Planificación de

Ingeniería del Software II

FACULTAD DE CIENCIAS EXACTAS, INGENIERIA Y AGRIMENSURA

Código/s: R-421

Identificación y características del Espacio Curricular

Carrera/s:	Licenciatura en Ciencias de la Computación		
Plan de Estudios:	2010, TO2024	Carácter:	Obligatoria
Bloque/Campo:		Área:	Ingeniería de Software, Bases de Datos y Sistemas de Información
Régimen de cursado:	Cuatrimestral		
Cuatrimestre:	8º [LCC], 8º [LCC]		
Carga horaria:	135 hs. / 9 hs. semanales	Formato curricular:	Asignatura
Escuela:	Ciencias Exactas y Naturales	Departamento:	Ciencias de la Computación
Docente responsable:	CRISTIA, Maximiliano		

Programa Sintético

El impacto del diseño y arquitectura de software en el desarrollo de sistemas de software: desarrollo, mantenimiento, reingeniería, costos. Diseño de software: Diseño Basado en Ocultación de Información, Diseño Basado en Tipos Abstractos de Datos, Diseño Orientado a Objetos, Documentación de Diseño. Patrones de diseño. Estilos Arquitectónicos. Diseño centrado en el usuario. Testing de software: testing estructural, testing funcional basado en especificaciones formales.

Espacios Curriculares Relacionados

Previos Aprobados:	R-411 - Ingeniería del Software I
Simultaneos Recomendados:	R-422 - Compiladores, R-423 - Complementos de Matemática II
Posteriores:	

Vigencia desde 2024

Firma Profesor	Fecha	Firma Aprob. Escuela	Fecha

Con el aval del Consejo Asesor:

Fundamentación

Una de las características más importantes de los sistemas de software es la posibilidad de modificarlos sin costos aparentes. Sin embargo, modificar un sistema de software de manera que alcance los objetivos esperados no es fácil ni barato. Uno de los principios aceptados por la comunidad académica de la Ingeniería de Software, indica que el costo de modificación de un sistema de software está en relación directa con la calidad y propiedades de su diseño y arquitectura. Por tal motivo en esta asignatura se muestra a los estudiantes los conceptos, técnicas y herramientas que permiten generar buenos diseños de software. A su vez, se sabe que no existe una única forma de diseñar software que sea aplicable en todos los casos. Por lo tanto, el curso tratan diferentes técnicas de diseño y arquitectura de software que le permite a los estudiantes conocer las mejores técnicas a aplicar en distintas situaciones. Además, se introducen las nociones básicas de validación y verificación de software, haciendo énfasis en el testing funcional basado en especificaciones formales. La asignatura incluye la realización de un proyecto de desarrollo usando métodos formales que se inicia durante el cursado de Ingeniería de Software I. Con este fin los alumnos seleccionan un requerimiento funcional de baja complejidad, escriben una especificación formal de aquel, traducen la especificación formal a un lenguaje de implementación de alto nivel y realizan un par de actividades de verificación. La asignatura se relaciona con todos los cursos dedicados a programación dado que la idea de diseño de un programa aplica independientemente del lenguaje de programación con el que se trabaje.

Resultados del aprendizaje

Al finalizar el cursado los/las estudiantes serán capaces de:

RA1 Diseñar sistemas de software de complejidad reducida aplicando el principio de ocultación de la información, tipos abstractos de datos y diseño orientado a objetos
RA2 Documentar el diseño de sistemas de software de complejidad reducida
RA3 Aplicar patrones de diseño al diseño de sistemas de software de complejidad reducida
RA4 Aplicar estilos arquitectónicos al diseño de sistemas de software de complejidad reducida
RA5 Generar casos de pruebas para programas de complejidad reducida utilizando criterios de testing estructural
RA6 Generar casos de pruebas para programas de complejidad reducida utilizando criterios de testing basado en especificaciones formales
RA7 Implementar un proyecto de desarrollo de software de baja complejidad basado en el uso de técnicas formales de desarrollo

