UNIVERSIDAD DISTRITAL FRANCISCO JOSÉ DE CALDAS

FACULTAD DE INGENIERÍA

SYLLABUS

Espacio Académico:	Ingeniería de Software Colaborativa: Git y CI/CD
Tipo de Curso:	Teórico ☐ Práctico ☐ Teórico-Práctico ☒
Alternativas Metodológicas:	Clase Magistral ☒ Seminario ☐ Seminario-Taller ☒ Taller ☒ Prácticas ☒ Proyecto ☐
Conocimientos Previos:	Se requiere al menos conocimientos básicos de programación en cualquier lenguaje y una comprensión fundamental del trabajo colaborativo.

I. JUSTIFICACIÓN DEL ESPACIO ACADÉMICO

En el plan de estudios actual de Ingeniería de Sistemas, los estudiantes adquieren sólidas bases de programación y diseño de software. Sin embargo, la industria tecnológica exige competencias adicionales que frecuentemente quedan fuera de las asignaturas tradicionales: **el trabajo colaborativo asíncrono, la gestión rigurosa de versiones de código y la automatización de procesos de desarrollo (DevOps)**.

Este curso de extensión busca cerrar la brecha entre la "programación académica" (caracterizada por proyectos individuales o compartidos mediante archivos comprimidos) y la "ingeniería de software profesional", estandarizando el uso de **Git y GitHub/GitLab** como herramientas fundamentales para el éxito en materias avanzadas y en el ejercicio profesional.

A través de una metodología basada en proyectos reales, los estudiantes experimentarán simulaciones de entornos laborales, adoptando roles técnicos rotativos y aplicando flujos de trabajo colaborativos estándar de la industria. El curso es **agnóstico del lenguaje de programación**: aunque se sugiere inicialmente **Java + Maven** (por facilidad de integración con workflows de CI/CD), los estudiantes pueden aplicar Git y CI/CD a cualquier stack tecnológico (Python, JavaScript, C++, etc.), garantizando compatibilidad multiplataforma (Linux, Windows y macOS).

II. PROGRAMACIÓN DEL CONTENIDO

OBJETIVO GENERAL

Capacitar al estudiante en herramientas profesionales de ingeniería de software colaborativa, desarrollando competencias para gestionar proyectos con Git, automatizar procesos de construcción y pruebas (CI/CD), y trabajar eficientemente en equipos de desarrollo siguiendo estándares de la industria, aplicables a cualquier stack tecnológico.

OBJETIVOS ESPECÍFICOS

1. Comprender la arquitectura distribuida de Git y su diferencia con sistemas centralizados.
2. Configurar entornos de desarrollo profesional multiplataforma (Linux, Windows, macOS) con Git y herramientas de construcción del lenguaje elegido.
3. Dominar el ciclo de vida del código: estados, staging, commits semánticos y gestión del historial.
4. Implementar estrategias de branching (Git Flow) y resolver conflictos de integración de manera efectiva.
5. Utilizar plataformas de hosting (GitHub/GitLab) para colaboración remota, code review y pull requests.
6. Aplicar roles rotativos en equipos (Tech Lead, Developer) para simular dinámicas laborales reales.
7. Automatizar procesos de construcción y pruebas mediante pipelines de CI/CD (GitHub Actions, GitLab CI).
8. Validar competencias mediante la entrega de un repositorio auditado con evidencia de trabajo colaborativo.

COMPETENCIAS DE FORMACIÓN

Competencias Técnicas:

Configuración y uso de herramientas de desarrollo profesional (Git, Maven, IDEs), gestión de repositorios distribuidos, automatización de procesos de construcción y despliegue mediante pipelines CI/CD, y depuración de conflictos de integración.

Competencias Colaborativas:

Trabajo en equipos multidisciplinarios con roles definidos, comunicación técnica efectiva mediante issues y pull requests, revisión de código constructiva (code review), y adaptación a flujos de trabajo ágiles.

Competencias Profesionales:

Esta formación contribuye al desarrollo de competencias en gestión de proyectos de software, aplicación de estándares de la industria (Conventional Commits, Semantic Versioning), uso de metodologías DevOps y preparación para entornos laborales tecnológicos modernos. El curso se enmarca en el dominio de "herramientas de ingeniería de software" del área de formación complementaria del proyecto curricular de Ingeniería de Sistemas.

PROGRAMA SINTÉTICO

Módulo 1: Fundamentos y Estandarización (Semanas 1–3)

1. Configuración del entorno de ingeniería: Git, entorno de desarrollo del lenguaje elegido (sugerencia inicial: Java + Maven), variables de entorno.
2. Arquitectura de Git: Local, Staging, Remote. Estados del archivo.
3. Ciclo de vida del código: Commits semánticos (Conventional Commits), `.gitignore` para el stack tecnológico usado.
4. Gestión del historial: `log`, `diff`, `restore`, `tags` para versionamiento.

