Gestión Desarrollo de Software – 2025

Prof. Politi Raul

Clase 21 (Unidad 7)

Gestion Codigo Fuente

Introducción

La **organización y gestión del código fuente** constituye una de las actividades más importantes dentro del ciclo de vida del desarrollo de software.
Consiste en **estructurar, controlar y mantener de forma ordenada todas las versiones del código** que conforman un proyecto.
Su objetivo principal es **garantizar la trazabilidad, integridad y colaboración eficiente** entre los distintos miembros del equipo de desarrollo.

El código fuente es el **activo más valioso** de un proyecto de software, ya que representa el resultado del conocimiento, la lógica y las decisiones tomadas durante el desarrollo. Por eso, una correcta gestión del mismo permite evitar errores, pérdidas de trabajo y conflictos de versiones.

Importancia de la gestión del código fuente

Una buena gestión del código permite:

- Asegurar la **integridad y seguridad** del software.
- Controlar **qué cambios se realizaron, cuándo y por quién**.
- Permitir el trabajo colaborativo de varios desarrolladores sin pisar o perder modificaciones.
- Facilitar la **reversión a versiones anteriores** en caso de errores.
- Mantener una **historia completa del proyecto**.
- Integrar procesos automatizados de **pruebas, despliegue y control de calidad**.

Sin una gestión adecuada, los proyectos pueden sufrir pérdidas de código, conflictos de versiones, inconsistencias y dificultades para mantener el producto.

Control de versiones

El **control de versiones** (Version Control System – VCS) es el conjunto de herramientas y prácticas que permiten **registrar, rastrear y administrar los cambios en los archivos de un proyecto** a lo largo del tiempo.

Tipos de sistemas de control de versiones

1. **Centralizados (CVCS):**
   Todo el código se guarda en un servidor central, y los desarrolladores trabajan sobre copias locales que se sincronizan con ese servidor.
   Ejemplo: *Subversion (SVN), Perforce, CVS.*

2. **Distribuidos (DVCS):**
   Cada desarrollador tiene una copia completa del repositorio, incluyendo el historial completo del proyecto. Esto permite trabajar offline y facilita las fusiones.
   Ejemplo: *Git, Mercurial.*

Actualmente, **Git** es el estándar de facto en la industria.

Principales conceptos en Git

- **Repositorio:** Carpeta que contiene todo el código y su historial de cambios.
- **Commit:** Registro de un conjunto de modificaciones.
- **Branch (rama):** Línea de desarrollo independiente.
- **Merge:** Proceso de combinar cambios de una rama en otra.
- **Clone:** Copia local de un repositorio remoto.
- **Pull / Push:** Acciones para obtener o enviar cambios entre el repositorio local y remoto.

Estructura y organización del código

La **organización del código fuente** implica definir una estructura clara y coherente dentro del proyecto.
Un código bien organizado facilita su comprensión, mantenimiento y escalabilidad.

Buenas prácticas de organización

- Separar el código por **módulos o capas** (por ejemplo: presentación, lógica de negocio, datos).
- Usar **nombres descriptivos** para archivos, carpetas y funciones.
- Mantener una **estructura coherente** entre proyectos del mismo equipo.
- Incluir un **archivo README.md** con instrucciones de uso e instalación.
- Incorporar un **archivo .gitignore** para excluir archivos innecesarios (como dependencias o archivos temporales).
- Utilizar **comentarios claros y concisos** dentro del código.
- Aplicar **estándares de estilo y formato**, ayudándose con linters o herramientas como *Prettier* o *ESLint*.

Gestión colaborativa del código

En equipos de desarrollo, varios programadores pueden trabajar en distintas partes del mismo proyecto.
Para evitar conflictos y garantizar una integración fluida, se aplican estrategias como:

Flujo de trabajo con ramas (branching model)

Una práctica común es el uso del modelo **Git Flow**, que propone la creación de ramas específicas para cada tarea o entorno:

- `main` o `master`: contiene la versión estable del producto.
- `develop`: rama principal de desarrollo.
- `feature/*`: ramas temporales para nuevas funcionalidades.
- `release/*`: ramas de preparación para una nueva versión.
- `hotfix/*`: ramas destinadas a corregir errores urgentes.

