UNIVERSIDAD NACIONAL DE RÍO CUARTO

FACULTAD DE CIENCIAS EXACTAS, FÍSICO-QUÍMICAS Y NATURALES

DEPARTAMENTO DE COMPUTACIÓN

CARRERA/S: Analista en Computación

PLAN DE ESTUDIOS: 1999

ASIGNATURA: Proyecto **CÓDIGO: 1998**

DOCENTE RESPONSABLE: Dr. Nazareno Aguirre

EQUIPO DOCENTE: Dr Nazareno Aguirre, A.C. Facundo Molina, Lic. Nicolás Ricci, Lic. Simón Gutiérrez Brida

AÑO ACADÉMICO: 2016

REGIMEN DE LA ASIGNATURA: Cuatrimestral

RÉGIMEN DE CORRELATIVIDADES:

Para cursar:

Aprobada	***Regular***
-	Análisis y Diseño de Sistemas (c. 3303)
-	Bases de Datos (c. 1959)

Para rendir

Aprobada
Análisis y Diseño de Sistemas (c. 3303)
Bases de Datos (c. 3303)

CARGA HORARIA TOTAL: 8 horas semanales

TEÓRICAS: 4 hs **PRÁCTICAS:** 4 hs **LABORATORIO:**

CARÁCTER DE LA ASIGNATURA: Obligatoria

A. CONTEXTUALIZACIÓN DE LA ASIGNATURA

Proyecto es asignatura obligatoria del plan de estudios de la carrera Analista en Computación. Esta materia consiste de un curso práctico, en el cual se simulan las condiciones de desarrollo de un proyecto real a nivel industrial, llevado a cabo de manera altamente colaborativa, y distribuida. El proyecto es desarrollado por equipos de estudiantes de nuestra Universidad.

La asignatura, siendo trabajo final de carrera, tiene el objetivo de aplicar de manera combinada los conocimientos adquiridos en diferentes asignaturas de la carrera, en la resolución de un proyecto de software concreto.

B. OBJETIVOS PROPUESTOS

El objetivo esencial de la materia es lograr que los alumnos se familiaricen con los principios y las prácticas que hacen a la ingeniería de software, basándose en sólidos (y modernos) conceptos de ingeniería de software y ciencias de la computación. La materia es esencialmente práctica, y uno de los objetivos de la misma es intentar reproducir lo más fielmente posible las condiciones en las cuales los desarrolladores se encuentran al encarar un proyecto de software (de mediana o gran escala).

Un objetivo particular de esta materia es conseguir integrar los conocimientos obtenidos por los alumnos en muchas materias anteriores, de variadas áreas (algoritmos, ingeniería de software, matemática discreta, etc.). Debido a las tecnologías a utilizar en la materia, se hará un fuerte énfasis en especificaciones de software en términos de pre- y post-condiciones, las nociones fundamentales de aserciones de corrección, invariantes de ciclos, etc. (en general, los temas ligados a construcción rigurosa de sistemas).

C. CONTENIDOS BÁSICOS DEL PROGRAMA A DESARROLLAR

Revisión de conceptos previos. Anotaciones en programas. La importancia de las especificaciones. Pre- y post-condición. Invariantes de ciclos. Funciones variantes. Análisis de corrección de programas. Análisis de terminación de programas.

Revisión de conceptos básicos de orientación a objetos e ingeniería de software. Modularidad. Reutilizabilidad. Tipos abstractos de datos. Las estructuras estática y dinámica: clases y objetos. Tipos básicos y de clase. Herencia y subtipado. Dispatch dinámico.

Soporte para la definición de contratos en lenguajes de programación modernos. Verificación de contratos en tiempo de ejecución. Violación de contratos y excepciones. El manejo de excepciones tradicional en lenguajes imperativos. Manejo de eventos. Programación de rutinas polimórficas en lenguajes orientados a objetos.

Programación basada en componentes. Componentes esenciales en lenguajes de programación modernos. Bibliotecas para entrada/salida. Bibliotecas para redes. Sockets. Sockets de redes y sockets Unix. Aplicaciones cliente-servidor. Bibliotecas

para persistencia. Manejo de archivos. Almacenamiento y recuperación de objetos. Interacción de aplicaciones con motores de bases de datos.

La importancia de la modularización en la construcción de software. Desarrollo de software orientado a objetos. Nociones fundamentales. Clases como definiciones de tipos y módulos. Unificación de métodos y atributos. Dualidades entre las interpretaciones de clases como módulos y tipos. Ejemplos. Relaciones entre clases. Herencia. La herencia como extensión de módulos. La herencia como definición de subtipos. Clases como clientes de otras clases. Especificación de las responsabilidades de un módulo. Invariantes de módulos.

