**Reporte de Modificaciones y Decisiones Técnicas**

**Del Proyecto 1 al Proyecto 2**

La transición del proyecto "Contador Líneas" a "Líneas por Clase" requirió modificaciones significativas en múltiples módulos para soportar el nuevo enfoque de análisis centrado en clases, eliminando completamente el conteo de líneas lógicas y priorizando el análisis a nivel de clase.

En la estructura base, la clase Nodo se extendió con el método obtener\_nombre\_clase() para identificar y extraer nombres de clases del código fuente, implementando un análisis sintáctico robusto que maneja casos especiales y malformaciones en las declaraciones de clase. Esto permitió la identificación única de clases como unidades de análisis, manteniendo la funcionalidad jerárquica padre-hijo existente para preservar la estructura del árbol sintáctico.

El módulo Metricas experimentó una reestructuración profunda con la introducción de la clase MetricasClase, diseñada para almacenar información detallada sobre clases individuales incluyendo nombre, métodos y líneas físicas. La clase MetricasArchivo se modificó para manejar una lista de MetricasClase en lugar de contadores simples, incorporando el cálculo automático del total de líneas físicas y facilitando la agregación de métricas a nivel de archivo.

El FormateadorMetricas se adaptó para visualizar la nueva estructura jerárquica de datos, expandiendo el sistema de colores y rediseñando la tabla para mostrar cinco columnas con información detallada de clases y métodos. La visualización ahora incluye una fila principal con la primera clase, filas secundarias para clases adicionales, y totales de líneas físicas, mejorando significativamente la legibilidad y organización de los datos.

En el componente de análisis sintáctico, el ArbolSintactico incorporó tres métodos cruciales: obtener\_nodos\_clase(), obtener\_nodos\_metodos(), y obtener\_nodo\_otros(). Estos métodos permiten extraer y analizar clases y métodos específicos, mientras manejan el código fuera de clases como una unidad especial, garantizando un análisis completo del archivo fuente.

El `Analizador` se modificó significativamente, eliminando el contador de líneas lógicas y añadiendo métodos específicos para el análisis de clases. Los métodos \_analizar\_clases y \_analizar\_clase se implementaron para procesar clases individuales y código fuera de clases, extrayendo nombres, contando métodos y calculando líneas físicas con una precisión mejorada en la detección de estructuras de código.

El AlmacenamientoMetricas se adaptó para manejar la nueva estructura de datos, modificando los métodos de serialización y deserialización para trabajar con objetos MetricasClase. La conversión entre JSON y objetos se actualizó para reflejar el nuevo modelo de datos centrado en clases, implementando validaciones adicionales para garantizar la integridad de los datos almacenados.

El módulo principal mantuvo su estructura general, pero se actualizaron las rutas de importación y nombres de comandos para reflejar la nueva estructura del proyecto. La ruta de almacenamiento de métricas se modificó a lineas\_por\_clase\_registro.json para separar los datos de diferentes sistemas de análisis, mejorando la organización y mantenimiento del proyecto a largo plazo.

**Del Proyecto 2 al Proyecto 3**

La transición del proyecto "Líneas por Clase" a "Analizador Cambios" requirió modificaciones sustanciales en los módulos existentes para implementar la capacidad de detectar y analizar cambios entre versiones de código. El enfoque principal fue añadir funcionalidades de mapeo y rastreo de líneas mientras se preservaba la estructura base del sistema.

La clase Nodo se extendió con el atributo numero\_nodo en el constructor, permitiendo asociar cada nodo con su línea correspondiente en el archivo original. Esta modificación fue crucial para rastrear la posición exacta de cada línea en el código fuente, facilitando la identificación y marcado de cambios entre versiones, mientras se mantenía la funcionalidad existente de la estructura jerárquica.

El ConstructorArbol experimentó cambios significativos con la introducción del diccionario arbol\_a\_lineas para mapear nodos a posiciones en el código. Se implementaron elementos de control como el contador numero\_nodo, la lista posicion\_linea y la variable identacion\_previa. El proceso de construcción ahora mantiene un registro detallado de líneas múltiples y espacios en blanco, permitiendo un mapeo preciso entre nodos y líneas físicas.

En el ArbolSintactico, la adición del atributo mapeo\_lineas permite mantener la relación entre nodos y sus posiciones en el archivo original. Esta modificación se integró preservando todos los métodos existentes como obtener\_nodos\_clase, obtener\_nodos\_metodos y obtener\_nodo\_otros, añadiendo la capacidad de rastreo de líneas sin alterar la funcionalidad base.

El Analizador incorporó el formateador de líneas a través de la clase FormateadorLinea, añadiendo el método formatear\_codigo para procesar cada línea del código fuente. Se modificó \_obtener\_codigo para utilizar el formateador durante la lectura, asegurando un procesamiento consistente antes de la construcción del árbol sintáctico y permitiendo una detección precisa de cambios.

El módulo principal se transformó para manejar la comparación entre versiones, reescribiendo procesar\_archivo como procesar\_archivos para manejar dos archivos simultáneamente. Se añadió el argumento -cc para mostrar resúmenes de cambios y se incorporó el manejo de la excepción ExcepcionFormateo. El sistema ahora genera archivos comentados con sufijo "\_comentado" que contienen anotaciones detalladas de los cambios detectados.

Además de las modificaciones a los módulos existentes, el proyecto requirió la incorporación de nuevos componentes especializados para soportar la detección, análisis y documentación de cambios entre versiones. Estos nuevos módulos implementan la funcionalidad core del sistema de comparación y análisis:

Se introdujeron dos módulos de configuración fundamentales: longitud\_lineas.py que define LONGITUD\_MAXIMA\_LINEA (80 caracteres) siguiendo el estándar de codificación del proyecto, y umbral.py que establece UMBRAL\_SIMILITUD (0.6) para la detección de similitud entre líneas. Esta separación en módulos independientes facilita el ajuste y mantenimiento de parámetros críticos del sistema.

El modelo de cambios se implementó mediante el módulo cambios.py, introduciendo la enumeración TipoCambio para categorizar modificaciones (AGREGADA, BORRADA) y la clase Cambio que encapsula información detallada sobre cada modificación, incluyendo nodos involucrados, posición e impacto cuantitativo del cambio.

El ComparadorArboles implementa un algoritmo sofisticado de comparación mediante análisis recursivo de nodos, utilizando SequenceMatcher con umbral configurable. El sistema maneja casos especiales como líneas desplazadas, modificaciones parciales y bloques de código alterados, manteniendo diccionarios de mapeo para rastrear cambios en la estructura del árbol.

El ComparadorVersiones actúa como interfaz principal, coordinando la comparación entre versiones a través de comparar\_archivos y proporcionando estadísticas mediante contar\_cambios. Esta capa de abstracción simplifica el uso del sistema mientras mantiene la flexibilidad en el procesamiento y presentación de resultados.

La clase EscribirCambios procesa y documenta las modificaciones detectadas, generando comentarios descriptivos para cambios en el código. El sistema utiliza un formato estandarizado (ej: "# AÑADIDA EN UN 100%", "# BORRADA") y maneja casos especiales como cambios multilínea y modificaciones parciales, facilitando la comprensión de las alteraciones realizadas.

El FormateadorLinea implementa un sistema complejo para procesar construcciones Python, manejar líneas largas, preservar comentarios y mantener indentación. Se incorporó ExcepcionFormateo para manejar casos excepcionales, garantizando consistencia en la comparación y manteniendo la legibilidad del código durante el análisis de cambios.