifica estándares de codificación comunes (longitud de líneas, indentación, comentarios, etc.).
- Identifica funciones y bloques principales en el código.
- Cuenta las líneas físicas y lógicas del archivo.
- Presenta un resumen detallado en formato tabular.

Cómo Usar el Script

1. Ejecución Inicial

- Guarde el script en un archivo .py (por ejemplo, analizador_codigo.py).

- Ejecute el script desde la terminal o un IDE:

python analizador_codigo.py

2. Selección de Archivo

- Aparecerá una ventana de selección de archivos.
- Seleccione los archivos Python que desea analizar.
- Si no selecciona un archivo, el programa mostrará un mensaje y se cerrará.

3. Resultados

El script genera los siguientes resultados en la consola:

1. Verificación de Estándares:

- Identifica errores relacionados con:
- Longitud de líneas (>79 caracteres).
- Formato de comentarios.
- Indentación (múltiplos de 4 espacios).
- Identificadores (nombres en minúscula).
- Estructuras de control y definiciones de funciones.

2. Resumen de Funciones y Bloque Principal:

- Muestra las siguientes columnas en formato tabular:
- Programa: Identifica si es una función o el bloque principal.
- Función / Procedimiento: Nombre de la función (si aplica).
- Total de métodos: Número de funciones en el programa.
- LOC físicas: Cantidad de líneas físicas no vacías.
- LOC lógicas: Cantidad de instrucciones clave.

3. Totales del Código Completo:

- Total de líneas físicas y lógicas en el programa.

Flujo de Trabajo del Usuario

1. **Inicio:**

- Ejecute el script desde un terminal o un entorno de desarrollo (IDE).
- Seleccione las dos versiones del archivo que desea comparar utilizando los cuadros de diálogo emergentes.

2. **Análisis:**

- El script analiza los dos archivos seleccionados línea por línea para:
- Identificar líneas añadidas y líneas eliminadas.
- Reorganizar líneas largas que exceden 80 caracteres, dividiéndolas en partes más pequeñas.

El script también compara las versiones y genera un archivo de salida (output.py), que incluye:

- **Líneas añadidas**, marcadas con comentarios que indican su inclusión.
- **Líneas eliminadas**, convertidas en comentarios para su referencia.
- Verifica estándares de codificación, como:
    - Líneas largas.
    - Formato correcto de indentación y comentarios.
- Cuenta líneas físicas y lógicas, proporcionando métricas detalladas.

3. **Resultados:**
    - Se genera un archivo output.py que incluye las diferencias entre las versiones.
    - La consola muestra:

- Una tabla tabular con las líneas físicas y lógicas de las funciones y el bloque principal del programa.
- Totales generales del código, incluyendo líneas físicas y lógicas.

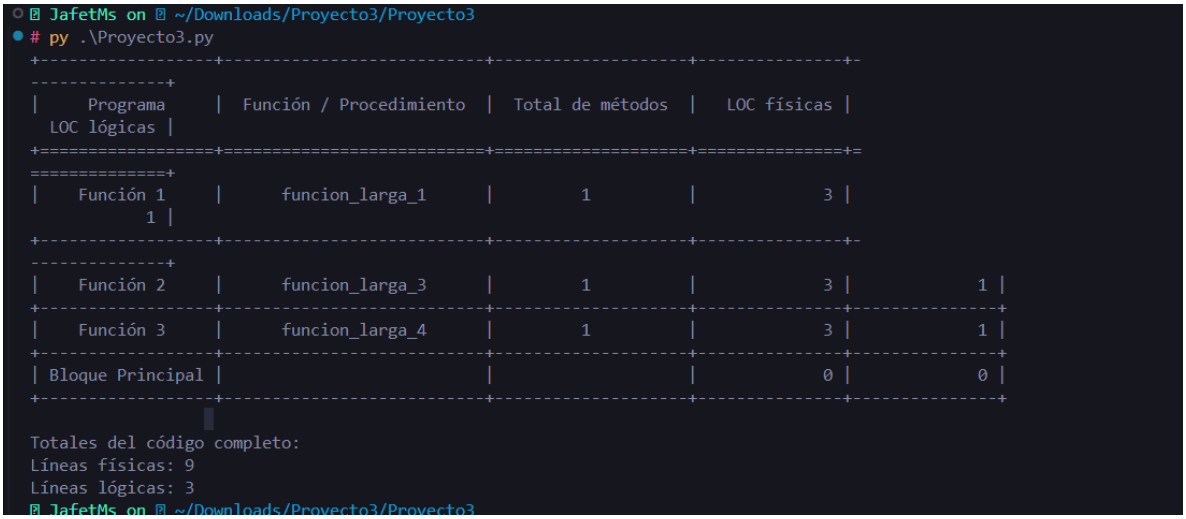
Ejemplo de Salida

Archivo Analizado:

