ORIENTACIÓN A OBJETOS II	**Carrera/ Plan:** *Licenciatura en Informática* Plan 2021/Plan 2015/Plan 2012 *Licenciatura en Sistemas* Plan 2021/Plan 2015/Plan 2012 *Analista Programador Universitario* Plan 2021/Plan 2015/Plan 2007 Analista en Tecnologías de la Información y la Comunicación Plan 2021/Plan 2017
Año 2024	**Año:** 3ro **Régimen de Cursada:** *Semestral* **Carácter (Obligatoria/Optativa):** Obligatoria **Correlativas**: Orientación a Objetos I, Taller de lecto-comprensión y traducción de Ingles **Profesor/es:** Alejandra Garrido, Federico Balaguer, Leandro Antonelli. **Hs. semanales teoría**: 3 hrs **Hs. semanales práctica**: 3 hrs

FUNDAMENTACIÓN

La programación orientada a objetos es central para la formación del profesional informático. Los principios básicos del paradigma, cubiertos en Orientación a Objetos 1, brindan las bases para obtener micro-diseños donde los objetos/clases son módulos que exhiben características de bajo acoplamiento y alta cohesión. Sobre esas bases podemos construir diseños y arquitecturas de mayor envergadura, como aplicaciones, servicios, librerías y frameworks. Abstraer a ese nivel requiere nuevas herramientas conceptuales y tecnológicas. Los patrones de diseño orientado a objetos capturan y comunican soluciones probadas a problemas frecuentes de diseño OO. Los frameworks nos permiten capturar, en un artefacto de código bien diseñado e implementado, la complejidad de todo un dominio. El proceso de diseño da lugar al refactoring continuo de aplicaciones para lo cual el testing automatizado de unidad es una herramienta fundamental.

OBJETIVOS GENERALES

Profundizar los temas desarrollados por el alumno en Orientación a Objetos 1 e introducir conceptos fundamentales en la construcción de arquitecturas de software modulares, extensibles y reusables, a través de conceptos fundamentales como son: patrones de diseño, refactoring hacia patrones y frameworks orientados a objetos. Se profundizará también en el uso de un lenguaje de modelado gráfico orientado a objetos (UML), que le permitirá construir diagramas especificando distintos aspectos de un sistema. Los trabajos prácticos se realizarán usando el lenguaje de modelado y el lenguaje de implementación Java, que resulta el más apropiado de acuerdo a estos objetivos.

RESULTADOS DE APRENDIZAJE

1.2. Describir las características de los últimos avances en hardware y software y sus correspondientes aplicaciones prácticas (Adecuado).
1.3. Describir los avances informáticos actuales e históricos y demostrar cierta visión sobre tendencias y avances futuros (Adecuado).
1.4. Aplicar e integrar conocimientos de otras disciplinas informáticas como apoyo al estudio de la propia área de especialidad (o áreas de especialidad) (Adecuado).

1.5. Demostrar sensibilización ante la necesidad de contar con amplios conocimientos a la hora de crear aplicaciones informáticas en otras áreas temáticas (Adecuado).
2.2. Describir un determinado problema y su solución a varios niveles de abstracción (Adecuado).
2.3. Seleccionar y utilizar los correspondientes métodos analíticos, de simulación y de modelización (Adecuado).

2.4. Escoger los patrones de solución, algoritmos y estructuras de datos apropiados (Adecuado).
2.5. Analizar la medida en la que un determinado sistema informático cumple con los criterios definidos para su uso actual y desarrollo futuro (Básico).
3.1. Definir y diseñar hardware/software informático/de red que cumpla con los requisitos establecidos (Adecuado).
3.2. Describir las fases implicadas en distintos modelos de ciclo de vida con respecto a la definición, construcción, análisis y puesta en marcha de nuevos sistemas y el mantenimiento de sistemas existentes (Adecuado).
3.3. Elegir y utilizar modelos de proceso adecuados, entornos de programación y técnicas de gestión de datos con respecto a proyectos que impliquen aplicaciones tradicionales así como aplicaciones emergentes (Adecuado).
3.4. Describir y explicar el diseño de sistemas e interfaces para interacción persona-ordenador y ordenador-ordenador (Adecuado).
5.2. Describir y explicar las técnicas de gestión correspondientes al diseño, implementación, análisis, uso y mantenimiento de sistemas informáticos, incluyendo gestión de proyectos, de configuración y de cambios, así como las técnicas de automatización correspondientes (Adecuado).

