Logotipo

Descripción generada automáticamente

**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](#_Toc184385133)

[Manual de usuario 3](#_Toc184385134)

[Manual Técnico 7](#_Toc184385135)

[Unidad #2 13](#_Toc184385136)

[Casos de prueba 13](#_Toc184385137)

[**Pruebas Unitarias:** 14](#_Toc184385138)

[**Pruebas de integración:** 16](#_Toc184385139)

[Unidad #3 20](#_Toc184385140)

[Estimación de tamaño 20](#_Toc184385141)

[Unidad #4 25](#_Toc184385142)

[Mantenimiento, documentación de las decisiones y modificaciones 25](#_Toc184385143)

[Unidad #5 27](#_Toc184385144)

[Aseguramiento de la calidad 27](#_Toc184385145)

[Inspecciones 27](#_Toc184385146)

# 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).

1. Archivos de Entrada:

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

1. 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

1. 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.

1. 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.

1. 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.

1. **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.

1. **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**

Texto

Descripción generada automáticamente

**Output comparación entre versiones:**

**Texto

Descripción generada automáticamente**

**Mensajes de Error Comunes**

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

* Asegúrese de seleccionar un archivo existente.

1. 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.

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

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

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

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

## Manual Técnico

Este manual técnico describe en detalle la estructura, funcionalidad y lógica del script de análisis 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:

* Verificar estándares de codificación en un archivo Python.
* Identificar funciones y bloques principales en el código.
* Contar líneas físicas y lógicas de código.
* Presentar resultados en un formato tabular.

**Estándares Validados**

* Longitud de líneas (<80 caracteres).
* Formato de comentarios.
* Indentación (múltiplos de 4 espacios).
* Uso de identificadores en minúsculas.
* Validez de nombres de funciones.
* Correcto uso de estructuras de control.

1. Descripción Técnica
   1. Estructura del Código
2. Importaciones

* tkinter: Usado para manejar la selección de archivos mediante una GUI.
* filedialog: Permite al usuario seleccionar un archivo desde un cuadro de diálogo.
* tabulate: Genera tablas para los resultados.
* re: Procesa expresiones regulares para la validación de comentarios y estructuras.

1. Funciones Principales

* Verificar\_estandares\_archivo(archivo):

Valida estándares de codificación línea por línea.

* Leer\_codigo(filename):

Divide el archivo en funciones y el bloque principal.

* Contar\_lineas\_fisicas(codigo):

Cuenta las líneas no vacías ni comentarios.

* Contar\_lineas\_logicas(codigo):

Cuenta líneas que representan instrucciones clave.

* Imprimir\_bloques(funciones, bloque\_principal):

Muestra un resumen tabular de las funciones y el bloque principal.

* Contar\_lineas\_totales(codigo):

Calcula el total de líneas físicas y lógicas del programa.

1. Bloque Principal (if \_\_name\_\_ == "\_\_main\_\_":)

* Gestiona la interacción con el usuario y ejecuta las funciones anteriores.
  1. Detalle de Funciones
  2. Verificar\_estandares\_archivo(archivo)
* Entrada: Ruta de un archivo .py.
* Procesos:
  + Valida longitud de línea, formato de comentarios, indentación, uso de identificadores, y estructuras de control.
* Salida:
  + Mensaje en consola si el archivo cumple estándares o una excepción detallada.
  1. Leer\_codigo(filename)
* Entrada: Ruta de un archivo .py.
* Procesos:
  + Lee el archivo línea por línea.
  + Clasifica el código en funciones y bloque principal.
* Salida:
  + Lista de funciones con sus líneas y bloque principal.
  1. Contar\_lineas\_fisicas(codigo)
* Entrada: Lista de líneas de código.
* Procesos:
  + Ignora comentarios y líneas vacías.
* Salida: Número total de líneas físicas.
  1. Contar\_lineas\_logicas(codigo)
* Entrada: Lista de líneas de código.
* Procesos:
  + Identifica líneas que comienzan con palabras clave (if, for, while, etc.).
* Salida: Número total de líneas lógicas.
  1. Imprimir\_bloques(funciones, bloque\_principal)
* Entrada: Listas de funciones y bloque principal.
* Procesos:
  + Calcula LOC físicas y lógicas para cada función y bloque.
  + Genera una tabla con los datos.
* Salida: Tabla tabulada impresa en consola.
  1. Contar\_lineas\_totales(codigo)
* Entrada: Lista de líneas del archivo completo.
* Procesos:
  + Llama a las funciones de conteo para líneas físicas y lógicas.
* Salida: Totales de líneas físicas y lógicas impresos en consola.
  1. Lógica del Bloque Principal

Texto

Descripción generada automáticamente

1. Herramientas y Configuración
   1. Requisitos de Software

* Python 3.6 o superior.
* Bibliotecas requeridas:
  + tabulate: Instalar con pip install tabulate.