Proyecto3.py

Salida Generada:

Para el conteo



Output comparación entre versiones:

```python

# Línea de abajo es añadida
# Archivo nuevo de ejemplo con líneas largas
def funcion_larga_1():
    texto_largo = "Este es un ejemplo de línea que excede los 80 caracteres, \
por lo tanto, debería ser dividida en fragmentos más pequeños para cumplir \
    con la regla de longitud de línea. Este es el final de la línea."
    print(texto_largo)


# Línea de abajo es añadida
def funcion_larga_3():

# Línea de abajo es añadida
    nueva_linea_larga = "Esta línea es nueva en la versión actual, y \
también supera los 80 caracteres, por lo que debe ser dividida adecuadamente \
para evitar que se rompa el código. Debemos asegurarnos de que el código \
    siga siendo funcional tras la división."

# Línea de abajo es añadida
    print(nueva_linea_larga)


# Línea de abajo es añadida
def funcion_larga_4():
    otra_linea_larga = "Aquí también tenemos una línea larga que supera los \
80 caracteres, lo que significa que debe ser dividida en varias partes cuando \
el programa se ejecute. Esta línea también debe ser manejada correctamente \
    por el código de comparación."
    print(otra_linea_larga)
# # Archivo anterior de ejemplo con líneas largas  # Línea borrada
# def funcion_larga_2():  # Línea borrada

# Líneas añadidas: 5
# Líneas borradas: 2
# Total de líneas: 7

```

Mensajes de Error Comunes

1. Error: "No se pudo encontrar el archivo"

- Asegúrese de seleccionar un archivo existente.

2. Excepciones Personalizadas:

- Ejemplo: Línea 3: Indentación incorrecta, debe ser múltiplo de 4 espacios.

- Descripción: Detalla el problema y la línea afectada.

Preguntas Frecuentes (FAQ)

1. ¿Qué es una línea lógica?

- Es una línea que incluye instrucciones clave como if, for, def, etc.

2. ¿Qué hacer si encuentro un error en el archivo?

- Revise el mensaje del error y corríjalo en el archivo Python.

3. ¿Se pueden analizar varios archivos a la vez?

- Actualmente, el script solo permite seleccionar un archivo a la vez.

Manual Técnico

Este manual describe en detalle la estructura, funcionalidad y lógica del script para el análisis y gestión de código Python. Este documento está destinado a desarrolladores y técnicos que necesiten entender, mantener o modificar el código.

---

1. Propósito del Script

El script permite:

- Leer y dividir un archivo de código Python en funciones y bloque principal.
- Contar líneas físicas y lógicas en un archivo.
- Detectar y ajustar líneas de código que exceden 80 caracteres.
- Comparar dos versiones de un archivo de código para identificar diferencias.
- Generar un resumen tabular del código analizado.

Estándares Validados:

- Longitud de líneas (<80 caracteres).
- Identificación y división de funciones y bloque principal.
- Uso correcto de estructuras de control (if, for, while, def, try, with).

---

2. Descripción Técnica

2.1. Estructura del Código

1. **Importaciones**
    - tkinter: Para gestionar la selección de archivos mediante una GUI.
    - filedialog: Permite al usuario seleccionar archivos desde un cuadro de diálogo.
    - tabulate: Genera tablas para presentar resultados de manera clara.
    - re: Procesa expresiones regulares para detectar URLs y rutas de archivos.
2. **Funciones Principales**

a) Leer_codigo(lineas)

- **Entrada**: Lista de líneas de código.
- **Proceso**: Divide las líneas en funciones y bloque principal.
- **Salida**: Listas de funciones, bloque principal y todas las líneas leídas.

b) Contar_lineas_fisicas(codigo)

- **Entrada**: Lista de líneas de código.
- **Proceso**: Ignora comentarios y líneas vacías, cuenta líneas útiles.
- **Salida**: Total de líneas físicas.

c) Contar_lineas_logicas(codigo)

