**National University of Engineering (UNI)**
School of Computer Science
Syllabus 2026-I

## 1. COURSE

CS392. Tópicos en Ingeniería de Software (Elective)

## 2. GENERAL INFORMATION

| | | |
|---|---|---|
| **2.1 Course** | : | CS392. Tópicos en Ingeniería de Software |
| **2.2 Semester** | : | $9^{th}$ Semester. |
| **2.3 Credits** | : | 3 |
| **2.4 Hours** | : | 2 HT; 2 HP; |
| **2.5 Duration of the period** | : | 16 weeks |
| **2.6 Type of course** | : | Elective |
| **2.7 Learning modality** | : | Face to face |
| **2.8 Prerrequisites** | : | CS391. Software Engineering III. ($8^{th}$ Sem) |

## 3. PROFESSORS

Meetings after coordination with the professor

## 4. INTRODUCTION TO THE COURSE

Modern software development requires rigorous engineering practices to ensure quality, maintainability, and reliability. This course explores advanced topics in the software lifecycle, focusing on team collaboration, formal verification, and advanced refactoring. Students will master the use of professional tools to design, construct, and validate complex systems while ensuring high standards of reliability and formal correctness.

## 5. GOALS

- Apply advanced software construction and refactoring techniques to improve code quality.
- Use formal methods and validation strategies to ensure system reliability.
- Manage complex requirements and design architectures that meet stakeholder needs.
- Lead software teams using professional tools and collaborative workflows.

## 6. COMPETENCES

**AG-C09)** Design and Development of Solutions: Designs, implements, and evaluates solutions for complex computing problems. (Usage)

**AG-C03)** Individual and Team Work: Performs effectively as an individual and as a member or leader in diverse teams. (Usage)

## 7. TOPICS

**Unit 1: Colaboración y Dinámica de Equipo (4 hours)**

**Competences Expected: AG-C03,AG-C09**

| Topics | 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<br>• Causas comunes de conflicto en el equipo y enfoques para la resolución de conflictos.<br>• Programación cooperativa:<br>– Programación por parejas (pair programming) o Enjambre (swarming)<br>– Revisión de código<br>– Colaboración mediante control de versiones<br>• Roles y responsabilidades en un equipo de software: Sociedad, ética y la Profesión (SEP)-ProfessionalEthics<br>– Ventajas del trabajo en equipo<br>– Riesgos y complejidad de dicha colaboración<br>• Procesos del equipo: responsabilidades para las tareas, estimación de esfuerzo, estructura de reuniones, cronograma de trabajo | • Seguir prácticas efectivas de comunicación en equipo [Analizar]<br>• 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]<br>• Facilitar una estrategia de resolución de conflictos y resolución de problemas en un entorno de equipo [Analizar]<br>• Colaborar efectivamente en desarrollo/programación cooperativa [Analizar]<br>• Proponer y delegar los roles y responsabilidades necesarios en un equipo de desarrollo de software [Proponer]<br>• Redactar y seguir una agenda para una reunión de equipo [Componer]<br>• 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] |

**Readings :** [Som15]

**Unit 2: Control de Versiones y CI/CD (4 hours)**

**Competences Expected: AG-C03,AG-C09**

| **Topics** | **Learning Outcomes** |
|---|---|
| • Gestión de configuración de software y control de versiones: Fundamentos de Desarrollo de Software (SDF)-Practices<br>– Configuración en control de versiones, builds/configuración reproducibles.<br>– Estrategias de ramificación en control de versiones. Ramas de desarrollo vs ramas de lanzamiento. Desarrollo basado en tronco (*trunk-based development*).<br>– Estrategias de fusión/rebase, cuando sean relevantes.<br>• Gestión de lanzamientos (*release management*).<br>• Herramientas de prueba, incluyendo herramientas de análisis estático y dinámico. Fundamentos de Desarrollo de Software (SDF)-Practices,SEC-Coding<br>• Automatización de procesos de software:<br>– Sistemas de construcción (*build systems*): el valor de builds rápidos, herméticos y reproducibles, comparar/contrastar enfoques para construir un proyecto.<br>– 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).<br>– 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.<br>– Gestión de dependencias: actualización de dependencias externas/aguas arriba, gestión de paquetes, SemVer. | • Describir la diferencia entre la gestión de configuración de software centralizada y distribuida [Describir]<br>• Describir cómo el control de versiones puede usarse para ayudar en la gestión de lanzamientos de software [Describir]<br>• Identificar elementos de configuración y usar una herramienta de control de código fuente en un proyecto pequeño basado en equipo [Analizar]<br>• Describir cómo las herramientas de prueba estáticas y dinámicas disponibles pueden integrarse en el entorno de desarrollo de software [Describir]<br>• 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] |

