Proyecto: **Plan Maestro de SCM — Aplicativo Tienda de Ropa (Java)**

Integrantes: Erick Alpusig, Claudio Peñaherrrera, Saúl Tualombo

Curso: GR2SW

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

Configuración del Entorno y Repositorio

- **Herramientas principales:**
  Git (control de versiones), GitHub (repositorio remoto), IntelliJ IDEA (IDE principal), Maven (gestor de dependencias), y Figma (diseño de interfaces).
- **Estructura del Repositorio:**
  El repositorio se llamará **"tienda-ropa-java"** y será **monorepo**, incluyendo módulos de backend, frontend y documentación.
- tienda-ropa-java/
- ├── src/
- ├── docs/
- ├── tests/
- ├── .github/
- └── README.md
- **Política de Ramas (Branching Strategy):**
  - Rama principal: **main** (protegida; no se puede hacer push directo).
  - Otras ramas:
    - develop: integración de nuevas características.
    - feature/*: desarrollo de funcionalidades nuevas.
    - bugfix/*: correcciones menores.
    - release/*: versiones candidatas.
    - hotfix/*: correcciones urgentes en producción.

Definición del Flujo de Trabajo (Workflow)

- **Creación de ramas:** cada nueva funcionalidad se desarrolla en una rama feature/nombre-feature.
- **Integración de cambios:** todo cambio se hace mediante **Pull Request (PR)** hacia develop y requiere:
    - Aprobación de al menos **1 revisión de compañero** (peer review).
    - Éxito en las **pruebas automáticas (CI)** antes de fusionar.

Gestión de Artefactos (No-Código)

- **Requisitos:** almacenados en /docs/requisitos.md.
- **Diseños UI/UX:** enlace de Figma incluido en /docs/diseño_ui.md.
- **Manual de instalación y despliegue:** /docs/manual_instalacion.md.
- **Diagramas UML:** almacenados en /docs/uml/.

Fase 2: DURANTE el Desarrollo (Codificación y Pruebas)

Gestión de Cambios en Acción

- **Proceso de Pull Request (PR):**
    - Todo PR debe incluir:
        - Descripción breve del cambio.
        - ID del issue relacionado (ejemplo: closes #15).
        - Evidencia de pruebas exitosas.
    - **Definición de Hecho (DoD):**
        - Código revisado.
        - Pruebas unitarias exitosas.
        - Cumplimiento del estándar de codificación.
        - Documentación actualizada.
- **Trazabilidad:**

- Cada commit debe tener un mensaje descriptivo, por ejemplo:
  git commit -m "Agrega lógica de login (closes #12)".

Integración Continua (CI)

- **Automatización:**
  Se configurará **GitHub Actions** para ejecutar:
  - mvn test (pruebas unitarias)
  - mvn verify (validación de calidad)
  - checkstyle (formato del código)
- Si alguna falla, el PR no se podrá fusionar.

Gestión de Línea Base (Baselining)

- **Creación de tags Git:**
  Se generará un tag por cada hito clave.
  Ejemplo:
  - v1.0-beta: módulo de registro de usuarios terminado.
  - v1.0.0: primera versión estable en producción.

Fase 3: DESPUÉS del Lanzamiento (Despliegue y Mantenimiento)

Gestión de Releases y Despliegue (CD)

- **Despliegue:** automático desde GitHub Actions hacia servidor remoto (Apache Tomcat).
- **Versionamiento:**
  Se usará **Versionamiento Semántico (SemVer)**:
  - MAJOR → cambios incompatibles.
  - MINOR → nuevas funcionalidades sin romper compatibilidad.
  - PATCH → corrección de errores.
    Ejemplo: v1.2.3.

Plan de Mantenimiento Proactivo (Los 4 Tipos)

Tipo de Mantenimiento	Descripción	Ejemplo / Política
Correctivo	Solución inmediata a fallos críticos.	Rama hotfix/bug-login desde main, merge tras revisión. Nueva etiqueta v1.0.1.
Adaptativo	Actualizar librerías o compatibilidades.	Cada semestre, revisar dependencias Maven y APIs de terceros.
Perfectivo	Optimizar y refactorizar código existente.	Último viernes del mes: sprint de refactorización.
Preventivo	Evitar vulnerabilidades futuras.	Activar **Dependabot** y análisis de seguridad en GitHub.