Los contratos. Contratos como especificaciones de las responsabilidades de clases. Contratos para rutinas. Relaciones entre contratos y nociones de Ciencias de la Computación. Triplas de Hoare y verificación de programas. El significado de las pre- y post-condiciones. La interpretación de las pre-y post-condiciones por parte de los clientes de una clase. Contratos de clases en forma de invariantes. Contratos y herencia. El principio de subcontratación. Subcontratación e invariantes. Subcontratación con respecto a pre- y post-condiciones.

El proceso de desarrollo de software propuesto en combinación con contratos. La importancia de la reversibilidad. La generalización como un paso obligatorio en el proceso de desarrollo. Distintas etapas en el ciclo de vida. Especificación, diseño, implementación, validación y verificación. Reutilización.

Casos prácticos de aplicación de patrones de diseño y sus beneficios. Evolución de software, cambios en los requerimientos y su propagación en los diseños. Diseños resistentes a cambios en requisitos.

D. FUNDAMENTACIÓN DE LOS CONTENIDOS

La asignatura abarca problemas asociados a la ingeniería de software, y cubre varias metodologías de desarrollo de software de manera comparativa. Se hace fuerte énfasis en el rol de la modularización en la construcción de sistemas de tamaño mediano a pequeño, y de cómo las técnicas para *programming in the small* deben complementarse con técnicas para programación *in the large*. Además de intentar que los estudiantes adquieran conocimiento de los problemas asociados a la ingeniería de software y su historia, nos concentraremos en la enseñanza del *diseño por contratos* como metodología de desarrollo, poniendo énfasis en la aplicación de esta metodología en la práctica, en particular en el desarrollo de software de manera distribuida.

La relevancia del Diseño por Contratos en la Ingeniería de Software está evidenciada en las numerosas librerías y herramientas que permiten equipar programas con contratos, para una variedad de lenguajes: JML para Java, Code Contracts para C#, e incluso lenguajes de anotaciones para lenguajes de modelado, como es el caso de OCL para UML.

El uso de contratos es considerado hoy en día una herramienta clave para conseguir escalabilidad en el desarrollo de software.

E. ACTIVIDADES A DESARROLLAR

La materia involucra un proyecto de desarrollo colaborativo, altamente distribuido, coordinado por el equipo de la asignatura. Este proyecto permitirá que los alumnos se familiaricen con los principios y las prácticas que hacen a la ingeniería de software distribuida, mediante la reproducción de las condiciones en las cuales los desarrolladores se encuentran al encarar un proyecto de software (de mediana o gran escala) a ser desarrollado de manera colaborativa y distribuida.

Se fomentará además la lectura de material adicional y la auto organización de los alumnos en sus actividades. Además, se dejará en manos de los alumnos la instalación y manejo de las herramientas de software utilizadas en la asignatura, como una manera de estimular la práctica en cuestiones más técnicas (no necesariamente ligadas a los tópicos que la asignatura abarca), y la experiencia en la utilización de herramientas nuevas.

CLASES TEÓRICO-PRÁCTICAS: 8 horas semanales

F. NÓMINA DE TRABAJOS PRÁCTICOS

Además del proyecto de desarrollo de la materia, se proveerá una serie de ejercicios adicionales (cuya resolución es opcional) que acompañarán cada una de las clases teórico-prácticas.

G. HORARIOS DE CLASES:

El dictado de la asignatura estará compuesto por ocho horas semanales de clases teórico-prácticas, en los siguientes horarios:

Clases teórico-prácticas:
- martes de 18 a 21, lab. 101 del pabellón 2
- lunes de 16 a 18, lab. 101 del pabellón 2
- jueves de 18 a 21, lab. 101 del pabellón 2

HORARIO DE CLASES DE CONSULTAS: Las clases de consulta se darán semanalmente bajo demanda de los alumnos, en horario a convenir con los mismos.

H. MODALIDAD DE EVALUACIÓN:

Evaluaciones Parciales: La materia no tendrá exámenes parciales. Se evaluará sin embargo el desempeño de los alumnos en las diferentes etapas del proceso de desarrollo del proyecto (producción de documentos de alcance y especificación de requisitos, planificación del proyecto, diseño, implementación, etc).

