Escuela Politécnica Nacional
Construcción de Software
Proyecto

Proyecto 001 – Plan Maestro de SCM

Asignatura: Construcción de Software
Periodo: 2025-B
Grupo: Jotcelyn Godoy, Cristian Robles

1. Definición del Proyecto

Nombre del Proyecto

“UniTask”

Descripción del Proyecto

UniTask es una aplicación web diseñada para ayudar a estudiantes universitarios a gestionar sus tareas académicas de forma organizada. La aplicación permite registrar materias, crear tareas con fechas límite, asignar prioridades y marcar el progreso de cada actividad. Su objetivo principal es mejorar la planificación del tiempo y reducir el riesgo de atrasos en entregas académicas.

El proyecto está pensado como una solución escalable, con la posibilidad de incorporar futuras funcionalidades como notificaciones, recordatorios automáticos y sincronización con calendarios externos.

2. Plan Maestro de SCM

El presente Plan Maestro de Gestión de Configuración de Software (SCM) define las reglas, procesos y herramientas que se aplicarán a lo largo de todo el Ciclo de Vida del Desarrollo de Software (SDLC), con el fin de garantizar un proyecto mantenible, controlado y preparado para el crecimiento futuro.

Fase 1: ANTES de Escribir la Primera Línea de Código

(Planificación y Diseño)

2.1 Configuración del Entorno y Repositorio

Herramientas

- Sistema de control de versiones: **Git**
- Plataforma de alojamiento del repositorio: **GitHub**
- Gestión de tareas e incidencias: **GitHub Issues**
- Documentación: **Markdown dentro del repositorio**

Estructura del Repositorio

- Se utilizará un **repositorio único (monorepo)** que contendrá:
  - Código fuente
  - Documentación
  - Configuraciones del proyecto
- Nombre del repositorio: unitask-app

Rama Principal

- La rama principal se llamará **main**
- La rama main estará **protegida**, por lo que:
  - No se permitirán pushes directos
  - Todo cambio deberá integrarse mediante Pull Requests (PR)

2.2 Definición del Flujo de Trabajo (Workflow)

Se utilizará una variante de **GitHub Flow**, adecuada para proyectos pequeños y medianos.

Convención de Ramas

- feature/nombre-funcionalidad → desarrollo de nuevas funcionalidades
- bugfix/descripcion-bug → corrección de errores no críticos
- hotfix/descripcion-urgente → corrección de errores críticos en producción

Política de Integración

- Todo cambio debe realizarse en una rama distinta a main
- La integración se hará exclusivamente mediante **Pull Requests**
- Un PR solo podrá fusionarse si cumple con los criterios definidos en la Fase 2

2.3 Gestión de Artefactos (No-Código)

Requisitos

- Los requisitos funcionales se documentarán como **Historias de Usuario** en GitHub Issues
- Se mantendrá una carpeta /docs con:

- Descripción general del sistema
- Alcance del proyecto
- Reglas básicas de uso

Diseños (UI/UX)

- Los mockups de la interfaz se crearán en **Figma**
- El enlace al diseño estará documentado en el archivo README.md

Fase 2: DURANTE el Desarrollo

(Codificación y Pruebas)

3.1 Gestión de Cambios en Acción

Proceso del Pull Request (PR)
Un Pull Request será aprobado únicamente si cumple con la siguiente **Definición de Hecho (Definition of Done):**

- El código fue revisado y aprobado por al menos **un integrante del equipo**
- No presenta errores de compilación
- Cumple con las normas básicas de estilo del proyecto
- Está correctamente documentado cuando sea necesario

Trazabilidad

- Cada funcionalidad o corrección deberá estar asociada a un Issue
- Los mensajes de commit seguirán esta estructura:
- Descripción corta del cambio (closes #ID)

Ejemplo:

Agrega creación de tareas por materia (closes #7)

3.2 Integración Continua (CI)

Para asegurar la calidad del código, se configurará **GitHub Actions** con los siguientes procesos automáticos:

- Ejecución de pruebas básicas en cada PR
- Verificación de errores comunes en el código

- Validación automática antes de permitir la fusión a main

Esto permitirá detectar errores tempranamente y evitar que código defectuoso llegue a la rama principal.

3.3 Gestión de Línea Base (Baselining)

- Los hitos importantes del proyecto se marcarán mediante **tags de Git**
- Ejemplos de líneas base:
    - v0.1-alpha → funcionalidades básicas implementadas
    - v1.0-beta → sistema funcional listo para pruebas
- Cada línea base representará un estado estable del sistema

Fase 3: DESPUÉS del Lanzamiento

(Despliegue y Mantenimiento)

4.1 Gestión de Releases y Despliegue (CD)

Proceso de Despliegue

- El despliegue al entorno de pruebas será automático tras la fusión a main
- El despliegue a producción será **manual**, mediante la creación de un tag de release

Versionamiento

- Se utilizará **Versionamiento Semántico (MAJOR.MINOR.PATCH)**
  Ejemplo: v1.2.3

Criterios de Incremento

- **PATCH**: corrección de errores
- **MINOR**: nuevas funcionalidades compatibles
- **MAJOR**: cambios que rompen compatibilidad

4.2 Plan de Mantenimiento Proactivo

Mantenimiento Correctivo

- Los errores se reportarán mediante GitHub Issues con la etiqueta bug

- Para errores críticos:
    - Se creará una rama hotfix/
    - Se fusionará a main
    - Se liberará una nueva versión PATCH (ej. v1.0.1)

Mantenimiento Adaptativo

- Cada semestre se revisará:
    - Compatibilidad con navegadores actuales
    - Cambios en librerías o frameworks utilizados

Mantenimiento Perfectivo

- Se destinará tiempo periódico para:
    - Refactorización de código
    - Mejora de rendimiento
    - Optimización de estructura interna

Mantenimiento Preventivo

- Se utilizarán herramientas automáticas (como Dependabot) para:
    - Detectar vulnerabilidades
    - Actualizar dependencias inseguras
    - Reducir riesgos futuros

Conclusión

Este Plan Maestro de SCM establece una base sólida para el desarrollo del proyecto UniTask, asegurando control, calidad y trazabilidad desde la planificación hasta el mantenimiento. La aplicación de estas prácticas permitirá que el software sea sostenible en el tiempo, fácil de mantener y preparado para futuras ampliaciones, cumpliendo con los principios fundamentales de una buena gestión de configuración de software.