Universidad Nacional de Ingeniería (UNI)
Escuela Profesional de
Ciencia de la Computación
Sílabo 2026-I

1. CURSO

CS292. Ingeniería de Software II (Obligatorio)

2. INFORMACIÓN GENERAL

2.1 Curso	:	CS292. Ingeniería de Software II
2.2 Semestre	:	7^{mo} Semestre.
2.3 Créditos	:	4
2.4 Horas	:	2 HT; 4 HP;
2.5 Duración del periodo	:	16 semanas
2.6 Condición	:	Obligatorio
2.7 Modalidad de aprendizaje	:	Presencial
2.8 Prerrequisitos	:	CS291. Ingeniería de Software I. (5^{to} Sem)

3. PROFESORES

Atención previa coordinación con el profesor

4. INTRODUCCIÓN AL CURSO

Ingeniería de Software II se basa en los principios fundamentales del primer curso, cambiando el enfoque hacia patrones de diseño avanzados, evolución de software y la gestión de proyectos a gran escala. Explora las complejidades de mantener sistemas heredados, asegurar la calidad del software mediante validación rigurosa, y la aplicación de técnicas profesionales de gestión de proyectos en entornos ágiles.

5. OBJETIVOS

- Aplicar patrones de diseño orientados a objetos avanzados para resolver problemas arquitectónicos complejos.
- Dominar la gestión de configuración de software y control de versiones en entornos colaborativos.
- Comprender los principios de evolución y mantenimiento de software.
- Gestionar proyectos de software usando técnicas de estimación y análisis de riesgos.
- Implementar estrategias de prueba avanzadas y métricas de calidad.

6. RESULTADOS DEL ESTUDIANTE

AG-C09) Diseño y Desarrollo de Soluciones: Diseña, implementa y evalúa soluciones para problemas complejos de computación. (Usage)

AG-C03) Trabajo Individual y en Equipo: Se desempeña efectivamente como individuo y como miembro o líder en equipos diversos. (Usage)

7. TEMAS

Unidad 1: Colaboración y Dinámica de Equipo (14 horas)	
Resultados esperados: AG-C03,AG-C09	
Temas	**Aprendizaje esperado (*Learning Outcomes*)**
• Comunicación efectiva, incluida la oral y escrita, así como formal (correo electrónico, documentos, comentarios, presentaciones) e informal (chat de equipo, reuniones). Sociedad, ética y la Profesión (SEP)-Communication • Causas comunes de conflicto en el equipo y enfoques para la resolución de conflictos. • Programación cooperativa: – Programación por parejas (pair programming) o Enjambre (swarming) – Revisión de código – Colaboración mediante control de versiones • Roles y responsabilidades en un equipo de software: Sociedad, ética y la Profesión (SEP)-ProfessionalEthics – Ventajas del trabajo en equipo – Riesgos y complejidad de dicha colaboración • Procesos del equipo: responsabilidades para las tareas, estimación de esfuerzo, estructura de reuniones, cronograma de trabajo	• Seguir prácticas efectivas de comunicación en equipo [Analizar] • Articular las fuentes, peligros y beneficios potenciales del conflicto en el equipo, especialmente enfocándose en el valor de estar en desacuerdo sobre ideas o propuestas sin insultar a las personas [Articular] • Facilitar una estrategia de resolución de conflictos y resolución de problemas en un entorno de equipo [Analizar] • Colaborar efectivamente en desarrollo/programación cooperativa [Analizar] • Proponer y delegar los roles y responsabilidades necesarios en un equipo de desarrollo de software [Proponer] • Redactar y seguir una agenda para una reunión de equipo [Componer] • Facilitar, a través de la participación en un proyecto de equipo, los elementos centrales de la formación de equipos, el establecimiento de una cultura de equipo saludable y la gestión de equipos, incluida la creación y ejecución de un plan de trabajo del equipo [Crear]
Lecturas : [Som15], [PM19]	