COMPETENCIAS

- CGS6- Capacidad para interpretar la evolución de la Informática con una visión de las tendencias tecnológicas futuras.
- CGT2- Concebir, diseñar y desarrollar proyectos de Informática.
- CGT3- Gestionar, planificar, ejecutar y controlar proyectos de Informática.
- CGT8 Capacidad de interpretación y resolución de problemas multidisciplinarios, desde los conocimientos de la disciplina informática.
- LI- CE4 – Planificar, dirigir, realizar y/o evaluar proyectos de relevamiento de problemas del mundo real, especificación formal de los mismos, diseño, implementación, prueba, verificación, validación, mantenimiento y control de calidad de sistemas de software/sistemas de información que se ejecuten sobre equipos de procesamiento de datos, con capacidad de incorporación de tecnologías emergentes del cambio tecnológico. Capacidad de análisis, diseño y evaluación de interfases humano computador y computador-computador.
- LI- CE6 – Controlar las normas de calidad en el software o software integrado a otros componentes. Capacidad de evaluación de perfomance de sistemas de software y sistemas que integren hardware y software.
- LS- CE1 – Planificar, dirigir, realizar y/o evaluar proyectos de relevamiento de problemas del mundo real. Especificación formal, diseño, implementación, prueba, verificación, validación, mantenimiento y control de calidad de sistemas de software que se ejecuten sobre sistemas de procesamiento de datos, con capacidad de incorporación de tecnologías emergentes del cambio tecnológico. Capacidad de análisis, diseño y evaluación de interfases humano computador y computador-computador.
- LS- CE5 – Establecer métricas y normas de calidad y seguridad de software, contralando las mismas a fin de tener un producto industrial que respete las normas nacionales e internacionales. Control de la especificación formal del producto, del proceso de diseño, desarrollo, implementación y mantenimiento. Establecimiento de métricas de validación y certificación de calidad. Capacidad de evaluación de perfomance de sistemas de software y sistemas que integren hardware y software.

CONTENIDOS MINIMOS (de acuerdo al Plan de Estudios)

- Diseño Orientado a Objetos
- Patrones de diseño
- Construcción de aplicaciones con frameworks orientados a objetos.
- Refactoring. Testing. Metodologías de Diseño Ágiles.

PROGRAMA ANALÍTICO

I - Diseño Orientado a Objetos

1. La filosofía del proceso de desarrollo de software. Las etapas del proceso de desarrollo de software. Procesos de desarrollo iterativos e incrementales, basados en modelos: utilidad de los modelos. Los modelos a través del proceso de desarrollo de software. Cualidades y clasificación de los modelos.

2. UML como Lenguaje de modelado. Diagrama de clases. Diagrama de estados. Diagrama de interacción.

II -Patrones de Diseño:

3. Introducción a Patrones. Definición de Patrón. Descripción de un patrón. Catálogo de Patrones.

4. Patrones de diseño. Definición. Descripción de un patrón de diseño. Organización del Catálogo de patrones de diseño. Utilidad de los patrones de diseño. Selección de los patrones de diseño. Uso de los patrones de diseño.

5. Patrones creacionales: Factory Method, Builder, Singleton.

6. Patrones estructurales: Adapter, Composite, Decorador, Proxy.

7. Patrones de comportamiento: Template Method, Observer, State, Strategy.

8. Otros patrones: Type Object, Null Object, patrones de análisis.

III -Refactoring

9. Introducción a Refactoring. Su importancia como práctica en el contexto de los métodos ágiles de desarrollo. El concepto de "bad smells" como disparadores de refactorings. Catálogo de refactorings.

10. Manipulación de métodos largos: Extract Method, Inline Method, Replace Temp with Query, Split Temporary Variable. Mover aspectos entre objetos: Move Method, Move Field, Extract class. Organización de datos: Self Encapsulate Field, Encapsulate Field, Replace Data Value with Objects. Simplificación de invocación de métodos: Rename Method, Replace Parameter with Method, Preserve Whole Object. Simplificación de expresiones condicionales: Replace Conditional with Polimorfism, Decompose Conditional. Manipulación de la generalización: Pull Up Method, Push Down Method, Extract Subclass, Extract Superclass, Collapse Hierarchy.

