Universidad Nacional de Ingeniería (UNI)
Escuela Profesional de
Inteligencia Artificial
Sílabo 2027-I

1. CURSO

CS291. Ingeniería de Software I (Obligatorio)

2. INFORMACIÓN GENERAL

2.1 Curso	:	CS291. Ingeniería de Software I
2.2 Semestre	:	5to 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	:	▪ CS113. Programación Orientada a Objetos II. (3er Sem) ▪ CS271. Bases de Datos I. (4to Sem)

3. PROFESORES

Atención previa coordinación con el profesor

4. INTRODUCCIÓN AL CURSO

La ingeniería de software es una disciplina colaborativa que aplica procesos estructurados para construir sistemas de alta calidad. Este primer curso se enfoca en los aspectos fundamentales del ciclo de vida del software, desde los requisitos iniciales y el diseño centrado en el usuario hasta las pruebas sistemáticas. Al integrar principios de HCI y fundamentos de desarrollo rigurosos, los estudiantes aprenderán a crear software que sea técnicamente sólido y orientado al usuario.

5. OBJETIVOS

- Comprender y aplicar varios modelos de proceso de software y gestión de configuración.
- Dominar la elicitación y modelado de requisitos usando notaciones estandarizadas.
- Diseñar interfaces de usuario intuitivas basadas en principios de evaluación y diseño de HCI.
- Implementar estrategias sistemáticas de prueba y comprender la calidad del software.
- Desarrollar habilidades profesionales para el desarrollo colaborativo de software.

6. RESULTADOS DEL ESTUDIANTE

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

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

7. TEMAS

Unidad 1: Control de Versiones y CI/CD (12 horas)	
Resultados esperados: AG-C03,AG-C09	
Temas	**Aprendizaje esperado (*Learning Outcomes*)**

- Gestión de configuración de software y control de versiones: Prácticas de Desarrollo de Software
  - Configuración en control de versiones, builds/configuración reproducibles.
  - Estrategias de ramificación en control de versiones. Ramas de desarrollo vs ramas de lanzamiento. Desarrollo basado en tronco (*trunk-based development*).
  - Estrategias de fusión/rebase, cuando sean relevantes.
- Gestión de lanzamientos (*release management*).
- Herramientas de prueba, incluyendo herramientas de análisis estático y dinámico. Prácticas de Desarrollo de Software, Flujo de Información y No Interferencia, Inyección y Validación de Entrada, Seguridad de Memoria y Tipos, Análisis de Malware y Seguridad Avanzada
- Automatización de procesos de software:
  - Sistemas de construcción (*build systems*): el valor de builds rápidos, herméticos y reproducibles, comparar/contrastar enfoques para construir un proyecto.
  - Integración Continua (CI): el uso de automatización y pruebas automatizadas para hacer una validación preliminar de que la revisión actual del tronco se construye y pasa las pruebas (básicas).
  - Despliegue Continuo (CD): el uso de automatización para liberar de forma automática cada cambio que supera las pruebas hacia el entorno de producción, garantizando entregas frecuentes y confiables.
  - Gestión de dependencias: actualización de dependencias externas/aguas arriba, gestión de paquetes, SemVer.
- Flujo de revisión colaborativa como puerta de calidad
  - Revisión obligatoria de *pull requests* antes de fusionar
  - Propiedad del código mediante reglas de tipo CODEOWNERS
  - Convenciones de mensajes de *commit* (*conventional commits*)
  - Reglas de protección de ramas que exigen verificaciones antes de fusionar
- Escaneo de vulnerabilidades de dependencias y de la cadena de suministro integrado en los pipelines de CI
	- Describir la diferencia entre la gestión de configuración de software centralizada y distribuida [Describir]
	- Describir cómo el control de versiones puede usarse para ayudar en la gestión de lanzamientos de software [Describir]
	- Identificar elementos de configuración y usar una herramienta de control de código fuente en un proyecto pequeño basado en equipo [Analizar]
	- Describir cómo las herramientas de prueba estáticas y dinámicas disponibles pueden integrarse en el entorno de desarrollo de software [Describir]
	- Comprender el uso de sistemas de CI/CD como una fuente de verdad para el estado del código compartido del equipo (éxito en la construcción y pruebas) [Explicar]
	- Aplicar un flujo de revisión de *pull requests* con reglas de propiedad de código y protección de ramas para imponer puertas de calidad en un proyecto de equipo [Aplicar]
	- Usar un escáner de vulnerabilidades de dependencias/cadena de suministro como parte de un pipeline de CI para detectar dependencias vulnerables [Usar]

Lecturas : [**Sommerville2015**], [**Pressman2019**]	

