BÍTACORA DE INGENIERÍA

Equipo 2

12 de mayo de 2025

Realizó:

Canul Ordoñez, Josué Israel

Garcilazo Cuevas, Mónica

Leo Fernández, José Carlos

Pool Flores, Endrick Alfredo

Rodríguez Coral, Samuel David

1. Introducción

La presente libreta de ingeniería tiene como objetivo documentar de manera estructurada las actividades de mantenimiento de software realizadas por el equipo. Este documento servirá como referencia para registrar las consideraciones, modificaciones y nuevas características efectuadas en el sistema, garantizando trazabilidad y continuidad en el desarrollo.

2. Alcance

Esta libreta está dirigida al equipo de mantenimiento de software y abarca todas las fases del proceso de mantenimiento, incluyendo notas, corrección de errores, mejoras, incorporación de nuevas funcionalidades y adaptaciones a los documentos.

3. Propósito

El propósito de esta libreta de ingeniería es proporcionar una herramienta organizada y centralizada para la documentación del mantenimiento de software. Permitiendo una mejor comunicación dentro del equipo y asegurará que cada modificación realizada sea correctamente documentada y entendida por los involucrados con el proyecto.

4. Estructura del Documento

Consideraciones: Listado de las cosas que se tuvieron que tomar en cuenta al momento de hacer el mantenimiento

Modificaciones: Modificaciones realizadas durante el mantenimiento a productos ya existentes.

Agregados: Nuevos productos que surgieron durante el mantenimiento.

5. Notas tomadas durante la realización del proyecto

5.1 Consideraciones

1. En el documento “Documento_especificacion_requerimientos_software”, se encontraron algunos errores de ortografía.

2. En el documento “Documento_arquitectura_de_software”, se encontraron algunos errores de ortografía.
3. En el documento “Documento_arquitectura_de_software”, se encontraron algunas inconsistencias con la redacción.
4. En el documento “Manual_de_usuario”, se encontraron algunos errores de ortografía.
5. En el documento “Manual_de_usuario”, se encontraron algunas inconsistencias con el formato.
6. En el documento “Manual_de_usuario”, se encontraron inconsistencias en el índice.
7. En los documentos en general, se encontraron inconsistencias de nombramiento, las cuales de estandarizaron posteriormente.
8. El código en general se encuentra muy acoplado con respecto a la funcionalidad de contabilización de líneas. Debido a ello solamente se pudo reutilizar las clases JavaFileScanner y la clase JavaFile para implementar la funcionalidad de comparación de dos proyectos.

5.2 Modificaciones

1. Para el documento “Documento_especificacion_requerimientos_software”, se modificó el alcance del proyecto para ajustarse a los nuevos requisitos del proyecto.
2. Para el documento “Documento_especificacion_requerimientos_software”, se corrigieron errores menores en el formato de este.
3. Para el documento “Documento_arquitectura_de_software”, se actualizó la arquitectura para agregar la funcionalidad de comparar versiones de un proyecto de software.
4. Para el documento “Documento_arquitectura_de_software”, se actualizaron los diagramas de casos de uso, de paquetes, de clases y de secuencia.
5. Para el documento “Manual_de_usuario”, se adaptó el contenido y la redacción del documento para hacer referencia a las nuevas características de la aplicación.
6. Para el documento “Documento_estimación_de_tamaño”, se actualizó el cálculo de puntos función considerando las nuevas funcionalidades implementadas en la tercera entrega, como la comparación de versiones, el etiquetado de líneas de código y la generación de reportes por clase. También se reclasificó el tamaño del proyecto como “pequeño”.
7. Para el documento “Estandar_de_conteo”, se añadieron las reglas para el análisis de diferencias entre versiones de archivos, el uso del algoritmo de

Levenshtein para detectar líneas modificadas, el etiquetado de cambios con [NEW], [DELETED], [MODIFIED], [ORIGINAL] y [SPLITED], así como la regla de formato para líneas mayores a 80 caracteres.