Competencias / Ejes transversales y Resultados del Aprendizaje

Competencia/Eje transversal al que tributa	Nivel	Resultados del Aprendizaje
CGT1-Identificación, formulación y resolución de problemas de informática	Bajo	RA1, RA2, RA3, RA4, RA5, RA6
CGT2-Concepción, diseño y desarrollo de proyectos de informática	Alto	RA1, RA2, RA3, RA4, RA5, RA6
CGT3-Gestión, planificación, ejecución y control de proyectos de informática	Alto	RA1, RA2, RA3, RA4, RA5, RA6
CGT4-Utilización de técnicas y herramientas de aplicación en la informática	Bajo	RA1, RA2, RA3, RA4, RA5, RA6
CGS2-Fundamentos para la comunicación efectiva	Medio	RA7
CGS4-Fundamentos para la evaluación y actuación en relación con el impacto social de su actividad en el contexto global y local	Medio	RA7

Programa Analítico

1. Unidad I: Diseño de software
1.1. Una teoría para el diseño de software
1.1.1. El diseño como fase del ciclo de desarrollo del sistema.
1.1.2. Diferencias entre modelo funcional y diseño.
1.1.3. Relación entre el modelo funcional y el diseño.
1.1.4. Relación entre los métodos formales y el diseño de software.
1.2. Problemas con el diseño funcional o estructurado
1.3. Diseño basado en ocultación de la información (DBOI)
1.3.1. Diseño para el cambio; la metodología de David L. Parnas; ítem con probabilidad de cambio; componentes, interfaz e implementación; abstracción y encapsulamiento.
1.3.2. Un ejemplo de DBOI; documentación del diseño; lenguaje 2MIL; limitaciones del DBOI.
1.4. Diseño Basado en Tipos Abstractos de Datos (DTAD)
1.5. Diseño Orientado a Objetos (DOO)
1.5.1. El concepto de herencia
1.6. Documentación de diseño
1.6.1. Ítem con probabilidad de cambio; interfaces; estructura de módulos; guía de módulos; estructura de uso; estructura de procesos; estructura de objetos; estructura de herencia.
1.7. Reingeniería de software
1.7.1. Reconstrucción del diseño a partir del código fuente; modificación del diseño para incorporar metodologías más modernas; cambios a la implementación para adecuarla al nuevo diseño.
1.8. Comentarios sobre diseño de software concurrente y de tiempo real
1.9. Comentarios sobre diseño centrado en el usuario
2. Unidad II: Patrones de diseño
2.1. Introducción
2.1.1. Herencia de clase y herencia de interfaces; herencia y composición de objetos.
2.2. Estudio y aplicación de algunos patrones de diseño
2.2.1. Composite; Abstract Factory; Command; Iterator; Visitor; Bridge; Decorator; Strategy.
3. Unidad III: Arquitecturas de software
3.1. Vocabulario, conceptos y problemas
3.1.1. Diferencias entre arquitectura y diseño; componentes y conectores; estilos arquitectónicos.
3.2. Estilos arquitectónicos
3.2.1. Un catálogo incompleto de estilo arquitectónicos.
3.2.2. Invocación Implícita.
3.2.3. Tubos y filtros.
3.2.4. Sistemas Estratificados.
3.2.5. Control de procesos
3.2.6. Blackboard Systems.
3.2.7. Cliente/Servidor de Tres Capas.
4. Unidad IV: Testing de software
4.1. Introducción al testing de software
4.1.1. Verificación y Validación (V&V); definición de testing y vocabulario básico; el proceso de testing; testing de distintos aspectos de un software; las dos metodologías clásicas de testing.
4.2. Testing estructural basado en flujo de control
4.2.1. Grafo de flujo de control de un programa; criterios de testing estructural basado en flujo de control: cubrimiento de sentencias, cubrimiento de flechas, cubrimiento de condiciones, criterio de valores de verdad, criterio de condiciones múltiples.
4.3. Testing basado en especificaciones Z
4.3.1. Introducción: programas y especificaciones, casos de prueba exitosos, espacio válido de entrada, funciones de refinamiento y abstracción, el proceso de testing basado en especificaciones formales.
4.3.2. Tácticas de testing funcional: formal normal disyuntiva, particiones estándar, propagación de

subdominios, mutación de especificaciones, causa-efecto, otras tácticas.

