Universidad Católica San Pablo (UCSP)
Escuela Profesional de
Ciencia de la Computación
SILABO

Universidad Católica San Pablo
2024-II

CS291. Ingeniería de Software I (Obligatorio)

1. Información general

1.1 Escuela	:	Ciencia de la Computación
1.2 Curso	:	CS291. Ingeniería de Software I
1.3 Semestre	:	5^{to} Semestre
1.4 Prerrequisitos	:	CS113. Ciencia de la Computación II. (3^{er} Sem) CS271. Bases de Datos I. (4to Sem)
1.5 Condición	:	Obligatorio
1.6 Modalidad de aprendizaje	:	Presencial
1.7 Horas	:	2 HT; 4 HP;
1.8 Créditos	:	4
1.9 Plan	:	Plan Curricular 2016

2. Profesores

Titular

- Gustavo Delgado Ugarte <ggdelgado@ucsp.edu.pe>
  - Máster en Ingeniería del Software, Escuela Universitaria de Ingeniería Industrial, Informática y Sistemas - UTA, Chile, 2009..

3. Fundamentación del curso

La tarea de desarrollar software, excepto para aplicaciones sumamente simples, exige la ejecución de un proceso de desarrollo bien definido. Los profesionales de esta área requieren un alto grado de conocimiento de los diferentes modelos de proceso de desarrollo, para que sean capaces de elegir el más idóneo para cada proyecto de desarrollo. Por otro lado, el desarrollo de sistemas de mediana y gran escala requiere del uso de bibliotecas de patrones y componentes y del dominio de técnicas relacionadas al diseño basado en componentes.

4. **Resumen**

1. Ingeniería de Requisitos 2. Diseño de Software 3. Construcción de Software

5. Objetivos Generales

- Brindar al alumno un marco teórico y práctico para el desarrollo de software bajo estándares de calidad.
- Familiarizar al alumno con los procesos de modelado y construcción de software a través del uso de herramientas CASE.
- Los alumnos deben ser capaces de seleccionar Arquitecturas y Plataformas tecnológicas ad-hoc a los escenarios de implementación.
- Aplicar el modelado basado en componentes con el fin de asegurar variables como calidad, costo y time-to-market en los procesos de desarrollo.
- Brindar a los alumnos mejores prácticas para la verificación y validación del software.

6. Contribución a los resultados (Outcomes)

Esta disciplina contribuye al logro de los siguientes resultados de la carrera:

1) S.O. Analizar un problema computacional complejo y aplicar los principios computacionales y otras disciplinas relevantes para identificar soluciones. (**Usar**)

2) S.O. Diseñar, implementar y evaluar una solución basada en computación para cumplir con un conjunto determinado de requisitos computacionales en el contexto de las disciplinas del programa. (**Usar**)

3) S.O. Comunicarse efectivamente en diversos contextos profesionales. (**Usar**)

6) S.O. Aplicar la teor´ıa de la computación y los fundamentos del desarrollo de software para producir soluciones basadas en computación. (**Evaluar**)

7. Unidades

UNIDAD 1: Ingeniería de Requisitos	
Resultados del estudiante: 1,6,2,3	
Contenido	**Objetivos Generales**
• 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 elicitación. • 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]
Lecturas: Pressman (2005), Sommerville (2008), Larman (2008)	

UNIDAD 2 : Diseño de Software	
Resultados del estudiante: 6	
Contenido	**Objetivos Generales**
• 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.	• 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 apropiados 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]

• 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: selección de componentes, diseño, adaptación y componentes de ensamblaje, componentes y patrones, componentes y objetos (por ejemplo, construir una GUI usando un standard widget set). • Diseños de refactorización utilizando patrones de diseño. • Calidad del diseño interno, y modelos para: eficiencia y desempeño, redundancia y tolerancia a fallos, trazabilidad de los requerimientos. • Medición y análisis de la calidad de un diseño. • Compensaciones entre diferentes aspectos de la calidad. • Aplicaciones en frameworks. • Middleware: El paradigma de la orientación 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 secure design). - Principio de privilegios mínimos - Principio de falla segura por defecto - Principio de aceptabilidad psicológica	• 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 de 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 refactorizar y discutir cuándo 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]
Lecturas: Pressman (2005), Sommerville (2008), Larman (2008)	

UNIDAD 3: Construcción de Software	
Resultados del estudiante: 1,6,2,3	
Contenido	**Objetivos Generales**
• Prácticas de codificación: técnicas, idiomas/patrones, mecanismos para construcción de programas de calidad: - Prácticas de codificación defensiva - 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 de carrera - Inicialización incorrecta, incluyendo la elección de los privilegios - Comprobación de entrada - 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 defensiva [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 una gran base de código existente [Evaluar] • Reescribir un programa sencillo para eliminar vulnerabilidades comunes, tales como desbordamientos de búfer, 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]
Lecturas: Pressman (2005), Sommerville (2008), Larman (2008)	

8. Metodología

1. El profesor del curso presentará clases teóricas de los temas señalados en el programa propiciando la intervención de los alumnos.

2. El profesor del curso presentará demostraciones para fundamentar clases teóricas.

3. El profesor y los alumnos realizarán prácticas.

4. Los alumnos deberán asistir a clase habiendo leído lo que el profesor va a presentar. De esta manera se facilitará la comprensión y los estudiantes estarán en mejores condiciones de hacer consultas en clase.

9. Evaluar

- **Evaluación Permanente 1:** 20%
- **Examen parcial:** 30%
- **Evaluación Permanente 2:** 20%
- **Examen final:** 30%

Referencias

- Larman, Craig (2008). *Applying UML and Patterns*. Prentice Hall.
- Pressman, Roger S. (Mar. 2005). *Software Engineering: A Practitioner's Approach*. 6th. McGraw-Hill.
- Sommerville, Ian (May 2008). *Software Engineering*. 7th. ISBN: 0321210263. Addison Wesley.