- **Entrada**: Lista de líneas de código.
- **Proceso**: Identifica líneas que contienen instrucciones clave (if, def, etc.).
- **Salida**: Total de líneas lógicas.

d) Imprimir_bloques(funciones, bloque_principal)

- **Entrada**: Listas de funciones y bloque principal.
- **Proceso**: Genera un resumen tabular con las líneas físicas y lógicas de cada bloque.
- **Salida**: Tabla impresa en consola.

e) Contar_lineas_totales(codigo)

- **Entrada**: Lista de líneas de código completo.
- **Proceso**: Llama a las funciones de conteo para calcular los totales.
- **Salida**: Totales de líneas físicas y lógicas impresos en consola.

f) Detectar_urls_o_direcciones(linea)

- **Entrada**: Una línea de texto.
- **Proceso**: Usa expresiones regulares para detectar URLs o rutas de archivos.
- **Salida**: Booleano que indica si la línea contiene una URL o ruta.

g) Lineas_de_codigo(ruta_resultado)

- **Entrada**: Ruta de un archivo.
- **Proceso**: Ajusta líneas que superan los 80 caracteres y actualiza el archivo.
- **Salida**: Archivo modificado con líneas ajustadas.

h) Comparar_versiones(ruta_version_antigua, ruta_version_nueva)

- **Entrada**: Rutas de dos archivos (versión antigua y nueva).
- **Proceso**: Compara las versiones, identifica diferencias, y genera un archivo de salida con los cambios.
- **Salida**: Archivo de diferencias generado y ajustado.

3. **Bloque Principal (if __name__ == "__main__":)**
    - Gestiona la selección de archivos y llama a Comparar_versiones.

2.2. Lógica del Bloque Principal

El bloque principal utiliza tkinter para abrir cuadros de diálogo y permite al usuario seleccionar archivos a comparar. Luego, llama a Comparar_versiones para realizar el análisis y genera un archivo de salida con los resultados.

---

3. Herramientas y Configuración

3.1. Requisitos de Software

- **Python**: 3.6 o superior.
- **Bibliotecas requeridas**:
    - tabulate: Instalar con pip install tabulate.
    - tkinter: Incluido en la instalación estándar de Python.

3.2. Configuración

- Asegúrese de tener los archivos a comparar accesibles desde su sistema.
- Ejecute el script desde una terminal o IDE compatible con Python.

---

4. Lógica de Validación

4.1. Validación de URLs y Rutas

- **Expresión regular**: Detecta URLs y rutas de archivos.
- **Resultado esperado**: Identifica correctamente si una línea contiene una URL o ruta.

4.2. Validación de Longitud de Líneas

- **Regla**: Líneas no deben superar los 80 caracteres.
- **Acción**: Ajusta líneas largas dividiéndolas en múltiples partes.

5. Salida del Script

5.1. Resumen Tabular

```
# 
py .\Proyecto3.py
+------------------+--------------------------+--------------------+---------------+---------------+
|     Programa     |  Función / Procedimiento  |  Total de métodos  |   LOC físicas |   LOC lógicas |
+==================+==========================+====================+===============+===============+
|    Función 1     |      funcion_larga_1      |         1          |             3 |             1 |
+------------------+--------------------------+--------------------+---------------+---------------+
|    Función 2     |      funcion_larga_3      |         1          |             3 |             1 |
+------------------+--------------------------+--------------------+---------------+---------------+
|    Función 3     |      funcion_larga_4      |         1          |             3 |             1 |
+------------------+--------------------------+--------------------+---------------+---------------+
| Bloque Principal |                           |                    |             0 |             0 |
+------------------+--------------------------+--------------------+---------------+---------------+

Totales del código completo:
Líneas físicas: 9
Líneas lógicas: 3
```