Modalidades de enseñanza

El docente a cargo de la asignatura dicta las clases teóricas mientras que el resto de los docentes se dedican a las clases prácticas. En las clases teóricas se exponen los temas usando un problema disparador para cada tema. Por ejemplo para introducir el principio conocido como “diseño para el cambio” se plantea el problema estructura el
código de una caja de ahorros a la cual hay que agregarle un requerimiento que modifica levemente su funcionamiento; se muestra un diseño típico basado en la intuición de los desarrolladores y luego se muestra un diseño basado en el principio de diseño para el cambio. Las clases de teoría son expositivas; el recurso áulico usado es el pizarrón.
En las clases prácticas se resuelven problemas de la misma índole que los problemas disparadores vistos en clase. Las cláses prácticas consisten en la exposición parcial del diseño o arquitectura de un sistema de software de manera tal que los alumnos luego la puedan completar; más adelante los alumnos deben intentar resolver por sí solos otros problemas similares para lo cual cuentan con la asistencia de los docentes. Los exámenes, sean parciales como finales, consisten esencialmente en resolver problemas similares a los vistos en teoría y práctica. Fuera de los horarios de clase los alumnos continuan desarrollando el trabajo práctico/proyecto individual iniciado en Ingeniería de Software I.

Recursos

Aula de clase; ocasionalmente proyector multimedia, software específico relacionado con los contenidos. El curso cuenta con un sitio web (www.fceia.unr.edu.ar/is2), además se usa el sitio de Comunidades de la UNR y existe un canal YouTube gestionado por el docente responsable (www.youtube.com/c/maximilianocristiais).

Actividades de Formación Práctica

Nº	Título	Descripción
1	Práctica 1	Problemas de diseño de software En cada problema se le presenta al alumno una descripción informal de un sistema de software y se le solicita hacer un diseño de software.
2	Práctica 2	Problemas de aplicación de patrones de diseño de software En cada problema se le presenta al alumno una descripción informal de un sistema de software y se le solicita aplicar uno o más patrones de diseño.
3	Práctica 3	Problemas de arquitectura de software En cada problema se le presenta al alumno una descripción informal de un sistema de software y se le solicita aplicar un estilo arquitectónico.
4	Práctica 4	Problemas sobre testing de software En cada problema se le presenta al alumno una descripción informal de un sistema de software y se le solicita aplicar uno o más criterios de testing de software.

5	Segunda parte del Proyecto de Especificación y Verificación (PEV)	En esta etapa el PEV consiste en aplicar testing basado en especificaciones a la especificación desarrollada en la primera parte utilizando la herramienta Fastest. La descripción del proyecto se encuentra aquí: https://www.fceia.unr.edu.ar/ingsoft/tp.html. Durante la realización del proyecto los alumnos cuentan con la asistencia del responsable de la asignatura.

Evaluación

Regularización-Promoción. Al finalizar el curso el alumno alcanzará una de las tres condiciones: Libre, Regular o Promovido. Cada una de estas condiciones está en relación con las notas que haya obtenido en las evaluaciones parciales.
Parciales. Se toman dos parciales y un recuperatorio durante el cuatrimestre. El primer parcial cubre la Unidad 1 mientras que el segundo cubre la Unidad 2. Para obtener la condición de Regular es necesario haber resuelto de manera satisfactoria al menos el 65% de cada parcial. Para obtener la condición de Promovido es necesario haber resuelto de manera satisfactoria al menos el 80% de cada parcial. La condición de Promovido dura hasta que comienza a dictarse nuevamente el curso. El recuperatorio, que se toma al finalizar el cuatrimestre, puede utilizarse para mejorar la nota de uno de los parciales, tanto sea para obtener la condición de Regular como la de Promovido. En caso de concurrir al recuperatorio el alumno perderá la nota obtenida en el parcial que está recuperando. La condición de Libre se obtiene si no se cumplen los requisitos para obtener las otras dos condiciones.
Finales. El alumno que haya obtenido la condición de Promovido solo deberá rendir, en el examen final, la Unidades 3 y 4 más los temas correspondientes al parcial que no haya logrado promover, si lo hubiere. El alumno que haya obtenido la condición de Regular deberá rendir un examen final en el que se incluyen problemas prácticos que cubren la la totalidad del curso. El alumno que haya quedado en condición de Libre deberá rendir un examen final en el que se incluyen contenidos de todo el curso.
Todos los exámenes (parciales y finales) consisten en resolver problemas prácticos. La única excepción es el examen final de un alumno libre el cual incluye, además, la evaluación de la teoría.
Trabajo Práctico. El Trabajo Práctico consiste en desarrollar el PEV mencionado más arriba. El Trabajo Práctico se comienza a realizar durante el cursado de Ingeniería de Software 1, se completa durante el cursado de esta asignatura y el alumno lo debe entregar el día que se presenta a rendir el examen final.