Módulo 2: Ramas y Estrategias de Colaboración (Semanas 4–6)

5. Branching: Concepto de punteros, creación y gestión de ramas (`feature/*`, `develop`, `main`).
6. Merging y resolución de conflictos: Fast-Forward vs Recursive, laboratorio de fusión.
7. Trabajo remoto: Autenticación SSH, `origin` y `upstream`, clonación y sincronización.
8. Git Flow: Flujo de trabajo profesional con ramas `develop`, `release` y `hotfix`.

Módulo 3: Calidad y Gestión de Proyectos (Semanas 7–8)

9. Code Review: Pull Requests (PR), comentarios constructivos, aprobaciones de cambios.
10. Project Management: Issues, Labels, Milestones, tableros Kanban (GitHub/GitLab Projects).
11. Vinculación de commits con issues: Cierre automático mediante `Fixes #N`, trazabilidad.

Módulo 4: Automatización e Integración Continua (Semanas 9–10)

12. Introducción a CI/CD: Concepto de pipeline, estructura YAML (`.github/workflows`, `.gitlab-ci.yml`).
13. Automatización de pruebas: Pipeline que ejecuta comandos de testing del stack elegido (ej. `mvn test`, `pytest`, `npm test`), bloqueo de PR si fallan tests.
14. Continuous Deployment: Generación de artifacts (ejecutables, paquetes), publicación en Releases mediante tags.

III. ESTRATEGIAS

Impartición y rol del GLUD:

- El curso será impartido por el Grupo de Trabajo GLUD como complemento a la formación curricular, aportando experiencia práctica, acompañamiento y recursos para la inserción en flujos colaborativos reales.

Metodología: Aprendizaje Basado en Proyectos (ABP):

- El curso abandona el formato de clase magistral para adoptar una **simulación de entorno laboral**.
- Los estudiantes trabajan en células de 3 personas (**tríos**) con roles rotativos semanales: *Tech Lead* (revisa código y aprueba cambios) y *Developers* (implementan features).
- **Stack tecnológico flexible:** Git (consola), GitHub/GitLab + lenguaje de programación a elección del equipo. *Sugerencia inicial: Java (JDK 17+) + Apache Maven* (facilita integración con workflows CI/CD mediante `mvn test`/`mvn package`).

Sesiones teórico-prácticas:

- Cada sesión combina **exposición teórica breve** (15–20 min) + **ejecución en terminal** (60–70 min).
- Los estudiantes replican comandos en tiempo real, resuelven ejercicios y trabajan en el proyecto colaborativo.
- Se fomenta la resolución inmediata de dudas mediante sesiones interactivas y checkpoints con el GLUD.
- Para sesiones de alta complejidad (ej. Laboratorio de Caos), se contará con **monitores del GLUD** para soporte técnico individualizado.

Proyecto final iterativo:

- Desarrollo incremental en entregas parciales: repositorio inicial (Semana 3), demo intermedia (Semana 7), entrega final auditada (Semana 10).
- Uso de flujos colaborativos reales: `feature branches`, `pull requests`, `code review`, pipelines CI/CD.
- El GLUD facilita **plantillas base** para múltiples lenguajes: repositorio inicial con estructura de proyecto, `.gitignore` apropiado, plantilla de pipeline CI/CD y ejemplos de buenas prácticas. *Stack recomendado para iniciar: Java + Maven*.

IV. RECURSOS BIBLIOGRÁFICOS

Recursos en línea:

- Documentación oficial de Git: `https://git-scm.com/doc`
- GitHub Learning Lab: `https://lab.github.com/`
- Atlassian Git Tutorials: `https://www.atlassian.com/git/tutorials`
- GitLab Documentation: `https://docs.gitlab.com/`
- Interactive Git Branching: `https://learngitbranching.js.org/`

V. ORGANIZACIÓN / TIEMPOS

Espacios, Tiempos, Agrupamientos:

El curso se organiza en **10 semanas (40 horas aproximadamente)** con sesiones teórico-prácticas. Los estudiantes trabajarán en células de 3 personas con roles rotativos semanales: *Tech Lead* (revisa código y aprueba cambios) y *Developers*. Se utilizarán GitHub/GitLab para colaboración asíncrona y Git Bash (Windows) o Terminal (Linux/macOS) para ejecución de comandos.

