National University of Engineering (UNI)
School of Computer Science
Syllabus 2024-II

1. COURSE

CS291. Software Engineering I (Mandatory)

2. GENERAL INFORMATION

2.1 Course	:	CS291. Software Engineering I
2.2 Semester	:	5^{th} Semester.
2.3 Credits	:	4
2.4 Horas	:	2 HT; 4 HP;
2.5 Duration of the period	:	16 weeks
2.6 Type of course	:	Mandatory
2.7 Learning modality	:	Face to face
2.8 Prerrequisites	:	• CS113. Computer Science II. (3^{rd} Sem) • CS271. Data Management. (4^{th} Sem)

3. PROFESSORS

Meetings after coordination with the professor

4. INTRODUCTION TO THE COURSE

The aim of developing software, except for extremely simple applications, requires the execution of a well-defined development process. Professionals in this area require a high degree of knowledge of the different models and development process, so that they are able to choose the most suitable for each development project. On the other hand, the development of medium and large-scale systems requires the use of pattern and component libraries and the mastery of techniques related to component-based design

5. GOALS

- Provide the student with a theoretical and practical framework for the development of software under quality standards.
- Familiarize the student with the software modeling and construction processes through the use of CASE tools.
- Students should be able to select architectures and ad-hoc technology platforms for deployment scenarios
- Applying component-based modeling to ensure variables such as quality, cost, and time-to-market in development processes.
- Provide students with best practices for software verification and validation.

6. COMPETENCES

1) Analyze a complex computing problem and apply principles of computing and other relevant disciplines to identify solutions. (Usage)

2) Design, implement, and evaluate a computing-based solution to meet a given set of computing requirements in the context of the program's discipline. (Usage)

3) Communicate effectively in a variety of professional contexts.. (Usage)

6) Apply computer science theory and software development fundamentals to produce computing-based solutions. (Assessment)

7. TOPICS

Unit 1: Ingeniería de Requisitos (18 hours)

Competences Expected:

Topics	Learning Outcomes
• Al describir los requisitos funcionales utilizando, por ejemplo, los casos de uso o historias de los usuarios. • Propiedades de requisitos, incluyendo la consistencia, validez, integridad y viabilidad. • Requisitos de software elicitatión. • Descripción de datos del sistema utilizando, por ejemplo, los diagramas de clases o diagramas entidad-relación. • Requisitos no funcionales y su relación con la calidad del software. • Evaluación y uso de especificaciones de requisitos. • Requisitos de las técnicas de modelado de análisis. • La aceptabilidad de las consideraciones de certeza/incertidumbre sobre el comportamiento del software/sistema. • Prototipos. • Conceptos básicos de la especificación formal de requisitos. • Especificación de requisitos. • Validación de requisitos. • Rastreo de requisitos.	• Enumerar los componentes clave de un caso de uso o una descripción similar de algún comportamiento que es requerido para un sistema [Evaluar] • Describir cómo el proceso de ingeniería de requisitos apoya la obtención y validación de los requisitos de comportamiento [Evaluar] • Interpretar un modelo de requisitos dada por un sistema de software simple [Evaluar] • Describir los retos fundamentales y técnicas comunes que se utilizan para la obtención de requisitos [Evaluar] • Enumerar los componentes clave de un modelo de datos (por ejemplo, diagramas de clases o diagramas ER) [Evaluar] • Identificar los requisitos funcionales y no funcionales en una especificación de requisitos dada por un sistema de software [Evaluar] • Realizar una revisión de un conjunto de requisitos de software para determinar la calidad de los requisitos con respecto a las características de los buenos requisitos [Evaluar] • Aplicar elementos clave y métodos comunes para la obtención y el análisis para producir un conjunto de requisitos de software para un sistema de software de tamaño medio [Evaluar] • Comparar los métodos ágiles y el dirigido por planes para la especificación y validación de requisitos y describir los beneficios y riesgos asociados con cada uno [Evaluar] • Usar un método común, no formal para modelar y especificar los requisitos para un sistema de software de tamaño medio [Evaluar] • Traducir al lenguaje natural una especificación de requisitos de software (por ejemplo, un contrato de componentes de software) escrito en un lenguaje de especificación formal [Evaluar] • Crear un prototipo de un sistema de software para reducir el riesgo en los requisitos [Evaluar] • Diferenciar entre el rastreo (*tracing*) hacia adelante y hacia atrás y explicar su papel en el proceso de validación de requisitos [Evaluar]

Readings : [ES14], [HF03]

