PLANIFICACIÓN ANUAL DE CÁTEDRA

Ciclo Lectivo 2026

Carrera	**Tecnicatura Superior en Desarrollo de Software**
Unidad Curricular	Ingeniería de Software II
Curso	Segundo Año
Régimen	Anual
Carga horaria semanal	4 horas cátedra
Formato curricular	Materia con Práctica Formativa (mínimo 33 %)
Docente	Muselli Gonzalo Damian

1. Fundamentación

Ingeniería de Software II profundiza el recorrido iniciado en primer año, desplazando el foco desde la construcción de programas hacia el diseño de artefactos de software y su verificación y validación sistemática. En el ejercicio profesional, el técnico en desarrollo de software no solo escribe código: diseña estructuras de datos e interfaces, evalúa alternativas según criterios de eficiencia, mantenibilidad y escalabilidad, y garantiza la calidad del producto mediante procesos de prueba planificados y documentados.

La unidad curricular articula dos grandes campos: el diseño de software (principios generales, patrones, arquitecturas, diseño orientado a objetos y de interfaces) y la verificación y validación (fundamentos de testing, pruebas funcionales y estructurales, desarrollo conducido por pruebas, refactorización e inspecciones). Ambos campos se integran mediante una práctica formativa intensa, con proyectos grupales que simulan condiciones del trabajo profesional real: división de roles, integración de componentes producidos por distintos equipos, registro y seguimiento de fallas, y elaboración de informes técnicos.

2. Finalidad formativa

El desarrollo de esta unidad curricular tiene como finalidad afianzar en los estudiantes las siguientes capacidades:

- Diseñar artefactos de software que resuelvan problemas planteados.
- Analizar críticamente la eficiencia y mantenibilidad de diseños alternativos.
- Diseñar las estructuras de datos y las interfaces que las mismas utilicen, analizando y discutiendo su eficiencia y escalabilidad.
- A partir del diseño, analizar clases de equivalencia y diseñar esquemas de prueba.
- Documentar el diseño de manera clara, completa y comunicable.

3. Objetivos de aprendizaje

Al finalizar el cursado, se espera que el estudiante sea capaz de:

- Aplicar principios de diseño (descomposición, desacoplamiento, cohesión, reusabilidad, portabilidad, testeabilidad, flexibilidad y escalabilidad) en el modelado de soluciones.
- Reconocer situaciones-problema y relacionarlas con patrones de diseño adecuados.

- Comparar arquitecturas de software (distribuidas, pipe-and-filter, model-view-controller) discutiendo sus propiedades de calidad.
- Modelar sistemas con diseño orientado a objetos: encapsulamiento, herencia, polimorfismo, jerarquías de clases y colecciones.
- Diseñar interfaces con el usuario y construir prototipos rápidos con herramientas sencillas.
- Distinguir validación de verificación y aplicar enfoques estáticos y dinámicos.
- Planificar y ejecutar pruebas de caja negra y caja blanca, generando casos de prueba mediante clases de equivalencia y análisis de cobertura.
- Aplicar desarrollo conducido por pruebas (TDD), refactorización y testeo de regresión.
- Registrar, seguir e informar fallas mediante informes técnicos, y verificar artefactos que no constituyen código.
- Trabajar en equipo integrando componentes propios en un producto final común.

4. Contenidos: organización en unidades didácticas

Unidad 1 — Principios generales de diseño

Diseño como actividad central de la ingeniería de software. Descomposición y modularidad. Acoplamiento y cohesión. Reusabilidad, portabilidad, testeabilidad, flexibilidad y escalabilidad como atributos de calidad del diseño. Análisis crítico de diseños alternativos: criterios de eficiencia y mantenibilidad. Documentación del diseño.

Unidad 2 — Patrones de diseño y arquitecturas de software

Concepto de patrón de diseño: problema, contexto y solución. Patrones creacionales, estructurales y de comportamiento (Singleton, Factory, Adapter, Observer, Strategy, entre otros). Relación de situaciones-problema con patrones. Arquitecturas de software: concepto de vistas arquitectónicas. Arquitecturas distribuidas (cliente-servidor, capas), pipe-and-filter, model-view-controller (MVC). Discusión de propiedades de calidad: escalabilidad, portabilidad, seguridad, mantenibilidad.

Unidad 3 — Diseño orientado a objetos y diseño estructurado

Diseño orientado a objetos: encapsulamiento y ocultamiento de información; separación entre comportamiento e implementación. Clases y subclases; herencia como sustitución (principio de sustitución de Liskov). Polimorfismo: subtipos vs. herencia. Jerarquías de clases. Clases colección y protocolos de iteración. Diseño estructurado. Diseño orientado al reuso de componentes: incorporación de librerías y elementos disponibles al diseño. Modelado con UML: diagramas de clases, secuencia y casos de uso.

Unidad 4 — Diseño de interfaces con el usuario y prototipos

Principios de diseño de interfaces con el usuario. Usabilidad y accesibilidad. Prototipos rápidos: bocetos, wireframes y prototipos navegables con herramientas sencillas (papel, Figma, HTML estático). El prototipo como instrumento de validación temprana con el usuario.