11. Refactoring hacia patrones. Form Template Method. Replace conditional logic with strategy. Replace State-Altering Conditionals with State. Unify interfaces with Adapter. Move Embellishment to Decorator.

IV -Frameworks

12. Introducción a Frameworks. Reutilización de software vs. reutilización de diseño. Clasificación de frameworks según su propósito.

13. Frameworks basados en herencia (white box frameworks). Frameworks basados en composición (black boxframeworks).

14. Elementos centrales en la implementación de un framework: Inversión de control, Hostposts, Frozenspots.

15. Las plantillas y los ganchos como generadores de hotspots e inversión de control. Su implementación con herencia y con composición.

16. Instanciación de frameworks.

17. Diseño evolutivo de frameworks. El rol estratégico de los patrones, el refactoring, y los tests de unidad.

V - Test Driven Development

18. Testing. Importancia. Tipos de tests: de unidad, de integración, de aceptación. Metodología de desarrollo ágil TDD: "Test Driven Development". Relación entre refactoring y testing.

19. Patrones de tests de unidad: familia de patrones XUnit.

20. Test doubles. Mock objects.

BIBLIOGRAFÏA

BIBLIOGRAFÍA OBLIGATORIA

1. Design Patterns. Elements of Reusable Objects Oriented Software. Gamma, Helm, Johnson, Vlissides, Addison-Wesley, Professional Computing Series.
2. Refactoring: Improving the Design of Existing Code. Fowler, Martin. Addison-Wesley, 1999.
3. Refactoring to Patterns. Joshua Kerievsky. Addison Wesley, 2004. ISBN: 0-321- 21335-1
4. Implementing Application Frameworks: Object-Oriented Frameworks at Work (Hardcover). Mohamed E. Fayad (Editor), Douglas C. Schmidt (Editor), Ralph E. Johnson (Editor).

BIBLIOGRAFÍA COMPLEMENTARIA

1. Analysis Patterns. Reusable Object Models. Martin Fowler. Addison-Wesley, 1997.
2. Extreme Programming Explained. Kent Beck and Cynthia Andres. Addison-Wesley, 2005.
3. Building Application Frameworks: Object-Oriented Foundations of Framework Design. Mohamed E. Fayad (Editor), Ralph E. Johnson (Author), Douglas C. Schmidt (Editor).
4. Domain-Specific Application Frameworks: Frameworks Experience by Industry (Hardcover). Mohamed E. Fayad (Editor), Ralph E. Johnson (Editor).
5. Johnson, R. E. 1997. Components, frameworks, patterns. In Proceedings of the 1997 Symposium on Software Reusability (Boston, Massachusetts, United States, May 17 - 20, 1997). M. Harandi, Ed. SSR ’97. ACM Press, New York, NY, 10-17. [PDF]
6. Designing Reusable Classes. B. Foote, R. Johnson. Journal of Object-Oriented Programming, 1998.
7. D. Roberts and R. Johnson. Evolving Frameworks: A Pattern Language for Developing Object-Oriented Frameworks. Proceedings of Pattern Languages of Programs, Allerton Park, Illinois, September 1996.[PDF]
8. XUnit Test Patterns: Refactoring Test Code. Gerard Meszaros. Addison-Wesley, 2007.

METODOLOGÍA DE ENSEÑANZA

La cursada de la materia se organiza en clases teóricas, clases de actividades prácticas y clases de consulta teórico-práctica.

En las clases teóricas se imparten los conceptos de la materia, induciendo a la participación de los alumnos en la resolución de ejercicios de diseño que permite la incorporación activa de los conocimientos.

En las clases prácticas se aplican y fijan los contenidos vistos en teoría a través de trabajos prácticos enfocados, y proyectos integradores. Los alumnos pueden acercarse a realizar consultas individuales sobre la resolución de los trabajos durante turnos de consulta de práctica separados en distintas comisiones para garantizar una adecuada relación alumno-docente. Además, las clases de consultas son en sala equipadas con computadoras, fijas o móviles. La materia cuenta con una plataforma online para intercambio de materiales y en la que se fomenta la discusión e intercambio de conocimiento entre pares y con los docentes.

Las clases de consulta teórico-práctica requieren la preparación previa del alumno en base al material disponible y las prácticas, para dar lugar a una discusión más profunda y reflexiva entre los alumnos y los docentes, al tiempo que se evacúan dudas que pueden haber quedado con el material.