5.2. Totales del Código Completo

```
Totales del código completo:
Líneas físicas: 9
Líneas lógicas: 3
```

5.3. Archivo de Diferencias

Un archivo output.py que contiene:

```python
# Línea de abajo es añadida
# Archivo nuevo de ejemplo con líneas largas
def funcion_larga_1():
    texto_largo = "Este es un ejemplo de línea que excede los 80 caracteres, \
por lo tanto, debería ser dividida en fragmentos más pequeños para cumplir \
    con la regla de longitud de línea. Este es el final de la línea."
    print(texto_largo)


# Línea de abajo es añadida
def funcion_larga_3():

# Línea de abajo es añadida
    nueva_linea_larga = "Esta línea es nueva en la versión actual, y \
también supera los 80 caracteres, por lo que debe ser dividida adecuadamente \
para evitar que se rompa el código. Debemos asegurarnos de que el código \
    siga siendo funcional tras la división."

# Línea de abajo es añadida
    print(nueva_linea_larga)


# Línea de abajo es añadida
def funcion_larga_4():
    otra_linea_larga = "Aquí también tenemos una línea larga que supera los \
80 caracteres, lo que significa que debe ser dividida en varias partes cuando \
el programa se ejecute. Esta línea también debe ser manejada correctamente \
    por el código de comparación."
    print(otra_linea_larga)
# # Archivo anterior de ejemplo con líneas largas  # Línea borrada
# def funcion_larga_2():  # Línea borrada

# Líneas añadidas: 5
# Líneas borradas: 2
# Total de líneas: 7
```

6. Posibles Mejoras

- **Soporte para múltiples archivos**: Permitir comparar y analizar varios archivos simultáneamente.
- **Interfaz gráfica**: Implementar una GUI completa para reemplazar la interacción por consola.
- **Opciones de configuración**: Permitir ajustes personalizados para los estándares de validación.

Unidad #2

Casos de prueba

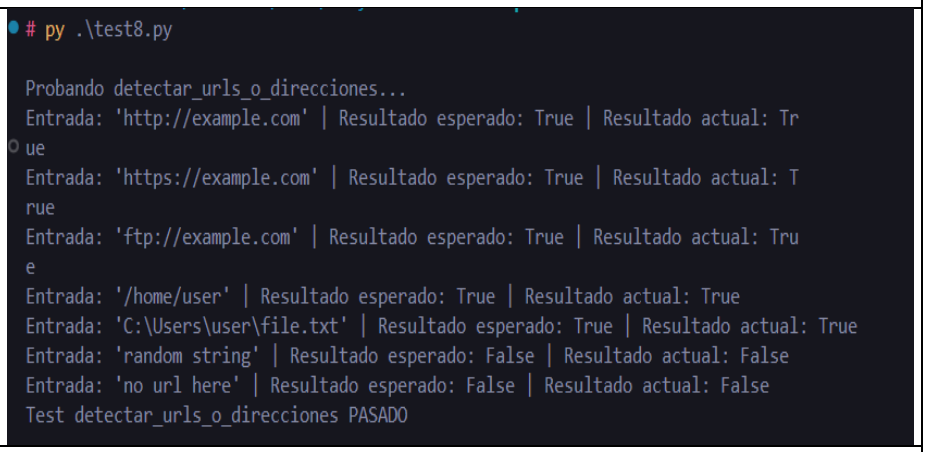
Los casos de prueba están diseñados para verificar que el código funcione como se espera en diferentes situaciones. Esto incluye comprobar que el programa cumpla con los requisitos establecidos y maneje correctamente las versiones de los archivos al compararlas.

El siguiente cuadro nos da una forma de registro de los casos de prueba:

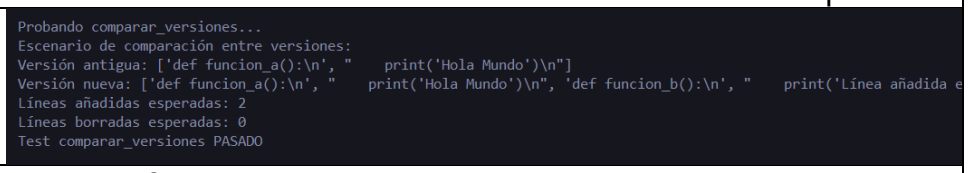
Tipo de prueba	
Nombre de la prueba/Numero objetivo de la prueba	Identificar único para la prueba
Descripción de la prueba	Describe las entradas y el procesamiento que va a tener el programa.
Condiciones de la prueba	Menciona las herramientas que se utilizaron para llevar a cabo en la prueba.
Resultados esperados	Lista de los resultados que se esperan obtener.
Resultados actuales	Lista de los resultados que se produjo durante la prueba.
Comentarios	Describe posibles comentarios que puedan ayudar a entender las pruebas.
Estado	Indicar el estado de la prueba: - Exitosa 1 - Fallida 0

Pruebas Unitarias:

Para los módulos de contar líneas lógicas y el de líneas físicas, usaremos los test del proyecto anterior para verificar que funcionen correctamente después de las modificaciones aplicadas.