Unidad 2: Principios y Arquitectura de Software (16 horas)	
Resultados esperados: AG-C03,AG-C09	
Temas	**Aprendizaje esperado (*Learning Outcomes*)**
• Principios de diseño de sistemas. Fundamentos de Sistemas (SF)-Reliability – Niveles de abstracción (por ejemplo, diseño arquitectónico y diseño detallado) – Separación de preocupaciones (separation of concerns) – Ocultación de información (information hiding) – Acoplamiento y cohesión (coupling and cohesion) • Arquitectura de software. Fundamentos de Sistemas (SF)-Reliability – Paradigmas de diseño ∗ Descomposición funcional de arriba hacia abajo/diseño por capas ∗ Arquitectura orientada a datos ∗ Análisis y diseño orientado a objetos ∗ Diseño dirigido por eventos – Arquitecturas estándar (por ejemplo, cliente-servidor y arquitecturas de microservicios incluyendo discusiones sobre REST, n-capas, tuberías y filtros, Modelo Vista Controlador) – Identificar límites y dependencias de componentes • Programación a gran escala vs programación a pequeña escala (programming in the large vs programming in the small). Fundamentos de Sistemas (SF)-Reliability • Malos olores del código (code smells) y otras indicaciones de la calidad del código, distintas de la corrección. Seguridad (SEC)-Engineering	• Identificar la arquitectura de software estándar de un diseño de alto nivel dado [Analizar] • Seleccionar y usar un paradigma de diseño apropiado para diseñar un sistema de software simple y explicar cómo se han aplicado los principios de diseño de sistemas en este diseño [Evaluar] • Adaptar un diseño de sistema defectuoso para que siga mejor principios como la separación de preocupaciones o la ocultación de información [Crear] • Identificar las dependencias entre un conjunto de componentes de software en un diseño arquitectónico [Analizar]
Lecturas : [Gam+94], [PM19]	

Unidad 3: Prácticas de Codificación (14 horas)	
Resultados esperados: AG-C03,AG-C09	
Temas	**Aprendizaje esperado (*Learning Outcomes*)**
• Pruebas prácticas a pequeña escala Fundamentos de Desarrollo de Software (SDF)-Practices – Pruebas unitarias – Desarrollo guiado por pruebas (Test-driven development): esto es particularmente valioso para los estudiantes psicológicamente, ya que es mucho más fácil involucrarse constructivamente con el desafío de identificar entradas desafiantes para una API dada (casos límite, casos extremos) a priori. Si implementan primero, el instinto es a menudo evitar intentar hacer fallar su nueva creación, mientras que un enfoque de pruebas primero les da la satisfacción intelectual de detectar los casos problemáticos y luego ver cómo pasan más pruebas durante el proceso de desarrollo. • Documentación Fundamentos de Desarrollo de Software (SDF)-Practices – Documentación de la interfaz: describir los requisitos de la interfaz, potencialmente incluyendo contratos (formales o informales), pre y post condiciones, invariantes. – La documentación de la implementación debe centrarse en partes de código complicadas y no obvias, ya sea porque el código usa características avanzadas del lenguaje o porque el comportamiento del código es complejo. (No agregar comentarios que reafirmen operaciones comunes/obvias y características simples del lenguaje.) ∗ Aclarar el flujo de datos, cómputo, etc., centrándose en lo que el código es. ∗ Identificar partes sutiles/complicadas del código y refactorizar para que sean autoexplicativas si es posible o proporcionar comentarios apropiados para aclarar. • Estilo de codificación Fundamentos de Desarrollo de Software (SDF)-Practices – Guías de estilo – Comentarios – Nomenclatura (*naming*) • "Mejores Prácticas" para la codificación: técnicas, patrones/idiomas, mecanismos para construir programas de calidad Seguridad (SEC)-Coding,SDF-Practices – Prácticas de codificación defensiva – Prácticas y principios de codificación segura – Usar mecanismos de manejo de excepciones para hacer los programas más robustos, tolerantes a fallos	• Escribir pruebas unitarias apropiadas para un componente pequeño (varias funciones, un solo tipo, etc.) [Escribir] • Escribir comentarios apropiados de interfaz y (si es necesario) de implementación para un componente pequeño [Escribir] • Describir técnicas, patrones de codificación y mecanismos para implementar diseños para lograr propiedades deseadas como confiabilidad, eficiencia y robustez [Describir] • Escribir código robusto utilizando mecanismos de manejo de excepciones [Escribir] • Describir prácticas de codificación segura y defensiva [Describir] • Seleccionar y usar un estándar de codificación definido en un proyecto de software pequeño [Usar] • Comparar y contrastar estrategias de integración, incluyendo integración de arriba hacia abajo (top-down), de abajo hacia arriba (bottom-up) y sándwich [Comparar] • Describir el proceso de analizar e implementar cambios en la base de código desarrollada para un proyecto específico [Describir] • Describir el proceso de analizar e implementar cambios en una base de código existente grande [Describir]