Resultado de Aprendizaje	Actividades/Modalidad de Enseñanza	Modalidad de Evaluación
RA1	Clases teóricas y clases prácticas	Resolución de problemas
RA2	Clases teóricas y clases prácticas	Resolución de problemas
RA3	Clases teóricas y clases prácticas	Resolución de problemas
RA4	Clases teóricas y clases prácticas	Resolución de problemas
RA5	Clases teóricas y clases prácticas	Resolución de problemas
RA6	Clases teóricas y clases prácticas	Resolución de problemas
RA7	Guía por parte de los docentes para la selección de bibliografía y otros materiales	Presentación del proyecto en forma oral y escrita

Bibliografía básica

Autores (Apellido, Inicial nombre)	Año de edición	Título de la obra	Editorial o Revista	Ejemplares disponibles o sitio web

Gamma, E., Helm, R., Johnson, R., Vlissides, J.	2003	Patrones de diseño	Addison-Wesley	1
Bass, L., Clements, P., Kazman, R.	2003	Software Architecture in Practice	Addison-Wesley	1
Buschmann, F., Meunier, R., Rohnert, H., Sommerland, P., Stal, M.	2004	Pattern-Oriented Software Architecture. A system of Patterns	John Wiley & Sons	2
Shaw, M., Garlan, D.	1996	Software Architecture: Perspectives on an Emergin Discipline	Prentice Hall	1
Garlan, D., Kaiser, G., Notkin, D.	1982	Using tool abstraction to compose systems	IEEE Computer, 25(6):30-38	https://www.researchgate.net/publication/2954124_Using_tool_abstraction_to_compose_system
Parnas, D.L.	1972	On the criteria to be used in decomposing systems into modules	Communications of the ACM	Poner: https://www.researchgate.net/publication/200085877_On_the_Criteria_To_Be_Used_in_Decomposing_Systems_into_Modules
Parnas, D.L.	1979	Designing software for ease of extension and contraction	IEEE Transactions on Software Engineering	Poner: https://www.researchgate.net/publication/3189208_Designing_Software_for_Ease_of_Extension_and_Contraction
Cristiá, M.	2015	Una Teoría para el Diseño de Software	Apunte de clase	https://www.fceia.unr.edu.ar/ingsoft/intro-diseno.pdf
Cristiá, M.	2015	Diseño de Software	Apunte de clase	https://www.fceia.unr.edu.ar/ingsoft/diseno-a.pdf
Cristiá, M.	2019	Problemas Resueltos – Diseño	Apunte de clase	https://www.fceia.unr.edu.ar/ingsoft/problemasDiseno.pdf
Cristiá, M.	2015	Estándar para documentar el uso de patrones de diseño en un diseño de software	Apunte de clase	https://www.fceia.unr.edu.ar/ingsoft/patrones-doc.pdf
Cristiá, M.	2019	Problemas Resueltos – Patrones de Diseño	Apunte de clase	https://www.fceia.unr.edu.ar/ingsoft/problemasPatrones.pdf
Cristiá, M.	2015	Catálogo Incompleto de Estilos Arquitectónicos	Apunte de clase	https://www.fceia.unr.edu.ar/ingsoft/estilos-cat.pdf