Tipo de prueba Prueba unitaria	
Nombre de la prueba/Numero objetivo de la prueba	P1 / Prueba_A
Descripción de la prueba	Validar que la función detectar_urls_o_direcciones detecta correctamente URLs, rutas de archivos válidas, y cadenas que no representan URLs.
Condiciones de la prueba	Casos de prueba creados en memoria sin necesidad de archivos externos.
Resultados esperados	**Resultados esperados:** 1. La cadena "http://example.com" devuelve True. 2. La cadena "https://example.com" devuelve True. 3. La cadena "ftp://example.com" devuelve True. 4. La cadena "/home/user" devuelve True. 5. La cadena "C:\\Users\\user\\file.txt" devuelve True. 6. La cadena "random string" devuelve False. 7. La cadena "no url here" devuelve False.
Resultados actuales	
Comentarios	La función devuelve los resultados esperados.
Estado	1

Tipo de prueba Prueba unitaria	
Nombre de la prueba/Num	P2 / Prueba_B

ero objetivo de la prueba	
Descripción de la prueba	Validar que la función comparar_versiones detecta correctamente las líneas añadidas entre dos versiones comparativas.
Condiciones de la prueba	Escenarios creados en memoria para simular las comparaciones directamente sin archivos externos.
Resultados esperados	I. Se detectan dos líneas añadidas en la comparación entre las dos versiones. II. No se detectan líneas borradas en el escenario de prueba.
Resultados actuales	
Comentarios	El código funcionó correctamente y simuló la comparación esperada.
Estado	1

Pruebas de integración:

Tipo de prueba Prueba integración	
Nombre de la prueba/Numero objetivo de la prueba	P3 / Prueba_ABC Validar que el código maneja correctamente la detección de funciones, bloques principales, el conteo de líneas físicas y lógicas, así como la comparación de versiones.
Descripción de la prueba	**Leer funciones y bloques principales:** Validar que Leer_codigo detecte correctamente las funciones y el bloque principal de un conjunto de líneas de prueba. **Contar líneas físicas y lógicas:** Verificar que las funciones Contar_lineas_fisicas y Contar_lineas_logicas calculen las métricas esperadas. **Simular comparación de versiones:** Probar que comparar_versiones detecte adecuadamente las líneas añadidas y borradas entre dos versiones simuladas.
Condiciones de la prueba	Un script test_9.py que contiene las pruebas para ejecutar los módulos en conjunto sin necesidad de archivos externos.

Resultados esperados	Leer código: Detectar 2 funciones (funcion_a, funcion_b) y 1 bloque principal correctamente. Conteo: Total de 6 líneas físicas y 3 líneas lógicas. Comparar versiones: • **Líneas añadidas:** 2. • **Líneas**
Resultados actuales	
Comentarios	El código cumple con los estándares establecidos para detección de funciones, bloques principales y análisis de métricas. Además, la comparación de versiones simula correctamente los cambios.
Estado	1

Unidad #3

Estimación de tamaño

Se utilizará la métrica de Puntos Funcionales para el tamaño y complejidad del sistema.

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

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

Asignación del grado de complejidad con base a 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:

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

La siguiente plantilla se utilizará 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	

Valores posibles para dar:

0-Irrelevante o sin influencia

1-Incidental

2-Moderado

3-Medio

4-Significativo

Ecuación para el factor de complejidad técnica:

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

Ecuación para el conteo de Puntos Funcionales Ajustado

$$FP = UFC \cdot TFC$$

Por cada punto funcional encontrado se considerará:

$$1pf\ tarda\ aprox\ 10\ hrs$$

1. **Descripción del sistema 3:**

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

   • Comparar

   - Si una línea está contenida en ambas versiones, es una línea original y no ha sufrido cambio.
   - Si una línea está en una versión, pero no en su versión previa es una línea añadida.
   - Si una línea está en una versión previa, pero no en la siguiente es una línea borrada.

   • Contar los cambios de líneas añadidas y borradas.

   • Por cada línea añadida

   - Comparar si es una pequeña modificación o si realmente es totalmente nueva.

   • Para no complicar las cosas

   - Si las líneas se mueven de lugar, éstas se considerarán borradas y añadidas.
   - Cada línea no debe ocupar más allá de 80 caracteres
   - Por lo que se tiene que formatear aquellas líneas que rebasen los 80 caracteres en dos o más líneas. Esta línea seguirá contando como una única línea de código
   - Etiquetar cada línea añadida en la nueva versión al final de esta con un comentario
   - Etiquetar cada línea borrada en la versión anterior al final de esta con un comentario

   •Realizar el conteo como lo indica el ejercicio 2.

