Escuela Politécnica Nacional

Construcción y evolución de software

Proyecto 001.
Nombres: Sebastián Ramos, Mateo Quisilema
Nombre del proyecto: InkHarmony – Sistema de gestión de canciones.

Definición del proyecto.
InkHarmony es una aplicación de escritorio diseñada para gestionar canciones, artistas, géneros y álbumes dentro de una biblioteca musical digital.
El sistema permite registrar canciones con sus metadatos (artista, duración, género, año), adjuntar imágenes, editar información, eliminar registros y visualizar un catalogó completo ordenado y filtrado.
El objetivo del proyecto es ofrecer una interfaz intuitiva para administrar colecciones musicales, facilitando la clasificación, control y visualización del contenido. InkHarmony incluye módulos como el catálogo de canciones, administración de usuarios, catálogo de artistas y funciones CRUD conectadas a una base de datos SQLite.
Para garantizar su integridad, escalabilidad y facilidad de mantenimiento, se establece el siguiente plan maestro de SCM, que define reglas, procesos y herramientas para controlar versiones, cambios, artefactos, releases y mantenimiento del software.

Plan Maestro de SCM
Fase 1: Planificación y Diseño (ANTES)
Herramientas:
Se usará Git como sistema de control de versiones y GitHub como plataforma de gestión del repositorio.
Repositorio:
Se llamará *InkHarmony.* La rama principal se llamará main y estará protegida para evitar modificaciones directas mediante push, toda actualización deberá realizarse a través de Pull request.
Workflow:
Las ramas de trabajo se crearán siguiendo la convención:
- feature/[nombre-de-la-funcionalidad] para nuevas características.
- bugfix/[descripcion-del-error] para correcciones menores.
- hotfix/[descripcion-del-error-critico] para problemas urgentes en producción.

Política de Ramas:
Se adoptará una estrategia de feature branching. Cada funcionalidad deberá ser desarrollada en una rama independiente y posteriormente integrada a main mediante un Pull Request. Todos los PR deberán estar asociados a un Issue previamente creado, con el fin de mantener la trazabilidad del cambio.

Artefactos (No código):
Los requisitos funcionales y no funcionales se registrarán en documentos Markdown ubicados en la carpeta /docs del repositorio.
Los diseños de interfaz (mockups o prototipos) estarán alojados en Figma, y el enlace correspondiente se incluirá en el archivo README.md.
Documentación adicional como diagramas UML, manuales y especificaciones técnicas se almacenarán dentro de /docs/uml y otras subcarpetas temáticas según corresponda.

Fase 2: Codificación y Pruebas (DURANTE)
Gestión de Cambios:
Todo cambio deberá gestionarse mediante Pull Requests. Cada PR debe cumplir con una Definition of Done que incluye:
- Compilar correctamente.
- Pasar las pruebas automatizadas.
- Cumplir con las reglas de estilo del código.
- Incluir una referencia explícita al Issue correspondiente.
- Tener al menos una revisión y aprobación por parte de otro miembro del equipo.

Un PR que incumpla estos criterios no podrá ser fusionado.
Trazabilidad:
Cada commit deberá incluir una referencia al Issue de GitHub que motivó el cambio.
Integración Continua:
Se configurará GitHub Actions para automatizar procesos de verificación.
Se ejecutarán dos flujos principales:
1. Linter: Verifica el cumplimiento del estilo del código en cada push.
2. Tests: Ejecuta las pruebas unitarias en cada Pull Request hacia main.

Si alguno de estos procesos falla, la fusión del PR quedará bloqueada.
Gestión de Línea Base:
Se crearán tags para señalar hitos importantes del proyecto, tales como:
- v0.5-alpha al finalizar los módulos básicos de catálogo.
- v1.0-beta tras integrar el sistema de usuarios.
- v1.0.0 para la primera versión estable lista para despliegue.

Fase 3: Despliegue y Mantenimiento (DESPUÉS)

Gestión de Releases:

El proyecto adoptará un esquema de versionamiento semántico:

MAJOR.MINOR.PATCH

- Se incrementará MAJOR cuando existan cambios incompatibles.
- Se incrementará MINOR cuando se agreguen nuevas funcionalidades que no rompan compatibilidad.

- Se incrementará PATCH para correcciones de errores menores.

El despliegue hacia entornos de prueba se realizará automáticamente mediante GitHub Actions, mientras que el despliegue a producción se gestionará de manera manual mediante la creación de un release tag.

Mantenimiento Correctivo:

Cuando se identifique un error crítico, se abrirá un Issue y se creará una rama hotfix/.

Una vez corregido el problema, se integrará a main y se liberará una versión PATCH, por ejemplo: v1.0.1.

Mantenimiento Adaptativo:

Cada seis meses se revisarán las dependencias del proyecto, la compatibilidad del entorno y posibles cambios en librerías utilizadas. Se documentarán las actualizaciones necesarias y se ejecutarán en una rama de mantenimiento adaptativo.

Mantenimiento Perfectivo:

El equipo dedicará parte del ciclo de desarrollo (aproximadamente el 15% de cada sprint) a tareas de refactorización, optimización de consultas, simplificación de controladores y mejora de estructura del código sin alterar la funcionalidad existente.

Mantenimiento Preventivo:

Se habilitará una herramienta de monitoreo de dependencias (Dependabot) para detectar vulnerabilidades o versiones desactualizadas. Asimismo, se realizarán análisis estáticos del código y respaldos periódicos de la base de datos SQLite.