Unidad 2: Ingeniería de Requisitos (14 horas)	
Resultados esperados: AG-C03,AG-C09	
Temas	**Aprendizaje esperado (*Learning Outcomes*)**
▪ Describir requisitos funcionales usando, por ejemplo, casos de uso o historias de usuario. • Usar al menos un método para documentar y estructurar requisitos funcionales. • Comprender cómo el método apoya el diseño y la implementación. • Fortalezas y debilidades de usar un enfoque específico. ▪ Propiedades de los requisitos, incluyendo consistencia, validez, completitud y viabilidad. ▪ Obtención (*elicitation*) de requisitos. • Fuentes de requisitos, por ejemplo, usuarios, administradores o personal de soporte. • Métodos de recopilación de requisitos, por ejemplo, encuestas, entrevistas o análisis de comportamiento. ▪ Requisitos no funcionales, por ejemplo, seguridad, usabilidad o rendimiento, también llamados Atributos de Calidad. Diseño Sostenible y Computación Ubicua, Huella Ambiental de los Sistemas Informáticos, Efectos Sistémicos y Contexto Social ▪ Identificación y gestión de riesgos, incluyendo consideraciones éticas en torno al producto propuesto. Códigos de Ética y Responsabilidad Legal, Responsabilidad Profesional y Política Pública, Valores, Naturaleza y Actualización Profesional, Disidencia Ética y Contexto Cultural, Ética en el Lugar de Trabajo ▪ Comunicar y/o formalizar especificaciones de requisitos.	▪ Comparar diferentes métodos de obtención de requisitos a lo largo de múltiples ejes [Comparar] ▪ Identificar diferencias entre dos métodos de describir requisitos funcionales (por ejemplo, entrevistas con clientes, estudios de usuarios) y las situaciones donde se preferiría cada uno [Analizar] ▪ Identificar qué comportamientos son requeridos, permitidos o prohibidos a partir de un conjunto dado de requisitos y una lista de comportamientos candidatos [Analizar] ▪ Recopilar un conjunto de requisitos para un sistema de software simple [Analizar] ▪ Identificar áreas de un sistema de software que deben cambiarse, dada una descripción del sistema y un conjunto de nuevos requisitos a implementar [Analizar] ▪ Identificar los requisitos funcionales y no funcionales en un conjunto de requisitos [Analizar]
Lecturas : [**Sommerville2015**], [**Wiegers2013**]	

Unidad 3: 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). Comunicación Técnica y Profesional, Comunicación en Equipo y Cultura ▪ 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: Códigos de Ética y Responsabilidad Legal, Responsabilidad Profesional y Política Pública, Valores, Naturaleza y Actualización Profesional, Disidencia Ética y Contexto Cultural, Ética en el Lugar de Trabajo • 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 : [**Wigdor2011**], [**Johnson2013**]	

Unidad 4: Principios y Arquitectura de Software (12 horas)	
Resultados esperados: AG-C03,AG-C09	
Temas	**Aprendizaje esperado (*Learning Outcomes*)**

- Principios de diseño de sistemas. Confiabilidad del Sistema
  - 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. Confiabilidad del Sistema
  - 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). Confiabilidad del Sistema
- Malos olores del código (code smells) y otras indicaciones de la calidad del código, distintas de la corrección. Diseño e Ingeniería de Controles de Seguridad, Análisis de Amenazas e Ingeniería de Seguridad, Computación Confiable e Ingeniería de Privacidad
	- 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 : [**Gamma1994**], [**Pressman2019**]	

Unidad 5: Conceptos de verificación y validación (12 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 : [**Sommerville2015**], [**Pressman2019**]	

Unidad 6: 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)
- Desarrollo guiado por comportamiento (BDD)
  - Expresar criterios de aceptación como escenarios Gherkin Given/When/Then
  - Ejecutar escenarios BDD como pruebas automatizadas de extremo a extremo
- Pruebas de mutación y análisis estático de calidad de código
  - Pruebas de mutación para evaluar la efectividad de una suite de pruebas
  - Detección estática de *code smells* y complejidad ciclomática excesiva
	- 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]
	- Escribir escenarios Gherkin Given/When/Then para los criterios de aceptación de una funcionalidad y ejecutarlos como pruebas automatizadas de extremo a extremo [Escribir]
	- Analizar el puntaje de mutación de una suite de pruebas y refactorizar las pruebas para eliminar los mutantes sobrevivientes [Analizar]

Lecturas : [**Sommerville2015**], [**Pressman2019**]	

Unidad 7: 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 : [**Bondi2015**], [**Gregg2020**]	

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