Evaluación Final: Para regularizar la asignatura los estudiantes deben desempeñarse exitosamente en el proyecto de la misma. La asignatura tiene un régimen de aprobación por promoción directa, y la calificación depende enteramente del desempeño en el proyecto, evaluando separadamente cada etapa de desarrollo del mismo. Al finalizar el proyecto los alumnos deberán dar una presentación del

mismo, incluyendo una descripción técnica del proyecto, y una demostración de utilización del mismo.

CONDICIONES DE REGULARIDAD: Aprobación del proyecto de la asignatura.

CONDICIONES DE PROMOCIÓN: Aprobación del proyecto de la asignatura.

PROGRAMA ANALÍTICO

A. CONTENIDOS

Revisión de conceptos previos. Anotaciones en programas. La importancia de las especificaciones. Pre- y post-condición. Invariantes de ciclos. Funciones variantes. Análisis de corrección de programas. Análisis de terminación de programas.

Revisión de conceptos básicos de orientación a objetos e ingeniería de software. Modularidad. Reutilizabilidad. Tipos abstractos de datos. Las estructuras estática y dinámica: clases y objetos. Tipos básicos y de clase. Herencia y subtipado. Dispatch dinámico.

Soporte para la definición de contratos en lenguajes de programación modernos. Verificación de contratos en tiempo de ejecución. Violación de contratos y excepciones. El manejo de excepciones tradicional en lenguajes imperativos. El manejo de excepciones en lenguajes de programación modernos, orientados a objetos. Reintento o propagación como respuestas a excepciones. Manejo de eventos. Programación de rutinas polimórficas en lenguajes orientados a objetos.

Algunas bibliotecas esenciales para el desarrollo de software. Bibliotecas para entrada/salida, y abstracciones fundamentales. Bibliotecas para la programación de aplicaciones de redes. Sockets. Sockets de redes y sockets Unix. Aplicaciones cliente-servidor. Bibliotecas para persistencia. Manejo de archivos. Almacenamiento y recuperación de objetos en lenguajes de programación modernos. Interacción de aplicaciones con motores de bases de datos.

Desarrollo de software basado en componentes. La importancia de la modularización en la construcción de software. Desarrollo de software orientado a objetos. Nociones fundamentales. Clases como definiciones de tipos y módulos. Unificación de métodos y atributos. Dualidades entre las interpretaciones de clases como módulos y tipos. Ejemplos. Relaciones entre clases. Herencia. La herencia como extensión de módulos. La herencia como definición de subtipos. Clases como clientes de otras clases. Especificación de las responsabilidades de un módulo. Invariantes de módulos.

Los contratos. Contratos como especificaciones de las responsabilidades de clases. Contratos para rutinas. Relaciones entre contratos y nociones de Ciencias de la Computación. Triplas de Hoare y verificación de programas. El significado de las pre- y post-condiciones. La interpretación de las pre-y post-condiciones por parte de los clientes de una clase. Contratos de clases en forma de invariantes. Contratos y herencia. El principio de subcontratación. Subcontratación e invariantes. Subcontratación con respecto a pre- y post-condiciones.

El proceso de desarrollo de software puesto en combinación con contratos. La importancia de la reversibilidad. La generalización como un paso obligatorio en el proceso de desarrollo. Distintas etapas en el ciclo de vida. Especificación, diseño, implementación, validación y verificación. Reutilización.

Casos prácticos de aplicación de patrones de diseño y sus beneficios. Evolución de software, cambios en los requerimientos y su propagación en los diseños. Diseños resistentes a cambios en requerimientos.

B. CRONOGRAMA DE CLASES Y PARCIALES

Semana	Día/Fecha	Teóricos	Día/Fecha	Prácticos	Día/Fecha	Laboratorios	Parciales / Recuperatorios
1	16 al 19 de Agosto	Repaso de conceptos básicos de Ingeniería de Software	16 al 19 de Agosto	Prácticas básicas con lenguaje Java	16 al 19 de Agosto	Prácticas básicas con lenguaje Java	
2	22 al 26 de Agosto	Conceptos básicos de diseño por contratos	22 al 26 de Agosto	Prácticas de uso de contratos JML en Java	22 al 26 de Agosto	Prácticas de uso de contratos JML en Java	
3	29 de Agosto al 2 de Septiembre	Contratos y herencia.	29 de Agosto al 2 de Septiembre	Uso de herencia en combinación con contratos: práctica	29 de Agosto al 2 de Septiembre	Uso de herencia en combinación con contratos: práctica	
4	5 al 9 de Septiembre	Conceptos avanzados de diseño por contratos y Java: jerarquía de excepciones correspondientes a contratos, manejo de excepciones	5 al 9 de Septiembre	Práctica de captura de excepciones JML en Java	5 al 9 de Septiembre	Práctica de captura de excepciones JML en Java	
5	12 al 16 de Septiembre	Teoría sobre ingeniería de requisitos con diseño por contratos	12 al 16 de Septiembre	Inicio del proyecto de la materia: definiciones iniciales	12 al 16 de Septiembre	Inicio del proyecto de la materia: definiciones iniciales	

