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

1. COURSE

CS391. Software Engineering III (Elective)

2. GENERAL INFORMATION

2.1 Course	:	CS391. Software Engineering III
2.2 Semester	:	8^{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	:	CS292. Software Engineering II. (7^{th} Sem)

3. PROFESSORS

Meetings after coordination with the professor

4. INTRODUCTION TO THE COURSE

Software Engineering III focuses on advanced topics of the software lifecycle, emphasizing software quality, evolution, and deployment in modern architectures. This course prepares students to handle the complexities of maintaining systems, ensuring software quality through rigorous validation, and the application of professional tools and design practices for resilient and maintainable software products.

5. GOALS

- Implement advanced quality assurance and reliability metrics.
- Understand software evolution, refactoring, and technical debt management.
- Integrate development and operations tools in collaborative environments.
- Design cloud-native applications using advanced software architectures.
- Analyze and apply systematic testing and validation strategies.

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: Conceptos de verificación y validación (12 hours)	
Competences Expected: AG-C03,AG-C09	
Topics	**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]
Readings : [Som15], [PM19]	

Unit 2: Herramientas y Análisis de Pruebas (3 hours)	
Competences Expected: AG-C03,AG-C09	
Topics	**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]
Readings : [PM19]	

Unit 3: Pruebas de Rendimiento y Evaluación Comparativa (3 hours)

Competences Expected: AG-C03,AG-C09

Topics	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]

Readings : [Bon15], [Gre20]

Unit 4: Compatibilidad y Versionado de API (12 hours)

Competences Expected: AG-C03,AG-C09

Topics	Learning Outcomes
• Ley de Hyrum / Ley de las Interfaces Implícitas • Compatibilidad con versiones anteriores – La compatibilidad no es una propiedad de una sola entidad, es una propiedad de una relación. – 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. • Control de versiones (versioning) – Versionado Semántico (SemVer) – 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] • Identificar cambios que pueden considerarse ampliamente ”compatibles con versiones anteriores”, potencialmente con declaraciones explícitas sobre qué uso está o no soportado [Analizar] • Evaluar si un cambio propuesto es lo suficientemente seguro dada la metodología de control de versiones en uso para un proyecto dado [Evaluar]

Readings : [Som15]

Unit 5: Control de Versiones y CI/CD (12 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 – 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. Fundamentos de Desarrollo de Software (SDF)-Practices,SEC-Coding • 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.	• 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]
Readings : [Kim+16]	

Unit 6: Principios y Arquitectura de Software (12 hours)	
Competences Expected: AG-C03,AG-C09	
Topics	**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]
Readings : [Fow17]	

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

[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.

[Kim+16] Gene Kim et al. *The DevOps Handbook*. IT Revolution Press, 2016.

[Fow17] Martin Fowler. *Refactoring: Improving the Design of Existing Code*. 2nd. Addison-Wesley, 2017.

[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.