Escuela Politécnica Nacional

Facultad de Ingeniería de Sistemas

ISWD633 CONTRUCCIÓN Y EVOLUCIÓN DE SOFTWARE

PROYECTO DEL PRIMER BIMESTRE

PLAN MAESTRO DE SCM PARA CINEMAX

Integrantes: Javier Quilumba, Jonathan Tipán
Asignatura: Construcción y Evolución de Software
Curso: GR2SW
Periodo: 2025-B
Fecha de Entrega: 14 de diciembre de 2025
Docente: Vicente Eguez

1. Definición del Proyecto

1.1 Nombre del Proyecto

“CineMax”

1.2 Descripción del Proyecto

CineMax es una aplicación de escritorio destinada a la gestión integral de un sistema de cines. Esta aplicación permite administrar empleados, roles, ventas de boletos, reportes, facturación y más, todo desde una interfaz moderna y fácil de usar. El sistema está diseñado para ser escalable, de modo que pueda adaptarse al crecimiento de la operación del cine y a nuevas funcionalidades, como la integración con plataformas de streaming, analítica avanzada y gestión de promociones.

2. Plan Maestro de SCM

Este Plan Maestro de Gestión de Configuración de Software (SCM) establece las políticas, herramientas y procesos que se aplicarán en cada fase del Ciclo de Vida del Desarrollo de Software (SDLC) del proyecto **CineMax**. Asegurará que el software sea manejable, robusto y listo para escalar, optimizando la colaboración del equipo y la gestión de cambios.

2.1 Fase 1: ANTES de Escribir la Primera Línea de Código (Planificación y Diseño)

2.1.1 Configuración del Entorno y Repositorio

Herramientas:

- **Control de versiones:** Git
- **Repositorio:** GitHub
- **Gestión de incidencias:** GitHub Issues
- **Documentación:** Markdown y Wiki en el repositorio

Estructura del Repositorio:

- **Repositorio Principal:** cinemax-app
    - Código fuente

- Documentación
- Archivos de configuración del proyecto
- Carpeta /docs para la documentación adicional

Rama Principal:

- La rama principal se llamará **main**.
- **Protección de rama:** No se permitirá hacer push directamente a **main**. Todos los cambios deberán integrarse mediante Pull Requests (PR).
- **Ramas de trabajo:**
  - feature/nueva-funcionalidad: Para el desarrollo de nuevas funcionalidades.
  - bugfix/correccion-bug: Para correcciones de errores.
  - hotfix/corregir-error-critico: Para solucionar errores críticos en producción.

2.1.2 Definición del Flujo de Trabajo (Workflow)

Convenciones de Ramas:

- **feature/**: Para nuevas funcionalidades que están en desarrollo. Ejemplo: feature/gestionar-boletos.
- **bugfix/**: Para correcciones no críticas. Ejemplo: bugfix/correccion-error-login.
- **hotfix/**: Para correcciones urgentes en producción. Ejemplo: hotfix/arreglo-error-pago.

Política de Integración:

- Los cambios se deben realizar en ramas separadas de **main**.
- Todo cambio debe pasar por revisión mediante un Pull Request.
- La fusión solo será aprobada si el PR cumple con los siguientes requisitos:
  - Revisión de código por al menos un compañero.
  - Las pruebas automáticas pasadas (si las hay).
  - No hay errores de compilación.
  - El código sigue las convenciones de estilo establecidas.

2.1.3 Gestión de Artefactos (No-Código)

Requisitos:

- Los requisitos se documentarán en **GitHub Issues**, y se organizarán por prioridad en un backlog.

Diseños (UI/UX):

- Los mockups de la interfaz se crearán en **Figma**.
- Los enlaces de los diseños estarán documentados en el archivo **README.md**.

Documentación adicional:

- Los documentos importantes como la descripción general del sistema, alcance del proyecto y reglas de uso se guardarán en una carpeta /docs dentro del repositorio.

2.2 Fase 2: DURANTE el Desarrollo (Codificación y Pruebas)

2.2.1 Gestión de Cambios en Acción

Proceso del Pull Request (PR):

- **Definición de Hecho (Definition of Done)** para un PR:
  - El código ha sido revisado y aprobado por al menos un miembro del equipo.
  - No tiene errores de compilación ni de ejecución.
  - Pasa todas las pruebas unitarias.
  - Está documentado según los estándares del proyecto.

Trazabilidad:

- Cada funcionalidad o corrección deberá asociarse a un **Issue** de GitHub.
- Los mensajes de **commit** deben seguir el formato:
  git commit -m "Agrega funcionalidad de gestión de ventas (closes #12)".

2.2.2 Integración Continua (CI)

Configuración de CI:

- **GitHub Actions** o **GitLab CI** se configurarán para ejecutar:
  - **Linting:** Verificación del formato del código.
  - **Pruebas unitarias:** Ejecución automática de pruebas unitarias.
  - **Validación de código:** Asegurar que no se introduzcan errores en el código al hacer push a una rama de trabajo.

2.2.3 Gestión de Línea Base (Baselining)

Marcado de hitos importantes:

- Al completar módulos clave (como el módulo de ventas de boletos o la funcionalidad de reportes), se creará un **tag** de Git.
    - Ejemplo: v1.0-beta para indicar que la versión es estable y lista para pruebas.

2.3 Fase 3: DESPUÉS del Lanzamiento (Despliegue y Mantenimiento)

2.3.1 Gestión de Releases y Despliegue (CD)

Proceso de Despliegue:

- **Despliegue automático:** Cuando se fusiona a la rama **main**, GitHub Actions se encargará de construir y desplegar la versión en el entorno de pruebas.
- **Despliegue a producción:** Se realizará de forma manual mediante la creación de un tag de release.

Versionamiento:

- Se utilizará **Versionamiento Semántico (MAJOR.MINOR.PATCH)** para cada versión.
    - **PATCH:** Para correcciones de errores.
    - **MINOR:** Para nuevas funcionalidades compatibles.
    - **MAJOR:** Para cambios incompatibles que rompen la funcionalidad existente.

2.3.2 Plan de Mantenimiento Proactivo

Mantenimiento Correctivo:

- Los errores reportados se gestionarán a través de **GitHub Issues** con la etiqueta **bug**.
- En caso de un error crítico, se creará una rama **hotfix/** desde **main** y se fusionará a **main** después de la corrección.

Mantenimiento Adaptativo:

- Se revisarán las dependencias cada seis meses para asegurarse de que estén actualizadas y compatibles con nuevas versiones de las tecnologías.

Mantenimiento Perfectivo:

- Se asignará un 10% del tiempo de desarrollo en cada sprint para tareas de refactorización y mejora de rendimiento.

Mantenimiento Preventivo:

- Se utilizarán herramientas automáticas como **Dependabot** en GitHub para detectar vulnerabilidades en las dependencias y mantenerlas actualizadas.

Conclusión

Este Plan Maestro de SCM establece las mejores prácticas para el desarrollo y mantenimiento del proyecto **CineMax**, asegurando que el software sea robusto, mantenible y escalable desde el primer día. La implementación de estos procesos y herramientas ayudará a mantener el proyecto bajo control durante todo su ciclo de vida, garantizando su calidad y permitiendo su evolución y mejora continua.