**Readings :** [PM14]

| **Unit 3: Ingeniería de Requisitos (6 hours)** | |
|---|---|
| **Competences Expected: AG-C03,AG-C09** | |
| **Topics** | **Learning Outcomes** |
| • Describir requisitos funcionales usando, por ejemplo, casos de uso o historias de usuario.<br>– Usar al menos un método para documentar y estructurar requisitos funcionales.<br>– Comprender cómo el método apoya el diseño y la implementación.<br>– Fortalezas y debilidades de usar un enfoque específico.<br>• Propiedades de los requisitos, incluyendo consistencia, validez, completitud y viabilidad.<br>• Obtención (*elicitation*) de requisitos.<br>– Fuentes de requisitos, por ejemplo, usuarios, administradores o personal de soporte.<br>– Métodos de recopilación de requisitos, por ejemplo, encuestas, entrevistas o análisis de comportamiento.<br>• Requisitos no funcionales, por ejemplo, seguridad, usabilidad o rendimiento, también llamados Atributos de Calidad. Sociedad, ética y la Profesión (SEP)-Sustainability<br>• Identificación y gestión de riesgos, incluyendo consideraciones éticas en torno al producto propuesto. Sociedad, ética y la Profesión (SEP)-ProfessionalEthics<br>• Comunicar y/o formalizar especificaciones de requisitos. | • Comparar diferentes métodos de obtención de requisitos a lo largo de múltiples ejes [Comparar]<br>• 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]<br>• Identificar qué comportamientos son requeridos, permitidos o prohibidos a partir de un conjunto dado de requisitos y una lista de comportamientos candidatos [Analizar]<br>• Recopilar un conjunto de requisitos para un sistema de software simple [Analizar]<br>• Identificar áreas de un sistema de software que deben cambiarse, dada una descripción del sistema y un conjunto de nuevos requisitos a implementar [Analizar]<br>• Identificar los requisitos funcionales y no funcionales en un conjunto de requisitos [Analizar] |
| **Readings :** [Som15] | |

**Unit 4: Principios y Arquitectura de Software (6 hours)**

**Competences Expected: AG-C03,AG-C09**

| Topics | Learning Outcomes |
|---|---|
| • Principios de diseño de sistemas. Fundamentos de Sistemas (SF)-Reliability<br>– Niveles de abstracción (por ejemplo, diseño arquitectónico y diseño detallado)<br>– Separación de preocupaciones (separation of concerns)<br>– Ocultación de información (information hiding)<br>– Acoplamiento y cohesión (coupling and cohesion)<br>• Arquitectura de software. Fundamentos de Sistemas (SF)-Reliability<br>– Paradigmas de diseño<br>∗ Descomposición funcional de arriba hacia abajo/diseño por capas<br>∗ Arquitectura orientada a datos<br>∗ Análisis y diseño orientado a objetos<br>∗ Diseño dirigido por eventos<br>– Arquitecturas estándar (por ejemplo, cliente-servidor y arquitecturas de microservicios incluyendo discusiones sobre REST, n-capas, tuberías y filtros, Modelo Vista Controlador)<br>– Identificar límites y dependencias de componentes<br>• Programación a gran escala vs programación a pequeña escala (programming in the large vs programming in the small). Fundamentos de Sistemas (SF)-Reliability<br>• 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]<br>• 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]<br>• Adaptar un diseño de sistema defectuoso para que siga mejor principios como la separación de preocupaciones o la ocultación de información [Crear]<br>• Identificar las dependencias entre un conjunto de componentes de software en un diseño arquitectónico [Analizar] |
| **Readings :** [Som15] | |

| Unit 5: Prácticas de Codificación (6 hours) | |
|---|---|
| **Competences Expected: AG-C03,AG-C09** | |
| **Topics** | **Learning Outcomes** |
| • Pruebas prácticas a pequeña escala Fundamentos de Desarrollo de Software (SDF)-Practices<br>– Pruebas unitarias<br>– 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.<br>• Documentación Fundamentos de Desarrollo de Software (SDF)-Practices<br>– Documentación de la interfaz: describir los requisitos de la interfaz, potencialmente incluyendo contratos (formales o informales), pre y post condiciones, invariantes.<br>– 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.)<br>∗ Aclarar el flujo de datos, cómputo, etc., centrándose en lo que el código es.<br>∗ Identificar partes sutiles/complicadas del código y refactorizar para que sean autoexplicativas si es posible o proporcionar comentarios apropiados para aclarar.<br>• Estilo de codificación Fundamentos de Desarrollo de Software (SDF)-Practices<br>– Guías de estilo<br>– Comentarios<br>– Nomenclatura (*naming*)<br>• "Mejores Prácticas" para la codificación: técnicas, patrones/idiomas, mecanismos para construir programas de calidad Seguridad (SEC)-Coding,SDF-Practices<br>– Prácticas de codificación defensiva<br>– Prácticas y principios de codificación segura<br>– 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]<br>• Escribir comentarios apropiados de interfaz y (si es necesario) de implementación para un componente pequeño [Escribir]<br>• Describir técnicas, patrones de codificación y mecanismos para implementar diseños para lograr propiedades deseadas como confiabilidad, eficiencia y robustez [Describir]<br>• Escribir código robusto utilizando mecanismos de manejo de excepciones [Escribir]<br>• Describir prácticas de codificación segura y defensiva [Describir]<br>• Seleccionar y usar un estándar de codificación definido en un proyecto de software pequeño [Usar]<br>• Comparar y contrastar estrategias de integración, incluyendo integración de arriba hacia abajo (top-down), de abajo hacia arriba (bottom-up) y sándwich [Comparar]<br>• Describir el proceso de analizar e implementar cambios en la base de código desarrollada para un proyecto específico [Describir]<br>• Describir el proceso de analizar e implementar cambios en una base de código existente grande [Describir] |