1. Lógica de Validación
   1. Validación de Comentarios

* Debe haber un espacio después de #.
* Los comentarios deben iniciar con una mayúscula.
  1. Validación de Identificadores
* Nombres de variables deben estar en minúsculas.
  1. Validación de Indentación
* La indentación debe ser un múltiplo de 4 espacios.

1. Salida del Script
   1. Verificación de Estándares:

* Mensaje en consola indicando problemas o confirmando que los estándares se cumplen.
  1. Resumen Tabular:

Ejemplo:

Imagen que contiene Calendario

Descripción generada automáticamente

* 1. Totales del Código Completo:
* Líneas físicas y lógicas totales.

1. Posibles Mejoras

* Soporte para múltiples archivos: Analizar varios archivos en una sola ejecución.
* Interfaz gráfica completa: Reemplazar la interacción por consola con una GUI.

# Unidad #2

## Casos de prueba

Los casos de prueba están diseñados para verificar que el código funcione sin errores, tal y como se debe esperar en diferentes situaciones. Así como que se verifique el funcionamiento del programa por parte del usuario.

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  Validar que el código sigue correctamente el estándar de conteo de líneas físicas. |
| Descripción de la prueba | El programa evalúa 5 escenarios predefinidos, diseñados para verificar los casos más relevantes relacionados con líneas físicas.   |  | | --- | | **1. Comentarios:** Solo comentarios simples Se espera que no sean detectados como líneas físicas. |  |  |  | | --- | --- | |  | **2. Declaraciones:** Declaraciones de variables, asignaciones e impresiones en consola, ignorando comentarios. |  |  |  | | --- | --- | |  | **3. Importaciones:** Involucra importaciones de librerías o archivos externos, que deben contarse como líneas físicas. |  |  |  | | --- | --- | |  | **4. Lógicas:** Contiene funciones, sentencias lógicas, impresiones en consola, docstrings y comentarios, sin afectar el conteo. |  |  |  | | --- | --- | |  | **5. Completo:** Combina todos los escenarios previos en un solo caso, validando la consistencia en presencia de múltiples casos. | |
| Condiciones de la prueba | Un script test4.py donde se tienen guardado las pruebas para realizar de manera automática sin necesidad de generar una por una. |
| Resultados esperados | 1. 0 líneas físicas  2. 5 líneas físicas  3. 4 líneas físicas  4. 11 líneas físicas  5. 15 líneas físicas |
| Resultados actuales |  |
| Comentarios | El código funciona correctamente en el conteo de líneas físicas. |
| Estado | 1 |