Unidad 5 — Fundamentos de verificación y validación

Distinción entre validación (¿construimos el producto correcto?) y verificación (¿construimos correctamente el producto?). Enfoques estáticos y dinámicos. Fundamentos de testing. Testeo de caja negra y de caja blanca. Pruebas funcionales: generación de casos y datos de prueba; clases de equivalencia y valores límite. Pruebas

estructurales: pruebas estáticas, pruebas dinámicas, cobertura de la prueba. Otros objetivos de prueba: verificación de usabilidad, confiabilidad y seguridad.

Unidad 6 — Niveles de prueba, TDD y mantenimiento del código

Prueba unitaria, de integración, de validación y prueba del sistema. Desarrollo conducido por el testeo (TDD): ciclo rojo-verde-refactor. Refactorización del código: malos olores y técnicas básicas. Testeo de regresión. Herramientas: frameworks de pruebas unitarias (Jest/Vitest para JavaScript, pytest para Python), control de versiones con Git.

Unidad 7 — Gestión de fallas e inspección de artefactos

Registro de fallas, seguimiento de fallas e informes técnicos. Ciclo de vida de un defecto. Herramientas de seguimiento (planillas, GitHub Issues). Verificación y validación de artefactos que no constituyen código: documentación, archivos de ayuda, material de capacitación. Inspecciones, revisiones cruzadas y auditorías.

5. Metodología y estrategias didácticas

La cátedra adopta una modalidad de aula-taller: cada eje conceptual se introduce con exposición dialogada breve y se consolida inmediatamente en actividades prácticas individuales y grupales. Las estrategias principales son:

- Resolución de problemas de diseño con discusión de alternativas y defensa oral de decisiones.
- Análisis de casos: lectura de código y diseños ajenos para identificar principios, patrones y defectos.
- Trabajo por proyectos: tres proyectos obligatorios que integran diseño, prototipado y testing, simulando condiciones del trabajo profesional (roles, plazos, integración de componentes).
- Revisiones cruzadas entre grupos: cada equipo prueba e inspecciona artefactos producidos por otro equipo.
- Uso de herramientas profesionales gratuitas: draw.io o Lucidchart (UML), Figma (prototipos), Git y GitHub (versionado y seguimiento de fallas), Jest/Vitest o pytest (pruebas automatizadas).

6. Práctica formativa (mínimo 33 % de la carga horaria)

Como parte de la adquisición de estos aprendizajes y demostración práctica de los resultados alcanzados, los estudiantes realizarán las siguientes actividades, que representan como mínimo el 33 % de la carga horaria anual y se concretan principalmente a través de los tres proyectos de la cátedra:

- Diseñar artefactos de software (clases, objetos, métodos, algoritmos, tablas) que resuelvan problemas planteados.
- Analizar críticamente la eficiencia y mantenibilidad de diseños alternativos.
- Relacionar situaciones con patrones de diseño.
- Analizar diversos tipos de arquitectura de sistemas de software, discutiendo sus propiedades de calidad (escalabilidad, portabilidad, seguridad, mantenibilidad).
- Construir prototipos rápidos con herramientas sencillas.
- Procesar pruebas e identificar defectos en artefactos propios y producidos por otros.
- Planificar y diseñar casos y conjuntos de datos para prueba de artefactos dados, respondiendo a objetivos y requisitos de cobertura.
- Implementar pruebas de programas y pequeños sistemas utilizando herramientas y creando los ambientes necesarios; realizar los procesos y revisar los resultados para generar informes de fallas.

- Desarrollar proyectos grupales que simulen condiciones similares a las del trabajo profesional, en los que cada estudiante aporte componentes que deben integrarse en el producto final.

7. Proyectos de la cátedra

Los tres proyectos concentran la práctica formativa y se distribuyen a lo largo del año. Son grupales (3 a 4 integrantes), con roles rotativos y entregas parciales con devolución. Cada proyecto culmina con una defensa oral.

Proyecto 1 — "Del problema al diseño": Sistema de gestión de turnos

Período: 1° cuatrimestre (abril – junio). Unidades 1 a 4.

Consigna: a partir de un enunciado real (centro de salud barrial, taller mecánico o peluquería que necesita administrar turnos, clientes y servicios), cada grupo debe producir el diseño completo del sistema, sin implementarlo todavía en profundidad.

Entregables

- Documento de diseño: descomposición en módulos, diagrama de clases UML (mínimo 6 clases con herencia y polimorfismo), diagrama de casos de uso y un diagrama de secuencia del flujo principal.
- Dos diseños alternativos para un mismo subsistema (por ejemplo, la agenda de turnos con lista vs. diccionario indexado por fecha), con análisis comparativo escrito de eficiencia, mantenibilidad y escalabilidad.
- Justificación de al menos dos patrones de diseño aplicados (por ejemplo, Singleton para la conexión de datos y Observer para notificar turnos) y elección fundamentada de arquitectura MVC.
- Prototipo rápido navegable de la interfaz (Figma o HTML estático sin lógica) validado con un "usuario" (otro grupo actúa como cliente).