| **Unit 6: Compatibilidad y Versionado de API (6 hours)** | |
|---|---|
| **Competences Expected: AG-C03,AG-C09** | |
| **Topics** | **Learning Outcomes** |
| • Ley de Hyrum / Ley de las Interfaces Implícitas<br>• Compatibilidad con versiones anteriores<br>– La compatibilidad no es una propiedad de una sola entidad, es una propiedad de una relación.<br>– La compatibilidad con versiones anteriores debe evaluarse en términos de proveedor + consumidor(es) o con un modelo bien especificado de qué formas de compatibilidad aspira/promete un proveedor.<br>• Control de versiones (versioning)<br>– Versionado Semántico (SemVer)<br>– Desarrollo basado en tronco (*trunk-based development*) | • Identificar tanto el comportamiento explícito como el implícito de una interfaz e identificar riesgos potenciales de la Ley de Hyrum [Analizar]<br>• Identificar cambios que pueden considerarse ampliamente "compatibles con versiones anteriores", potencialmente con declaraciones explícitas sobre qué uso está o no soportado [Analizar]<br>• Evaluar si un cambio propuesto es lo suficientemente seguro dada la metodología de control de versiones en uso para un proyecto dado [Evaluar] |
| **Readings :** [Fow18] | |

| **Unit 7: Conceptos de verificación y validación (6 hours)** | |
|---|---|
| **Competences Expected: AG-C03,AG-C09** | |
| **Topics** | **Learning Outcomes** |
| • Conceptos de verificación y validación<br>– Verificación: ¿Estamos construyendo la cosa correctamente?<br>– Validación: ¿Construimos la cosa correcta?<br>• Por qué importan las pruebas: ¿El componente sigue siendo funcional a medida que el código evoluciona?<br>• Objetivos de las pruebas<br>– Usabilidad<br>– Confiabilidad<br>– Conformidad con la especificación<br>– Rendimiento<br>– Seguridad | • Explicar por qué las pruebas son importantes [Explicar]<br>• Distinguir entre validación y verificación de programas [Distinguir]<br>• Describir diferentes objetivos de las pruebas [Describir] |
| **Readings :** [PM14] | |

| **Unit 8: Herramientas y Análisis de Pruebas (3 hours)** | |
|---|---|
| **Competences Expected: AG-C03,AG-C09** | |
| **Topics** | **Learning Outcomes** |
| • Herramientas y automatización de verificación y validación<br>– Análisis estático<br>– Cobertura de código<br>– Fuzzing<br>– Análisis dinámico y contención de fallos (sanitizadores, etc.)<br>– Registro y seguimiento de fallos<br>• Planificación y generación de pruebas<br>– Estimación de fallos y terminación de pruebas, incluyendo siembra de defectos<br>– Uso de números aleatorios y pseudoaleatorios en las pruebas<br>• Pruebas de sistemas asíncronos, paralelos y concurrentes<br>• Verificación y validación de artefactos que no son código (documentación, materiales de capacitación) | • Describir y comparar diferentes herramientas para verificación y validación [Describir]<br>• Automatizar el uso de diferentes herramientas en un proyecto de software pequeño [Diseñar]<br>• Explicar cómo y cuándo se deben usar números aleatorios en las pruebas [Explicar]<br>• Describir enfoques para la estimación de fallos [Describir]<br>• Estimar el número de fallos en una aplicación de software pequeña basándose en la densidad de fallos y la siembra de fallos [Estimar]<br>• Describir técnicas y problemas con las pruebas de software asíncrono, concurrente y paralelo [Describir]<br>• 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]<br>• Describir técnicas para la verificación y validación de artefactos que no son código [Describir] |
| **Readings :** [PM14] | |