Cristiá, M.	2015	Introducción al Testing de Software	Apunte de clase	https://www.fceia.unr.edu.ar/ingsoft/testing-intro-a.pdf
Cristiá, M.	2015	Introducción al Testing de Estructural	Apunte de clase	https://www.fceia.unr.edu.ar/ingsoft/testing-estruct-a.pdf
Cristiá, M.	2015	Testing Basado en Especificaciones Z	Apunte de clase	https://www.fceia.unr.edu.ar/ingsoft/testing-func-a.pdf

Bibliografía complementaria

Autores (Apellido, Inicial nombre)	Año de edición	Título de la obra	Editorial o Revista	Ejemplares disponibles o sitio web
Rapps, S., Weyuker, E.J.	1982	Data flow analysis techniques for test data selection	Proceedings of IEEE International Conference on Software Engineering	Poner: https://www.semanticscholar.org/paper/Data-flow-analysis-techniques-for-test-data-Rapps-Weyuker/7b0afd0d32f4a41c5141a11d1093a456412ff198
Stocks, P., Carringtong, D.	1996	A framework for specification-based testing	IEEE Transactions on Software Engineering	https://www.researchgate.net/publication/3187841_A_framework_for_specification-based_testing
Ghezzi, C., Jazayeri, M., Mandrioli, D.	2003	Fundamentals of Software Engineering	Prentice Hall	3

Distribución de la carga horaria

Presenciales

Teóricas		54 Hs.
Prácticas	Formación Experimental	
	Resolución de Problemas vinculados a la Profesión	
	Resolución de Problemas y Ejercicios	35 Hs.
	Actividades de Proyecto y Diseño	40 Hs.
	Formación en la Práctica Profesional	
Evaluaciones		6 Hs.
	Total	**135 Hs.**

Dedicadas por el alumno fuera de clase

Preparación Teórica	30 Hs.
Preparación Práctica	30 Hs.
Elaboración y redacción de informes, trabajos, presentaciones, etc.	20 Hs.

Total 80 Hs.

Cronograma de actividades

Semana	Unidad	Tema	Actividad
1	1	Diseño de software: introducción, conceptos de encapsulamiento y abstracción	Clases de teoría
2	1	Diseño de software: aplicación de técnicas de diseño a problemas prácticos, TAD, DOO, modularización, principio de ocultación de la información	Clase de teoría y clase de práctica
3	1	Diseño de software: aplicación de técnicas de diseño a problemas prácticos, TAD, DOO, modularización, principio de ocultación de la información	Clase de teoría y clase de práctica
4	1	Diseño de software: documentación de diseño	Clase de teoría y clase de práctica
5	2	Patrones de diseño: Composite, Strategy	Clase de teoría y clase de práctica
6	2	Patrones de diseño: Decorator, Abstract Factory	Clase de teoría y clase de práctica
7	2	Patrones de diseño: Bridge, Command	Clase de teoría y clase de práctica
8	2	Patrones de diseño: Iterator, Visitor	Clase de teoría y clase de práctica
9	3	Estilos arquitectónicos: introducción a la arquitectura de software, estilos arquitectónicos	Clases de teoría y clases de práctica, primer parcial
10	3	Estilos arquitectónicos: invocación implícita, tubos y filtros	Clase de teoría y clase de práctica
11	3	Estilos arquitectónicos: sistemas estratificados, control de procesos	Clase de teoría y clase de práctica
12	3	Estilos arquitectónicos: blackboard systems, cliente/servidor de 3 capas	Clase de teoría y clase de práctica
13	4	Testing de software: introducción a la validación y verificación de software, testing de software, testing estructural	Clases de teoría y clases de práctica, segundo parcial
14	4	Testing de software: testing funcional basado en especificaciones formales	Clase de teoría y clase de práctica
15	4	Testing de software: testing funcional basado en especificaciones formales	Clases de teoría y clases de práctica, recuperatorio