Unidad 4: Conceptos de verificación y validación (10 horas)	
Resultados esperados: AG-C03,AG-C09	
Temas	**Aprendizaje esperado (*Learning Outcomes*)**
• Conceptos de verificación y validación – Verificación: ¿Estamos construyendo la cosa correctamente? – Validación: ¿Construimos la cosa correcta? • Por qué importan las pruebas: ¿El componente sigue siendo funcional a medida que el código evoluciona? • Objetivos de las pruebas – Usabilidad – Confiabilidad – Conformidad con la especificación – Rendimiento – Seguridad	• Explicar por qué las pruebas son importantes [Explicar] • Distinguir entre validación y verificación de programas [Distinguir] • Describir diferentes objetivos de las pruebas [Describir]
Lecturas : [PM19]	

Unidad 5: Herramientas y Análisis de Pruebas (4 horas)	
Resultados esperados: AG-C03,AG-C09	
Temas	**Aprendizaje esperado (*Learning Outcomes*)**
• Herramientas y automatización de verificación y validación – Análisis estático – Cobertura de código – Fuzzing – Análisis dinámico y contención de fallos (sanitizadores, etc.) – Registro y seguimiento de fallos • Planificación y generación de pruebas – Estimación de fallos y terminación de pruebas, incluyendo siembra de defectos – Uso de números aleatorios y pseudoaleatorios en las pruebas • Pruebas de sistemas asíncronos, paralelos y concurrentes • Verificación y validación de artefactos que no son código (documentación, materiales de capacitación)	• Describir y comparar diferentes herramientas para verificación y validación [Describir] • Automatizar el uso de diferentes herramientas en un proyecto de software pequeño [Diseñar] • Explicar cómo y cuándo se deben usar números aleatorios en las pruebas [Explicar] • Describir enfoques para la estimación de fallos [Describir] • Estimar el número de fallos en una aplicación de software pequeña basándose en la densidad de fallos y la siembra de fallos [Estimar] • Describir técnicas y problemas con las pruebas de software asíncrono, concurrente y paralelo [Describir] • Crear un plan de pruebas para un segmento de código de tamaño mediano que contenga código asíncrono, concurrente y/o paralelo, incluyendo una matriz de pruebas y generación de datos y entradas de prueba [Crear] • Describir técnicas para la verificación y validación de artefactos que no son código [Describir]
Lecturas : [Som15], [PM19]	