8. Para el documento "Documento_plan_administración_de_configuración", se incluyó el control de versiones aplicado a los documentos generados en la tercera entrega, así como el nombramiento de los documentos.
9. Se modifico la clase ProgramBuilder para manejar flujos de trabajo, cada uno con sus propios procesos. Debido a que las funcionalidades anteriores se encontraban muy acopladas, decidimos encapsular la funcionalidad de conteo y separarlos de la nueva funcionalidad de comparar.

4.3 Nuevas características

1. Para el documento "Documento_especificacion_requerimientos_software" se añadieron los nuevos requisitos para la comparación de dos versiones de un proyecto de software.
2. Para el documento "Documento_arquitectura_de_software", se agregó un diagrama de secuencia para representar la funcionalidad de comparar versiones de un proyecto de software.
3. Para el documento "Manual_de_usuario", se agregaron las instrucciones y consideraciones relevantes para abordar la funcionalidad de comparar versiones de un proyecto de software.
4. Lo siguiente fue implementado en el código fuente:
    a. Se creo la carpeta comparators, para las clases relacionadas con la comparacion de 2 proyectos
        i. Se implemento la clase JavaFileComparator, cuya funcionalidad es realizar una comparación del contenido de dos documentos java. Si los documentos tienen una longitud distinta, primero se comparar las líneas de código que comparten el mismo índice, luego se marcará el resto de las líneas del contenido más extenso como nuevo o eliminado, según sea el caso.
        ii. Se implemento la clase LineComparator, determina si dos líneas de código son similares mediante la ecuación de similaridad presentada en (*Principles of Data Integration*, 2012).
        iii. Se implemento la clase ProyectComparator, esta clase tiene la funcionalidad de comparar las clases que se encuentren en

ambos proyectos, de lo contrario se registrará la clase como eliminado o nuevo según sea el caso.

iv. Se implemento la clase Status, a fin de tener un marcador con los posibles estados que puede tener una línea de código: NEW, DELETED, ORIGINAL, MODIFIED, SPLITED. Dicha información es la requerida para armar el reporte de las clases.

b. Se implementaron las siguientes clases en la carpeta reporters
   i. ComparationReport, clase que prepara los reportes iniciales con la información contenida dentro de cada linea, en esta clase se agregan las etiquetas a las lineas segun el valor de la variable Status almacenadas dentro de esta linea, se hace uso de la clase LineRecord para poder acceder tanto al estatus de la linea como a su contenido.
   ii. TxtReporter, clase donde se arma los reportes en los cuales el usuario final podra revisar de manera detallada los cambios producidos entre 2 proyectos.

c. Se implemento la clase LineRecord en la carpeta models del codigo fuente como estructura que almacena informacion de una linea

d. En la carpeta utils del codigo fuente fueron implementadas las siguientes clases
   i. LineSplitter, clase de utilidad usada al momento de generar los reportes en la clase TxtReporter para asegurar que no hayan lineas de mas de 80 caracteres, el objetivo de la clase es unicamente dar mayor claridad visual al momento de leer los reportes, pues la division de lineas no influye en el conteo final de cambios
   ii. LevenshteinDistanceComputer, esta clase tiene la funcionalidad de calcular la distancia el costo mínimo de transformar una secuencia de caracteres a otra, por medio de las siguientes operaciones: eliminar un carácter, insertar un carácter o cambiar un carácter por otro (*Principles of Data Integration*, 2012).

5. Se agregaron los siguientes casos de prueba:
   a. Detectar líneas sin cambio entre versiones
   b. Detección de línea nueva
   c. Detección de archivo más corto (líneas eliminadas)
   d. Detectar línea eliminada en la versión nueva
   e. Detectar línea modificada entre versiones
   f. Manejo de líneas vacías
   g. Resumen global correcto
   h. Generar archivos con etiquetas por línea

i. Crear línea [ORIGINAL]
j. Crear línea [MODIFIED]
k. Crear línea [NEW]
l. Crear línea [DELETED]
m. Agregar múltiples líneas nuevas.
n. Dividir línea larga preservando contenido lógico.

Bibliografía

Principles of Data Integration. (2012). Elsevier. https://doi.org/10.1016/C2011-006130-6