Si bien la asignatura no hace foco en una tecnología en particular, los cambios conceptuales y metodológicos que plantea la programación orientada a objetos han tenido un fuerte impacto en el desarrollo de las tecnologías. Tanto en esta asignatura como en Orientación a Objetos 1 (pre-requisito) se propone a los alumnos reflexionar sobre dicha evolución y cambios (por ejemplo, desde la programación estructurada a la programación orientada a objetos). Algunas de las herramientas tecnológicas que los alumnos incorporarán como tendencias de la disciplina son las herramientas de detección de problemas de diseño e implementación (lints), las herramientas de refactoring asistido, la depuración de aplicaciones en tiempo real, los inspectores de objetos, y los frameworks orientados a objetos. La cátedra acompaña el proceso del alumno, para contrastar las conclusiones del alumno y validar su habilidad para esta competencia.

En la cátedra se pone énfasis en el proceso de identificación de problemas del mundo real, especificación de los mismos como problemas resolubles desde la informática, formulación de proyectos para su solución y en el análisis y diseño de las soluciones en el marco del proyecto formulado.

El trabajo integrador (y otras entregas grupales) ponen en juego y fortalecen la capacidad del alumno para gestionar, planificar, ejecutar y controlar proyectos de Informática (a pequeña escala). El grado con el que alcanza estas competencias se refleja directamente en el resultado del proyecto integrador y demás entregas.

En la cátedra se tratan proyectos multidisciplinarios (los requerimientos y dominios de aplicación tomados como ejemplo provienen de muchas disciplinas). Además de ofrecer ejemplos motivadores que sirvan como practica de las temáticas de la asignatura, se trata de acompañar al alumno (aprovechando la experiencia de trabajo multidisciplinar del equipo docente) en la interpretación del rol del Informático como articulador de soluciones en áreas de conocimiento muy diferentes que requieren participación de expertos extra-disciplinares.

La calidad se enfoca desde la perspectiva de la claridad del código, facilidad de mantenimiento, bajo acoplamiento y alta cohesión entre módulos (objetos). La carencia de esos atributos de calidad se hace explicita en términos de “malos olores”, los cuales se han catalogado como indicadores de problemas en el código, que pueden medirse a través de métricas explícitas e indican la necesidad de aplicar refactorings (que mejoran la calidad interna del software sin introducir cambios en su comportamiento). Los tests de unidad automatizado se proponen como herramientas para hacer explícitos contratos que el software debe cumplir, de manera que puedan ser verificables. Evaluación de calidad, testing automatico, refactoring y

abstracción en elementos de código reutilizables (por ejemplo, frameworks) son pasos fundamentales en la planificación, realización y evaluación de proyectos de software. En cuanto a las interfaces humano computador, en la materia se diseña, desarrolla y evalúa una aplicación interactiva, haciendo énfasis en la importancia de la separación modelo-vista, característica fundamental para poder incorporar las competencias de diseño de interacción sin que se vean atadas a la evolución de la lógica del dominio.

Entre los ejemplos de frameworks orientados a objetos que se mencionan en la cátedra, encontramos aquellos que nos permiten abstraernos de dependencias concretas de hardware, redes, y sistema operativo. Calidad, en ese contexto, se define como en nivel de abstracción que las librerías/frameworks ofrecen al desarrollador de software sobre las plataformas (hardware/software) subyacente. Conseguir un bajo acoplamiento entre código de aplicación y código de infraestructura es la base para hacer posible la portabilidad y evolución independiente de ambas partes.

En el caso de que la materia se desarrolle en **modalidad virtual**, las clases teóricas se ofrecerán mediante una combinación de material audio-visual publicado en la plataforma virtual y clases virtuales sincrónicas. El docente expondrá el tema del día a través de una herramienta/plataforma de libre acceso para reuniones virtuales, promoviendo la participación de los alumnos. Las presentaciones quedarán grabadas y disponibles. Al igual que en el caso de las clases teóricas, de no ser posible la presencialidad, habrá actividades prácticas y clases de consulta teórico-prácticas en encuentros sincrónicos y discusiones asincrónicas. Estas últimas serán a través de los foros de discusión.