Unit 2: Diseño de Software (18 hours)	
Competences Expected:	
Topics	**Learning Outcomes**
• Principios de diseño del sistema: niveles de abstracción (diseño arquitectónico y el diseño detallado), separación de intereses, ocultamiento de información, de acoplamiento y de cohesión, de reutilización de estructuras estándar. • Diseño de paradigmas tales como diseño estructurado (descomposición funcional de arriba hacia abajo), el análisis orientado a objetos y diseño, orientado a eventos de diseño, diseño de nivel de componente, centrado datos estructurada, orientada a aspectos, orientado a la función, orientado al servicio. • Modelos estructurales y de comportamiento de los diseños de software. • Diseño de patrones. • Relaciones entre los requisitos y diseños: La transformación de modelos, el diseño de los contratos, invariantes. • Conceptos de arquitectura de software y arquitecturas estándar (por ejemplo, cliente-servidor, n-capas, transforman centrados, tubos y filtros). • El uso de componentes de diseño: seleccion de componentes,diseño,adaptacion y componentes de ensamblaje, componentes y patrones, componentes y objetos(por ejemplo,construir una GUI usando un standar widget set) • Calidad del diseño interno, y modelos para: eficiencia y desempeño, redundancia y tolerancia a fallos, trazavilidad de los requerimientos. • Calidad del diseño interno, y modelos para: eficiencia y desempeño, redundancia y tolerancia a fallos, trazavilidad de los requerimientos. • Medición y análisis de la calidad de un diseño. • Compensasiones entre diferentes aspectos de la calidad. • Aaplicaciones en frameworks. • Middleware: El paradigma de la orientacion a objetos con middleware, requerimientos para correr y clasificar objetos, monitores de procesamiento de transacciones y el sistema de flujo de trabajo. • Principales diseños de seguridad y codificación(cross-reference IAS/Principles of securre design). – Principio de privilegios mínimos – Principio de falla segura por defecto – Principio de aceptabilidad psicológica	• Formular los principios de diseño, incluyendo la separación de problemas, ocultación de información, acoplamiento y cohesión, y la encapsulación [Familiarizarse] • Usar un paradigma de diseño para diseñar un sistema de software básico y explicar cómo los principios de diseño del sistema se han aplicado en este diseño [Usar] • Construir modelos del diseño de un sistema de software simple los cuales son apropiado para el paradigma utilizado para diseñarlo [Usar] • En el contexto de un paradigma de diseño simple, describir uno o más patrones de diseño que podrían ser aplicables al diseño de un sistema de software simple [Familiarizarse] • Para un sistema simple adecuado para una situación dada, discutir y seleccionar un paradigma de diseño apropiado [Usar] • Crear modelos apropiados para la estructura y el comportamiento de los productos de software desde la especificaciones de requisitos [Usar] • Explicar las relaciones entre los requisitos para un producto de software y su diseño, utilizando los modelos apropiados [Evaluar] • Para el diseño de un sistema de software simple dentro del contexto de un único paradigma de diseño, describir la arquitectura de software de ese sistema [Familiarizarse] • Dado un diseño de alto nivel, identificar la arquitectura de software mediante la diferenciación entre las arquitecturas comunes de software, tales como 3 capas (*3-tier*), *pipe-and-filter*, y cliente-servidor [Familiarizarse] • Investigar el impacto de la selección arquitecturas de software en el diseño de un sistema simple [Evaluar] • Aplicar ejemplos simples de patrones en un diseño de software [Usar] • Describir una manera de refactorar y discutir cuando esto debe ser aplicado [Familiarizarse] • Seleccionar componentes adecuados para el uso en un diseño de un producto de software [Usar] • Explicar cómo los componentes deben ser adaptados para ser usados en el diseño de un producto de software [Familiarizarse] • Diseñar un contrato para un típico componente de software pequeño para el uso de un dado sistema [Usar] • Discutir y seleccionar la arquitectura de software

Unit 3: Construcción de Software (24 hours)	
Competences Expected:	
Topics	**Learning Outcomes**
• Prácticas de codificación: técnicas, idiomas/patrones, mecanismos para construcción de programas de calidad: – Prácticas de codificación defensive – Prácticas de codificación segura – Utilizando mecanismos de manejo de excepciones para hacer el programa más robusto, tolerante a fallas • Normas de codificación. • Estrategias de integración. • Desarrollando contexto: ""campo verde"" frente a la base de código existente : – Análisis de cambio impacto – Cambio de actualización • Los problemas de seguridad potenciales en los programas : – Buffer y otros tipos de desbordamientos – Condiciones elemento Race – Inicialización incorrecta, incluyendo la elección de los privilegios – Entrada Comprobación – Suponiendo éxito y corrección – La validación de las hipótesis	• Describir técnicas, lenguajes de codificación y mecanismos de implementación para conseguir las propiedades deseadas, tales como la confiabilidad, la eficiencia y la robustez [Evaluar] • Construir código robusto utilizando los mecanismos de manejo de excepciones [Evaluar] • Describir la codificación segura y prácticas de codificación de defensa [Evaluar] • Seleccionar y utilizar un estándar de codificación definido en un pequeño proyecto de software [Evaluar] • Comparar y contrastar las estrategias de integración incluyendo: de arriba hacia abajo (*top-down*), de abajo hacia arriba (*bottom-up*), y la integración Sándwich [Evaluar] • Describir el proceso de analizar e implementar los cambios a la base de código desarrollado para un proyecto específico [Evaluar] • Describir el proceso de analizar e implementar los cambios a la base de código desarrollado para un proyecto específico [Evaluar] • Reescribir un programa sencillo para eliminar vulnerabilidades comunes, tales como desbordamientos de búffer, desbordamientos de enteros y condiciones de carrera [Evaluar] • Escribir un componente de software que realiza alguna tarea no trivial y es resistente a errores en la entrada y en tiempo de ejecución [Evaluar]
Readings : [ES14], [HF03]	

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

[HF03] Brian Lyons Hans-Erik Eriksson Magnus Penker and Davis Fado. *UML 2 Toolkit*. 2nd. Wiley, Oct. 2003.

[ES14] Bert Bates Eric Freeman Elisabeth Robson and Kathy Sierra. *Head First Design Patterns*. 2nd. O'Reilly Media, Inc, July 2014.