Unidad 6: Pruebas de Rendimiento y Evaluación Comparativa (4 horas)	
Resultados esperados: AG-C03,AG-C09	
Temas	**Aprendizaje esperado (*Learning Outcomes*)**
• Pruebas de rendimiento y evaluación comparativa – Rendimiento (*throughput*) y latencia – Degradación bajo carga (pruebas de estrés, manejo FIFO vs LIFO de solicitudes) – Aceleración (*speedup*) y escalado (*scaling*) ∗ Ley de Amdahl ∗ Ley de Gustafson ∗ Escalado suave (*soft*) y débil (*weak*) – Identificar y medir figuras de mérito (*figures of merit*) – Cuellos de botella de rendimiento comunes ∗ Limitado por cómputo (*compute-bound*) ∗ Limitado por ancho de banda de memoria ∗ Limitado por latencia – Métodos estadísticos y mejores prácticas para la evaluación comparativa (benchmarking) ∗ Estimación de la incertidumbre ∗ Intervalos de confianza – Análisis y presentación (gráficos, etc.) – Técnicas de medición de tiempo (*timing*)	• Describir el rendimiento (*throughput*) y la latencia y proporcionar ejemplos de cada uno [Describir] • Explicar la aceleración (speedup) y las diferentes formas de escalado (scaling) y cómo se calculan [Explicar] • Describir cuellos de botella de rendimiento comunes [Describir] • Describir métodos estadísticos y mejores prácticas para la evaluación comparativa (benchmarking) de software [Describir] • Explicar técnicas y desafíos para medir el tiempo al construir una evaluación comparativa (benchmark) [Explicar] • Identificar las figuras de mérito, construir y ejecutar una evaluación comparativa (benchmark) y analizar estadísticamente y visualizar los resultados para un proyecto de software pequeño [Analizar]
Lecturas : [Bon15], [Gre20]	

Unidad 7: Conceptos de Confiabilidad (10 horas)	
Resultados esperados: AG-C03,AG-C09	
Temas	**Aprendizaje esperado (*Learning Outcomes*)**
• Concepto de confiabilidad como probabilidad de fallo o tiempo medio entre fallos, y fallas (faults) como causa de fallos (failures) • Identificar requisitos de confiabilidad para diferentes tipos de software • Fallos de software causados por defectos/errores (bugs), por lo que para alta confiabilidad el objetivo es tener un mínimo de defectos - inyectando menos defectos (mejor capacitación, educación, planificación) y eliminando la mayoría de los defectos inyectados (pruebas, revisión de código, etc.) • Confiabilidad del software, confiabilidad del sistema y comportamiento de fallos • Ciclo de inyección y eliminación de defectos, y diferentes enfoques para la eliminación de defectos • Comparar el enfoque de "presupuesto de error" (error budget) para la confiabilidad con el enfoque "libre de errores" (error-free) e identificar dominios donde cada uno es relevante.	• Describir cómo determinar el nivel de confiabilidad requerido por un sistema de software [Describir] • Explicar los problemas que existen para lograr niveles muy altos de confiabilidad [Explicar] • Comprender enfoques para minimizar fallas que pueden aplicarse en cada etapa del ciclo de vida del software [Explicar]
Lecturas : [Som15]	

8. PLAN DE TRABAJO

8.1 Metodología

Se fomenta la participación individual y en equipo para exponer sus ideas, motivándolos con puntos adicionales en las diferentes etapas de la evaluación del curso.

8.2 Sesiones Teóricas

Las sesiones de teoría se llevan a cabo en clases magistrales donde se realizarán actividades que propicien un aprendizaje activo, con dinámicas que permitan a los estudiantes interiorizar los conceptos.

8.3 Sesiones Prácticas

Las sesiones prácticas se llevan en clase donde se desarrollan una serie de ejercicios y/o conceptos prácticos mediante planteamiento de problemas, la resolución de problemas, ejercicios puntuales y/o en contextos aplicativos.

9. SISTEMA DE EVALUACIÓN

********* EVALUATION MISSING ********

10. BIBLIOGRAFÍA BÁSICA

[Gam+94] Erich Gamma et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[Bon15] André B. Bondi. *Foundations of Software and System Performance Engineering: Process, Performance Modeling, Requirements, Testing, Scalability, and Practice*. Upper Saddle River, NJ: Addison-Wesley, 2015.

[Som15] Ian Sommerville. *Software Engineering*. 10th. Pearson, 2015.

[PM19] Roger S. Pressman and Bruce Maxim. *Software Engineering: A Practitioner's Approach*. 9th. McGraw-Hill Education, 2019.

[Gre20] Brendan Gregg. *Systems Performance: Enterprise and the Cloud*. 2nd. Boston, MA: Addison-Wesley Professional, 2020.