6	19 al 23 de Septiembre	Teoría sobre ingeniería de requisitos con diseño por contratos	19 al 23 de Septiembre	Desarrollo de documento de alcance	19 al 23 de Septiembre	Desarrollo de documento de alcance	
7	26 de Septiembre a 20 de Septiembre	Teoría sobre herramientas de poyo a la ing. De requisitos	26 de Septiembre a 20 de Septiembre	Desarrollo del documento de requisitos	26 de Septiembre a 20 de Septiembre	Desarrollo del documentode requisitos	
8	3 al 7 de Octubre	Teoría sobre diseño orientado a objetos, patrones de diseño y contratos	3 al 7 de Octubre	Inicio de la etapa de diseño del proyecto	3 al 7 de Octubre	Inicio de la etapa de diseño del proyecto	Primera entrega 6/10. setup inicial (creación de cuentas, conformación de equipos, propuesta de temas para proyecto)
9	10 al 14 de Octubre	Teoría sobre diseño orientado a objetos, patrones de diseño y contratos	10 al 14 de Octubre	Producción de diseño de API de cada módulo del proyecto	10 al 14 de Octubre	Producción de diseño de API de cada módulo del proyecto	
10	17 al 21 de Octubre	Teoría sobre testing, y desarrollo guiado por tests (TDD)	17 al 21 de Octubre	Evaluación de diseño y documentoación de API mediante TDD	17 al 21 de Octubre	Evaluación de diseño y documentación de API mediante TDD	Segunda entrega 20/10. documento de alcance y plan de administración de proyecto
11	25 al 29 de Octubre	Teoría sobre herramientas de apoyo al desarrollo de software OO usando Java	25 al 29 de Octubre	Inicio de la primera etapa de implementacion del proyecto	25 al 29 de Octubre	Inicio de la primera etapa de implementacion del proyecto	
12	31 de Octubre al 4 de Noviembre	Testing unitario usando JUnit	31 de Octubre al 4 de Noviembre	Continuación con implementación del proyecto	31 de Octubre al 4 de Noviembre	Continuación con implementación del proyecto	Tercera entrega 3/11. documento de requisitos
13	9 al 13 de	I/O,	9 al 13 de	Continuaci	9 al 13 de	Continuación	

	Noviembre	networking y bases de datos en Java	Noviembre	ón con implementación del proyecto	Noviembre	con implementación del proyecto	
14	14 al 18 de Noviembre	Generalización y su importancia en el diseño por contratos	14 al 18 de Noviembre	Finalización de implementación del proyecto	14 al 18 de Noviembre	Finalización de implementación del proyecto	Cuarta entrega 17/11. Diseño de API y contratos para la interfaz.

Al finalizar las 14 semanas mencionadas, los alumnos deberán realizar una presentación, que funcionará como defensa, de las componentes de software que les hubiera tocado desarrollar.

C. BIBLIOGRAFÍA

Obligatoria

P. Ammann y J. Offutt, *Introduction to Software Testing*, Cambridge University Press, 2008.

F. P. Brooks Jr., *The Mythical Man-Month*, Essays on Software Engineering, Addison-Wesley Publishing Co., 1982.

C. Ghezzi, M. Jazayeri y D. Mandrioli, *Fundamentals of Software Engineering*, Second Edition, Prentice Hall 2003.

P. Jalote, *An Integrated Approach to Software Engineering*, 3rd Edition, Springer, 2005.

B. Liskov, *Program Development in Java, Abstraction, Specification and Object-Oriented Design,* Addison-Wesley, 2001.

R. Martin, *Agile Software Development, Principles, Patterns and Practices*, Pearson, 2002.

B. Meyer, *Object Oriented Software Construction,* second edition, Addison-Wesley, 2000.

B. Meyer, *Touch of Class: Learning to Program Well with Objects and Contracts*, Springer, 2009.

De consulta

Tutoriales y manuales de herramientas de desarrollo y de apoyo al desarrollo (manuales y tutoriales online de Java, JML y JavaScript, tutoriales de Git, Github y demás herramientas).