No.	Tema a desarrollar	Semanas académicas
1	**Configuración del Entorno**	Instalación de Git y entorno del lenguaje elegido (sugerencia: Java JDK 17+ + Maven). Variables de entorno. Arquitectura de Git (Local, Staging, Remote). Primer `git init`.
2	**Ciclo de Vida del Código**	Estados del archivo (Untracked, Staged, Committed). Conventional Commits. Estructura de proyecto del stack elegido. Configuración de `.gitignore` apropiado (`gitignore.io`).
3	**Viajes en el Tiempo**	Historial (`log`), diferencias (`diff`), restauración (`checkout/restore`). Etiquetas (`tags`) para versiones. Crear versión `v0.1.0-alpha`.
4	**El Multiverso (Branching)**	Concepto de punteros y referencias. Ramas `feature/login` y `feature/inventario`. Por qué nunca trabajar directo en `main`.

No.	Tema a desarrollar	Semanas académicas
5	**Fusión y Conflictos**	Merge (Fast-Forward vs Recursive). Rebase (concepto básico). **Laboratorio de Caos**: resolución guiada de conflictos en archivos del proyecto con apoyo de monitores GLUD.
6	**Flujos Remotos (GitHub/GitLab)**	Autenticación SSH (clave pública/privada). Conceptos de `origin` y `upstream`. Clonación de repositorios. Introducción a Git Flow (`develop`, `release`, `main`).
7	**Code Review y Pull Requests**	El arte de revisar código ajeno. Todo cambio requiere un PR aprobado. Comentarios constructivos. Rol de Maintainer rotativo.
8	**Project Management Integrado**	Issues, Labels, Milestones, tableros Kanban. Reportar bugs, crear Issues, vincular con commits (`Fixes #12`).
9	**Introducción a CI (Integración Continua)**	¿Qué es un Pipeline? Estructura YAML (`.gitlab-ci.yml`, `.github/workflows`). El GLUD proveerá *plantillas base* para distintos lenguajes. Configurar y modificar pipeline para ejecutar tests del stack (ej. `mvn test`, `pytest`, `npm test`). Bloqueo de PR si fallan tests.
10	**Entrega Final y Despliegue**	Continuous Deployment (CD) básico. Pipeline que genera artifacts (ejecutables, paquetes) y publica en Releases al detectar tag `v1.0`. **Proyecto Final**: entrega de repositorio auditado.

VI. EVALUACIÓN Y CERTIFICACIÓN

Modelo de Evaluación: El curso **no contempla exámenes teóricos tradicionales**. La certificación se otorga bajo el modelo de **Validación de Competencias por Evidencia**, simulando las prácticas de evaluación en entornos profesionales de desarrollo de software.

Evidencia Principal: Repositorio Auditado
Cada grupo debe entregar la URL de su repositorio público en GitHub/GitLab al finalizar la semana 10. El repositorio será auditado por el equipo GLUD utilizando los siguientes criterios:

Criterio de Aprobación	Descripción	Peso
Historial de commits	Commits semánticos (Conventional Commits), mensajes claros, atomicidad.	25 %
Pull Requests documentados	Al menos 5 PRs con descripciones, revisiones de código y aprobaciones del Tech Lead.	20 %
Pipeline CI/CD funcional	Pipeline configurado y pasando en verde (Build Success). Debe ejecutar tests del stack automáticamente.	20 %
Proyecto compilable/ejecutable	El proyecto debe compilar/ejecutar correctamente con las herramientas del stack elegido (ej. `mvn package`, `python setup.py`, `npm build`) sin errores.	15 %
Issues y gestión de proyecto	Al menos 8 Issues creados, etiquetados y cerrados mediante commits vinculados (`Fixes #N`).	10 %
Rotación de roles completada	Evidencia de que cada estudiante actuó como Tech Lead al menos una vez durante el proyecto (verificable mediante PRs aprobados).	10 %
TOTAL		**100 %**

Criterio de Aprobación: Se requiere un mínimo de **70/100 puntos** para obtener el certificado.

COMPETENCIAS DESARROLLADAS

- Configuración y uso profesional de Git y herramientas de construcción/testing del stack elegido en entornos multiplataforma.
- Dominio de flujos de trabajo colaborativos: branching, merging, resolución de conflictos.
- Aplicación de Conventional Commits y Semantic Versioning.
- Implementación de pipelines de CI/CD para automatización de pruebas y despliegues.
- Gestión de proyectos mediante Issues, Labels, Milestones y tableros Kanban.
- Trabajo en equipos con roles técnicos definidos (Tech Lead, Developer).
- Revisión de código (code review) y aprobación de cambios mediante Pull Requests.
- Uso de plataformas de hosting (GitHub/GitLab) para colaboración remota.
- Integración de Git en flujos de trabajo ágiles y DevOps.
- Capacidad de entregar proyectos de software con evidencia auditable de calidad.