Programa	Clase	Total de métodos	LOC físicas por clase	Total de LOC físicas del programa
123	ABC	3	86	
	DEF	4	92	
				178
456	…	…	…	

Programa	Función / Procedimiento	Total de métodos	LOC físicas p	Total de LOC físicas del programa
123	ABC	1	86	
	DEF	1	92	
				178
456	…	…	…	

Parte 1: Elementos identificados:

1- Entrada de los archivos Python a evaluar en el conteo y su comparación entre versiones.
2- Salida de tabla del total de líneas físicas, lógicas y si cumple con el estándar de codificación de cada módulo y el total de líneas al final.

Parte 2: Conteo de los puntos de Función sin ajuste:

Elemento	Cantidad	Peso	Total
Entradas Externas	2	4	8
Salidas Externas	3	5	15
Consultas Externas	0	0	0
Archivos Lógicos Externos	0	0	0
Archivos Lógicos Internos	1	7	7
Total			30

$$UFC = (2 \times 4) + (3 \times 5) + (1 \times 7) = 30$$

$$UFC = \ 30$$

Componentes del factor de complejidad técnica		
F_1	Fiabilidad de la copia de seguridad y recuperación	0
F_2	Funciones distribuidas	3
F_3	Configuración utilizada	0
F_4	Facilidad operativa	3
F_5	Complejidad de interfaz	0
F_6	Reutilización	3
F_7	Instalaciones múltiples	0
F_8	Comunicaciones de datos	0
F_9	Desempeño	0
F_{10}	Entrada de datos en línea	0
F_{11}	Actualización en línea	0
F_{12}	Procesamiento complejo	0
F_{13}	Facilidad de instalación	1
F_{14}	Facilidad de cambio	3
	Total	13

$$TFC = 0.65 + 0.01(13) = 0.78$$

$$FP = 30 \cdot\ 0.78 = 23.4\ \approx 23$$

Tiempo esperado que puede tomar realizar el sistema:

$$Tiempo_{hrs} = 23 \cdot 10\ hrs = 230\ hrs\ \approx 230\ hrs$$

$$Tiempo_{dias} = \frac{230}{24} = 9.583\ días\ \approx 10\ días$$

Considerando que dos personas trabajaran en el desarrollo del programa podemos deducir el siguiente valor:

$$Tiempo_{dias} = \frac{10\ dias}{2\ desarrolladores}$$

5 días para cada desarrollado

Unidad #4

Mantenimiento, documentación de las decisiones y modificaciones

Introducción

Resumen de Cambios

Este apartado tiene como objetivo detallar las decisiones y modificaciones implementadas en el código de la aplicación diseñada para comparar y contar las diferencias entre dos versiones de un archivo. El propósito de estas modificaciones es mejorar la capacidad del programa para identificar cambios entre las versiones de un código, optimizar su funcionamiento y asegurar que cumpla con los requisitos establecidos en el Proyecto 3.

El programa tiene como propósito principal comparar dos versiones de un archivo y determinar qué líneas han sido añadidas, borradas o no modificadas.

Resumen de cambios

1. Comparación de versiones:

Se ha implementado un proceso para comparar dos versiones de un archivo. El programa ahora determina si una línea de código está presente en ambas versiones (línea original), en solo la nueva versión (línea añadida), o solo en la versión anterior (línea borrada).

2. Conteo de cambios:

El programa ahora cuenta las líneas añadidas y borradas entre las versiones comparadas, permitiendo una visualización clara de los cambios.

3.Modificaciones pequeñas vs nuevas:

Se ha establecido un criterio para identificar líneas modificadas, determinando si una línea añadida es una modificación pequeña o completamente nueva. Las líneas movidas de lugar se consideran borradas y añadidas en vez de modificadas.

4. Formateo de líneas largas:

El programa ahora asegura que ninguna línea exceda los 80 caracteres. Si alguna línea sobrepasa este límite, se divide en dos o más líneas, pero se considera como una única línea de código para los efectos de conteo.

5. Etiquetado de líneas modificadas:

Cada línea **añadida** en la nueva versión se etiqueta con un comentario al final, indicando que es una línea **añadida**.

De manera similar, cada línea **borrada** en la versión anterior es etiquetada con un comentario indicando que es una línea **borrada**.