|  |  |
| --- | --- |
| Tipo de prueba  Prueba unitaria | |
| Nombre de la prueba/Numero objetivo de la prueba | P2 / Prueba\_B  Validar que el código sigue correctamente el estándar de conteo de líneas lógicas. |
| Descripción de la prueba | Resultados esperados por escenario   1. if: Contar las líneas lógicas asociadas a la palabra clave if, reconociendo las condiciones evaluadas como una sola línea lógica. 2. for: Identificar ciclos for, donde el encabezado del bucle se cuenta como una línea lógica, sin incluir las operaciones dentro del bloque. 3. while: Contar la línea lógica correspondiente al encabezado de un bucle while, ignorando el contenido del bloque repetitivo. 4. def: Reconocer la línea lógica asociada a la declaración de funciones mediante la palabra clave def. Cada función es una línea lógica. 5. try: Evaluar las líneas lógicas correspondientes al bloque try que gestiona excepciones, considerando el encabezado try y no los bloques except. 6. with: Contar la línea lógica asociada a bloques with, que gestionan el contexto de operaciones con recursos. 7. Combinado: Evaluar un archivo que contiene todas las estructuras anteriores, verificando que cada encabezado if, for, while, def, class, try, y with sea identificado correctamente como una línea lógica. |
| Condiciones de la prueba | Un script test5.py donde se tienen guardado las pruebas para realizar de manera automática sin necesidad de generar una por una. |
| Resultados esperados | 1. 1 líneas lógicas 2. 1 líneas lógicas 3. 1 líneas lógicas 4. 2 líneas lógicas 5. 1 líneas lógicas 6. 1 líneas lógicas 7. 4 líneas lógicas |
| Resultados actuales |  |
| Comentarios | El código funciona correctamente en el conteo de líneas físicas. |
| 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 sigue correctamente el estándar de conteo de líneas físicas y lógicas, además que con el bloque C se pretende contar las de todo el programa. |
| Descripción de la prueba | Resultados esperados por escenario   1. if: Contar las líneas lógicas asociadas a la palabra clave if, reconociendo las condiciones evaluadas como una sola línea lógica. 2. for: Identificar ciclos for, donde el encabezado del bucle se cuenta como una línea lógica, sin incluir las operaciones dentro del bloque. 3. while: Contar la línea lógica correspondiente al encabezado de un bucle while, ignorando el contenido del bloque repetitivo. 4. def: Reconocer la línea lógica asociada a la declaración de funciones mediante la palabra clave def. Cada función es una línea lógica. 5. try: Evaluar las líneas lógicas correspondientes al bloque try que gestiona excepciones, considerando el encabezado try y no los bloques except. 6. with: Contar la línea lógica asociada a bloques with, que gestionan el contexto de operaciones con recursos. 7. Combinado: Evaluar un archivo que contiene todas las estructuras anteriores, verificando que cada encabezado if, for, while, def, class, try, y with sea identificado correctamente como una línea lógica. |
| Condiciones de la prueba | Un script test6.py donde se tienen guardado las pruebas para realizar de manera automática sin necesidad de generar una por una. |
| Resultados esperados | 1. 6 líneas físicas, 1 líneas lógicas  2. 2 líneas físicas, 1 líneas lógicas  3. 3 líneas físicas, 1 líneas lógicas  4. 4 líneas físicas, 2 líneas lógicas  5. 6 líneas físicas, 1 líneas lógicas  6. 2 líneas físicas, 1 líneas lógicas  7. 14 líneas físicas, 6 líneas lógicas |
| Resultados actuales |  |
| Comentarios | El código funciona correctamente en el conteo de líneas físicas y lógicas. |
| Estado | 1 |
| Tipo de prueba  Prueba integración | |
| Nombre de la prueba/Numero objetivo de la prueba | P3 / Prueba\_completa  Validar que el código sigue correctamente el estándar de conteo de líneas físicas y lógicas, dividido en bloques y función principal. |
| Descripción de la prueba | Escenario:  El escenario es un código que cuenta con dos funciones las cuales se espera que el código las detecte como funciones independientes y haga el conteo normal. |
| Condiciones de la prueba | Un script test7.py donde se tienen guardado las pruebas para realizar de manera automática sin necesidad de generar una por una. |
| Resultados esperados | Función 1: suma, 1, 2, 1  Función 2: resta, 1, 2, 1  Bloque 3: vacío  Total Físicas: 4  Total Lógicas: 2 |
| Resultados actuales | Diagrama  Descripción generada automáticamente con confianza media |
| Comentarios | El código funciona correctamente en el conteo de líneas físicas y lógicas dividido en bloques. |
| 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:

La siguiente plantilla se utilizará para calcular el factor de complejidad técnica para los puntos funcionales sin ajustar:

Valores posibles para dar:

0-Irrelevante o sin influencia

1-Incidental

2-Moderado

3-Medio

4-Significativo

5-Esencial

|  |  |  |
| --- | --- | --- |
| Componentes del factor de complejidad técnica | | |
|  | Fiabilidad de la copia de seguridad y  recuperación |  |
|  | Funciones distribuidas |  |
|  | Configuración utilizada |  |
|  | Facilidad operativa |  |
|  | Complejidad de interfaz |  |
|  | Reutilización |  |
|  | Instalaciones múltiples |  |
|  | Comunicaciones de datos |  |
|  | Desempeño |  |
|  | Entrada de datos en línea |  |
|  | Actualización en línea |  |
|  | Procesamiento complejo |  |
|  | Facilidad de instalación |  |
|  | Facilidad de cambio |  |
| Total | |  |

Ecuación para el factor de complejidad técnica:

Ecuación para el conteo de Puntos Funcionales Ajustado

Por cada punto funcional encontrado se considerará:

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.

Tabla

Descripción generada automáticamente

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 |

|  |  |  |
| --- | --- | --- |
| Componentes del factor de complejidad técnica | | |
|  | Fiabilidad de la copia de seguridad y  recuperación | 0 |
|  | Funciones distribuidas | 3 |
|  | Configuración utilizada | 0 |
|  | Facilidad operativa | 3 |
|  | Complejidad de interfaz | 0 |
|  | Reutilización | 3 |
|  | Instalaciones múltiples | 0 |
|  | Comunicaciones de datos | 0 |
|  | Desempeño | 0 |
|  | Entrada de datos en línea | 0 |
|  | Actualización en línea | 0 |
|  | Procesamiento complejo | 0 |
|  | Facilidad de instalación | 1 |
|  | Facilidad de cambio | 3 |
| Total | | 13 |

Tiempo esperado que puede tomar realizar el sistema:

# 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.