Esto permite trabajar en paralelo, sin interferir con el código en producción.

Revisión y control de calidad

Antes de integrar nuevos cambios al repositorio principal, es recomendable que otro miembro del equipo revise el código mediante una **Pull Request (PR)** o **Merge Request**

(MR).
Este proceso permite:

- Detectar errores antes de que lleguen a producción.
- Mejorar la calidad y consistencia del código.
- Compartir conocimientos entre los desarrolladores.
- Garantizar que se cumplan las convenciones del proyecto.

Integración continua (CI) y entrega continua (CD) CD/CI

Estos conceptos son una extensión natural de la gestión del código fuente moderna.

Integración continua (CI):

Consiste en **automatizar la compilación y prueba del código** cada vez que se realizan cambios.
Permite detectar errores de forma temprana y mantener el sistema estable.

Entrega continua (CD):

Es la automatización del proceso de **despliegue del software** hacia los entornos de prueba o producción.
Ambos procesos se implementan usando herramientas como:

- **GitHub Actions**
- **GitLab CI/CD**
- **Jenkins**
- **CircleCI**

De esta forma, la gestión del código no solo abarca el almacenamiento, sino también su validación y despliegue continuo.

Plataformas y repositorios más utilizados

- **GitHub:** La plataforma más popular para proyectos open source y empresariales.
- **GitLab:** Permite repositorios privados y herramientas CI/CD integradas.
- **Bitbucket:** Muy usado en entornos corporativos; ofrece integración con Jira.

- **Azure DevOps:** Ideal para equipos que trabajan con tecnologías Microsoft.

Cada una de estas plataformas proporciona funcionalidades para:

- Gestionar ramas y versiones.
- Crear issues y tableros Kanban.
- Documentar con wikis integradas.
- Controlar accesos y permisos de usuarios.

Seguridad en la gestión del código

Algunas recomendaciones para proteger el código fuente:

- No subir **claves, contraseñas ni tokens** al repositorio.
- Utilizar **archivos de configuración fuera del código principal**.
- Aplicar **autenticación en dos pasos (2FA)** en plataformas remotas.
- Limitar los **permisos de escritura** en ramas principales.
- Usar **firmas digitales en los commits** para verificar la autoría.

Ventajas y desventajas de una buena gestión del código

✅ Ventajas

- Facilita el trabajo colaborativo y simultáneo.
- Permite volver a versiones anteriores en segundos.
- Mejora la trazabilidad y control de cambios.
- Favorece la calidad del producto y la detección temprana de errores.
- Automatiza procesos de compilación, prueba y despliegue.

⚠ Desventajas o desafíos

- Requiere **capacitación y disciplina** por parte del equipo.
- Pueden ocurrir conflictos de fusión (*merge conflicts*) si no se coordina bien.
- Configurar entornos CI/CD puede requerir tiempo inicial.
- Algunos equipos pequeños pueden considerarlo excesivo si no lo necesitan plenamente.

Buenas prácticas generales

1. Hacer **commits pequeños y frecuentes**, con mensajes descriptivos.
2. Evitar subir archivos binarios o innecesarios.
3. Proteger las ramas principales (`main`, `master`, `release`).
4. Revisar el código antes de fusionar cambios.
5. Etiquetar las versiones estables con **tags**.
6. Documentar el flujo de trabajo en el README del repositorio.
7. Mantener la coherencia en la estructura de carpetas y nombres.

Conclusión

La organización y gestión del código fuente no es solo una práctica técnica, sino una **estrategia de calidad y eficiencia**.
Un proyecto con código bien estructurado, versionado y documentado es más fácil de mantener, escalar y auditar.
Además, fortalece el trabajo en equipo, evita pérdidas de tiempo y reduce los errores en entornos colaborativos.

En la actualidad, dominar herramientas como **Git y plataformas como GitHub o GitLab** es una **competencia esencial para cualquier desarrollador profesional**, ya que constituyen el núcleo del desarrollo moderno de software.