6. Interfaz Gráfica:

La interfaz gráfica ha sido simplificada. Ahora el programa permite al usuario seleccionar los archivos Python a través de un cuadro de diálogo de Tkinter, mejorando la experiencia del usuario.

Motivos de las Modificaciones

1. Claridad y Legibilidad del Código:

Las modificaciones realizadas tienen como objetivo mejorar la claridad y legibilidad del código. Esto incluye una mejor organización y la inclusión de comentarios claros que faciliten el mantenimiento y la comprensión del código a largo plazo.

2. Cumplimiento de Estándares de Programación:

Asegurarse de que el código cumpla con las mejores prácticas es crucial. Se implementaron mejoras en el formato y la estructura del código, lo que incluye el etiquetado adecuado de las líneas y el formateo de las líneas largas para que se ajusten a los estándares de codificación.

3. Mejora de la Precisión y Exactitud en los Cálculos:

El cálculo de las líneas añadidas y borradas se ha mejorado, asegurando que las líneas que se mueven de lugar sean correctamente etiquetadas como borradas y añadidas.

Conclusión

Las modificaciones realizadas han mejorado la funcionalidad y eficiencia del programa al permitir una comparación más precisa de las versiones de los archivos. Las nuevas características, como el conteo de líneas añadidas y borradas, el etiquetado de las líneas modificadas, y el formateo adecuado de líneas largas, aseguran que el programa sea más robusto y eficiente. Además, la interfaz gráfica mejorada proporciona una experiencia de usuario más amigable. En general, estos cambios no solo optimizan el proceso de comparación, sino que también garantizan que el programa siga las mejores prácticas de desarrollo, asegurando su facilidad de mantenimiento y expansión en el futuro.

Unidad #5

Aseguramiento de la calidad

Inspecciones del Proyecto 3

Claro, aquí tienes la distribución de los roles con los cuatro nombres proporcionados:

Inspección 1

- **Fecha de la inspección**: 02 / 12 / 24
- **Módulo inspeccionado**: Comparación de versiones de archivos (función comparar_versiones).
- **Tipo de inspección**: Revisión de código.
- **Objetivo de la inspección**: Identificar defectos en la lógica de comparación entre las versiones de un archivo y en el conteo de líneas añadidas o borradas.

Participantes y roles:

- **Moderador**: Eduardo Alberto González Ortega.
- **Inspector**: Jafet Andree Mena Solís.
- **Autor**: Rogerio Emmanuel Canto Romero.
- **Secretario**: Santiago Efraín Itzincab Poot.
- **Lector**: Eduardo Alberto González Ortega.

Resumen de la inspección:

- **Defectos encontrados**: 2

1. **Descripción**: No se manejan correctamente las líneas que contienen diferencias mínimas entre las versiones (como un cambio de comentario o un cambio de espacio en blanco).
    - **Impacto**: Podría generar un conteo incorrecto de líneas añadidas.
    - **Acción propuesta**: Implementar un filtro que ignore los cambios pequeños y trate las líneas de manera más precisa.
    - **Responsable**: Jafet Andree Mena Solís.
2. **Descripción**: Las líneas que han sido reordenadas en el código no se identifican correctamente como añadidas y borradas.
    - **Impacto**: Puede provocar errores al contabilizar las diferencias entre versiones.
    - **Acción propuesta**: Añadir un proceso de detección para las líneas reordenadas que las considere tanto borradas como añadidas.
    - **Responsable**: Santiago Efraín Itzincab Poot.

Plazo estimado para las correcciones: 1 día.

Revisión de seguimiento:

- **Seguimiento**: El código fue corregido y evaluado.
- **Notas adicionales**: Se realizó una segunda revisión y los problemas fueron corregidos con éxito.

Inspección 2

- **Fecha de la inspección**: 03 /12/ 14

- **Módulo inspeccionado**: Formateo y conteo de líneas (función lineas_de_codigo).
- **Tipo de inspección**: Revisión de código.
- **Objetivo de la inspección**: Validar que el programa maneje correctamente las líneas largas y las divida en partes menores para cumplir con el estándar de 80 caracteres por línea.

Participantes y roles:

- **Moderador**: Santiago Efraín Itzincab Poot.
- **Inspector**: Jafet Andree Mena Solís.
- **Autor**: Rogerio Emmanuel Canto Romero.
- **Secretario**: Eduardo Alberto González Ortega.
- **Lector**: Jafet Andree Mena Solís.