Evaluación

Rúbrica sobre calidad del modelado, pertinencia de los patrones, profundidad del análisis comparativo, usabilidad del prototipo y defensa oral. Este diseño se retoma como insumo del Proyecto 3.

Proyecto 2 — "Cazadores de defectos": Plan de pruebas de una aplicación dada

Período: fin del 1° cuatrimestre y comienzo del 2° (junio – agosto). Unidades 5 y 7.

Consigna: la cátedra entrega una aplicación pequeña y funcional con defectos sembrados intencionalmente (por ejemplo, una calculadora de sueldos con validación de formularios, o un conversor de unidades web). Cada grupo debe planificar, ejecutar y documentar su prueba. Luego, en una segunda etapa, cada grupo prueba el prototipo del Proyecto 1 de otro grupo (revisión cruzada).

Entregables

- Plan de pruebas: objetivos, alcance, criterios de cobertura y ambiente de prueba.
- Tabla de clases de equivalencia y valores límite para cada entrada del sistema, con los casos de prueba derivados (mínimo 20 casos: datos de entrada, resultado esperado, resultado obtenido, veredicto).
- Ejecución de pruebas de caja negra sobre la aplicación dada y pruebas estáticas (inspección de código) identificando al menos tres defectos estructurales.
- Registro y seguimiento de fallas en GitHub Issues o planilla estandarizada: severidad, pasos para reproducir, evidencia (captura), estado.
- Informe técnico final de fallas dirigido a un "cliente" no técnico.

Evaluación

Cantidad y calidad de defectos reales detectados, corrección de las clases de equivalencia, trazabilidad entre casos de prueba y requisitos, y claridad del informe técnico.

Proyecto 3 — Integrador: "Sistema de biblioteca del instituto" con TDD e integración de componentes

Período: 2° cuatrimestre (agosto – noviembre). Unidades 3, 6 y 7. Proyecto integrador anual.

Consigna: desarrollo de un sistema de gestión de biblioteca institucional (socios, libros, préstamos y devoluciones con multas por atraso) implementado en JavaScript o Python. El curso completo trabaja sobre un mismo producto: cada grupo desarrolla un módulo (gestión de socios, catálogo, préstamos, reportes) respetando interfaces acordadas entre equipos, y al final todos los componentes deben integrarse en el producto final, simulando condiciones del trabajo profesional.

Entregables

- Diseño del módulo asignado: clases, colecciones e interfaces públicas acordadas con los demás equipos (contrato de integración firmado por todos los grupos).
- Implementación del módulo aplicando TDD: por cada funcionalidad, primero el test unitario (Jest/Vitest o pytest), luego el código; mínimo 15 pruebas unitarias por grupo con cobertura documentada.
- Al menos dos refactorizaciones documentadas (antes/después) con su correspondiente testeo de regresión demostrando que la suite sigue en verde.
- Pruebas de integración entre módulos de distintos grupos y prueba del sistema completo; registro de fallas de integración y su seguimiento hasta el cierre.
- Verificación de artefactos que no constituyen código: manual de usuario y archivo de ayuda del sistema, sometidos a inspección cruzada por otro grupo.

Evaluación

Funcionamiento del módulo y del sistema integrado, calidad y cobertura de la suite de pruebas, evidencia del ciclo TDD (historial de commits en Git), cumplimiento del contrato de interfaces, desempeño del rol asignado y defensa oral final.

8. Evaluación y acreditación

Criterios de evaluación

- Pertinencia y justificación de las decisiones de diseño; uso correcto del vocabulario técnico.
- Calidad, cobertura y trazabilidad de las pruebas diseñadas y ejecutadas.
- Rigor en el registro de fallas y claridad de los informes técnicos.
- Participación efectiva en el trabajo grupal y cumplimiento del rol asignado.
- Cumplimiento de plazos y completitud de la documentación.

Instrumentos

- Dos parciales escritos/prácticos (uno por cuatrimestre) con instancia de recuperatorio.
- Tres proyectos grupales con entregas parciales, rúbricas publicadas de antemano y defensa oral individual.
- Trabajos prácticos áulicos de resolución de problemas y análisis de casos.

Condiciones de regularidad y promoción

Regularidad: 75 % de asistencia, aprobación de los dos parciales (o sus recuperatorios) y de los tres proyectos con nota mínima de 6 (seis). Promoción directa: promedio de 8 (ocho) o superior en parciales y proyectos, con defensa oral final aprobada. Examen final: los estudiantes regulares que no promocionen rinden defensa integradora del Proyecto 3 más coloquio teórico. Estas condiciones se ajustan al Régimen Académico Institucional vigente.

10. Bibliografía

- Gamma, E., Helm, R., Johnson, R. y Vlissides, J. (1995). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.
- Fowler, M. (2018). Refactoring: Improving the Design of Existing Code (2ª ed.). Addison-Wesley.
- Beck, K. (2002). Test-Driven Development: By Example. Addison-Wesley.
- Myers, G., Sandler, C. y Badgett, T. (2011). The Art of Software Testing (3ª ed.). Wiley.