| **Unit 9: Pruebas de Rendimiento y Evaluación Comparativa (3 hours)** | |
|---|---|
| **Competences Expected: AG-C03,AG-C09** | |
| **Topics** | **Learning Outcomes** |
| • Pruebas de rendimiento y evaluación comparativa<br>– Rendimiento (*throughput*) y latencia<br>– Degradación bajo carga (pruebas de estrés, manejo FIFO vs LIFO de solicitudes)<br>– Aceleración (*speedup*) y escalado (*scaling*)<br>∗ Ley de Amdahl<br>∗ Ley de Gustafson<br>∗ Escalado suave (*soft*) y débil (*weak*)<br>– Identificar y medir figuras de mérito (*figures of merit*)<br>– Cuellos de botella de rendimiento comunes<br>∗ Limitado por cómputo (*compute-bound*)<br>∗ Limitado por ancho de banda de memoria<br>∗ Limitado por latencia<br>– Métodos estadísticos y mejores prácticas para la evaluación comparativa (benchmarking)<br>∗ Estimación de la incertidumbre<br>∗ Intervalos de confianza<br>– Análisis y presentación (gráficos, etc.)<br>– Técnicas de medición de tiempo (*timing*) | • Describir el rendimiento (*throughput*) y la latencia y proporcionar ejemplos de cada uno [Describir]<br>• Explicar la aceleración (speedup) y las diferentes formas de escalado (scaling) y cómo se calculan [Explicar]<br>• Describir cuellos de botella de rendimiento comunes [Describir]<br>• Describir métodos estadísticos y mejores prácticas para la evaluación comparativa (benchmarking) de software [Describir]<br>• Explicar técnicas y desafíos para medir el tiempo al construir una evaluación comparativa (benchmark) [Explicar]<br>• 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] |
| **Readings :** [Bon15], [Gre20] | |

| **Unit 10: Conceptos de Confiabilidad (5 hours)** | |
|---|---|
| **Competences Expected: AG-C03,AG-C09** | |
| **Topics** | **Learning Outcomes** |
| • Concepto de confiabilidad como probabilidad de fallo o tiempo medio entre fallos, y fallas (faults) como causa de fallos (failures)<br>• Identificar requisitos de confiabilidad para diferentes tipos de software<br>• 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.)<br>• Confiabilidad del software, confiabilidad del sistema y comportamiento de fallos<br>• Ciclo de inyección y eliminación de defectos, y diferentes enfoques para la eliminación de defectos<br>• 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]<br>• Explicar los problemas que existen para lograr niveles muy altos de confiabilidad [Explicar]<br>• Comprender enfoques para minimizar fallas que pueden aplicarse en cada etapa del ciclo de vida del software [Explicar] |
| **Readings :** [PM14] | |

| **Unit 11: Métodos Formales (5 hours)** | |
|---|---|
| **Competences Expected: AG-C03,AG-C09** | |
| **Topics** | **Learning Outcomes** |
| • Especificación formal de interfaces<br>– Especificación de pre y post condiciones<br>– Lenguajes formales para escribir y analizar pre y post condiciones.<br>• áreas de problemas bien atendidas por los métodos formales<br>– Programación libre de bloqueos (*lock-free*), condiciones de carrera de datos<br>– Sistemas asíncronos y distribuidos, bloqueos (*deadlock*), bloqueos vivos (*livelock*), etc.<br>• Comparación con otras herramientas y técnicas para la detección de defectos<br>– Pruebas<br>– *Fuzzing*<br>• Enfoques formales para el modelado y análisis de software<br>– Verificadores de modelos (model checkers)<br>– Buscadores de modelos (model finders) | • Describir el papel que las técnicas de especificación y análisis formal pueden desempeñar en el desarrollo de software complejo y comparar su uso como técnicas de validación y verificación con las pruebas [Describir]<br>• Aplicar técnicas de especificación y análisis formal a diseños y programas de software de baja complejidad [Aplicar]<br>• Explicar los beneficios y desventajas potenciales del uso de lenguajes de especificación formal [Explicar] |
| **Readings :** [Som15] | |

## 8. WORKPLAN

### 8.1 Methodology

Individual and team participation is encouraged to present their ideas, motivating them with additional points in the different stages of the course evaluation.

### 8.2 Theory Sessions

The theory sessions are held in master classes with activities including active learning and roleplay to allow students to internalize the concepts.

### 8.3 Practical Sessions

The practical sessions are held in class where a series of exercises and/or practical concepts are developed through problem solving, problem solving, specific exercises and/or in application contexts.

## 9. EVALUATION SYSTEM

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

## 10. BASIC BIBLIOGRAPHY


[PM14] Roger S. Pressman and Bruce Maxim. *Software Engineering: A Practitioner's Approach*. 8th. McGraw-Hill, 2014.

[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.

[Fow18] Martin Fowler. *Refactoring: Improving the Design of Existing Code*. 2nd. Addison-Wesley Professional, 2018.

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