Resumen de la inspección:

- **Defectos encontrados**: 1
    - **Descripción**: Algunas líneas largas no se dividen correctamente debido a un error en la detección de los espacios más cercanos para cortar las líneas.
        - **Impacto**: Esto puede causar que las líneas largas no se ajusten al estándar de 80 caracteres.
        - **Acción propuesta**: Mejorar la lógica para detectar y dividir correctamente las líneas largas.
        - **Responsable**: Rogerio Emmanuel Canto Romero.

Plazo estimado para las correcciones: 1 día.

Revisión de seguimiento:

- **Seguimiento**: El código fue corregido y evaluado.

- **Notas adicionales**: Se realizó una segunda revisión y el problema fue corregido satisfactoriamente.

Inspección 3

- **Fecha de la inspección**: 04 /12 /24
- **Módulo inspeccionado**: Detección de URLs y rutas (función detectar_urls_o_direcciones).
- **Tipo de inspección**: Revisión de código.
- **Objetivo de la inspección**: Validar que el programa detecte correctamente las URLs y direcciones de archivos en el código fuente.

Participantes y roles:

- **Moderador**: Eduardo Alberto González Ortega.
- **Inspector**: Santiago Efraín Itzincab Poot, Rogerio Emmanuel Canto Romero.
- **Autor**: Rogerio Emmanuel Canto Romero.
- **Secretario**: Santiago Efraín Itzincab Poot.
- **Lector**: Jafet Andree Mena Solis

Resumen de la inspección:

- **Defectos encontrados**: 1
    - **Descripción**: La expresión regular utilizada para detectar rutas de archivos no cubre ciertos casos de rutas relativas en sistemas Unix.
        - **Impacto**: Puede dejar de detectar algunas rutas, afectando la precisión del análisis.
        - **Acción propuesta**: Mejorar la expresión regular para detectar correctamente rutas relativas en sistemas Unix.

- **Responsable**: Rogerio Emmanuel Canto Romero.

Plazo estimado para las correcciones: 1 día.

Revisión de seguimiento:

- **Seguimiento**: El código fue corregido y evaluado.
- **Notas adicionales**: Se realizó una segunda revisión y los problemas fueron corregidos exitosamente.

I**nspección 4**

- **Fecha de la inspección**: 05 / 12 / 24
- **Módulo inspeccionado**: Interfaz gráfica de selección de archivos (función abrir_dialogo_archivo).
- **Tipo de inspección**: Revisión de código.
- **Objetivo de la inspección**: Verificar que el cuadro de diálogo para seleccionar archivos se abra correctamente en la carpeta que contiene el archivo que el usuario desea seleccionar, en lugar de la carpeta predeterminada del sistema.

  Participantes y roles:

- **Moderador**: Jafet Andree Mena Solís.
- **Inspector**: Eduardo Alberto González Ortega, Santiago Efraín Itzincab Poot.
- **Autor**: Rogerio Emmanuel Canto Romero.
- **Secretario**: Eduardo Alberto González Ortega.
- **Lector**: Santiago Efraín Itzincab Poot.

  Resumen de la inspección:

- **Defectos encontrados**: 1
    - **Descripción**: El cuadro de diálogo se abre en la carpeta predeterminada del sistema, no en la carpeta donde se encuentra el archivo que el usuario desea seleccionar.
        - **Impacto**: El usuario tiene que navegar a través de las carpetas para encontrar el archivo correcto, lo que puede resultar incómodo y consumir más tiempo.
        - **Acción propuesta**: Se debe modificar el cuadro de diálogo para que se abra en la carpeta de trabajo actual o la última carpeta seleccionada, en lugar de la carpeta predeterminada del sistema. De esta forma, el proceso de selección de archivos será más rápido y eficiente para el usuario.
        - **Responsable**: Rogerio Emmanuel Canto Romero.

    Plazo estimado para las correcciones: 1 día.

    Revisión de seguimiento:

- **Seguimiento**: El código fue corregido y evaluado con éxito.
- **Notas adicionales**: Se realizó una segunda revisión y el problema fue corregido. Ahora el cuadro de diálogo se abre correctamente en la carpeta de trabajo actual, mejorando la experiencia del usuario.

Estas inspecciones permiten mantener una alta calidad del código mediante la identificación y corrección de defectos. La participación de todos los miembros en las inspecciones asegura que las modificaciones y mejoras se realicen de manera eficiente y sin contratiempos.