EVALUACIÓN

La evaluación de los aspectos prácticos de la asignatura (cursada) tiene lugar en el visado de entregas y un examen escrito, presencial con dos oportunidades de recuperación. La aprobación final puede ser la realización de un proyecto y su defensa, la evaluación de promoción para quienes accedan a esta opción, o un examen final escrito.

La materia cuenta con un régimen de promoción bajo ciertas condiciones. Los alumnos que aprueben la promoción quedarán eximidos de rendir examen final de la materia. Para promocionar la materia se requiere: (i) aprobar la cursada en 1ra o 2da fecha de parcial; (ii) aprobar el parcial con calificación "Accede a promoción", alcanzable cuando el parcial se resuelve entre muy bien y sobresaliente; y (iii) aprobar un examen integrador de promoción con calificación 6 o más.

La evaluación de las competencias generales y específicas a cuya adquisición esta cátedra contribuye, tiene lugar en el examen parcial y las múltiples entrevistas y discusiones que el alumno tiene con los docentes durante el desarrollo de los trabajos prácticos visables, y se refleja en la calificación de los mismos.

En el caso que la materia se desarrolle en **modalidad virtual**, las entregas intermedias de trabajos prácticos visables se realizarán de la misma manera que en modalidad presencial. El coloquio de promoción se realizará a través de un examen utilizando una plataforma virtual de evaluación, y el examen final se desarrollará en forma individual y con entrega pautada a través de la modalidad virtual que determine el docente.

CRONOGRAMA DE CLASES Y EVALUACIONES

Clase	Contenidos/Actividades
1	Presentación de la asignatura. Diseño OO. Características de un buen diseño OO.
2	Patrones de diseño: introducción. Patrones Adapter, Composite y Template Method. Práctica de repaso de diseño.
3	Patrones State y Strategy. Práctica aplicando los patrones vistos.
4	Patrones wrappers: Decorator, Adapter y Proxy. Práctica con patrones.
5	Patrones creacionales: Builder, Factory Method, Singleton. Práctica con patrones.
6	Cómo leer un patrón: Type Object. Patrón Null Object. Patrones de análisis. Práctica con patrones.
7	Test doubles. Test Driven Development y métodos ágiles. Práctica de patrones.
8	Refactoring: Introducción. Catálogo de refactorings de Fowler. El concepto de "bad smell". Bad smells clásicos. Práctica de testing de unidad.
9	Refactoring hacia patrones de diseño. Practica de refactoring.
10	Frameworks Orientados a Objetos. Hot-spots y frozen-spots. Inversión de control. Ejemplos de frameworks Java. Practiva de refactoring hacia patrones.
11	Patrones en frameworks. Template Method como generador de hotspots e inversión de control. Plantillas y ganchos con herencia o con composición. Practica de frameworks.
12	Estrategias para implementar inversión de control. Inversión de dependencias. Practica de template method e inversión de control.
13	Integración de contenidos: Testing, Patrones, Refactoring, Frameworks. Prácticas integradoras.
14	Prácticas integradoras. Clase de consulta. Parcial
15	Prácticas integradoras. Clase de consulta.
16	Clase de consulta. Recuperatorio

17	Clase de consulta
18	Recuperatorio

Evaluaciones previstas	**Fecha**
Parcial	08/06/2024
Recuperatorio	29/06/2024
Recuperatorio	13/07/2024

Contacto de la cátedra (mail, sitio WEB, plataforma virtual de gestión de cursos):

Consultas referidas a la teoría: Alejandra.Garrido@lifia.info.unlp.edu.ar, Federico.Balaguer@lifia.info.unlp.edu.ar

Consultas referidas a la práctica: Gabriela.Perez@lifia.info.unlp.edu.ar

Firma del/los profesor/es

ORIENTACIÓN A OBJETOS II

Redictado

Año 2024

Carrera/ Plan:

Licenciatura en Informática Plan 2021/Plan 2015/Plan 2012
Licenciatura en Sistemas Plan 2021/Plan 2015/Plan 2012
Analista Programador Universitario Plan 2021/Plan 2015/Plan 2007
Analista en Tecnologías de la Información y la Comunicación Plan 2021/Plan 2017
Año: 3ro
Régimen de Cursada: *Semestral*
Carácter (Obligatoria/Optativa): Obligatoria
Correlativas: Orientación a Objetos I,Taller de lecto-comprensión y traducción de Ingles
Profesor/es: Alejandra Garrido
Federico Balaguer
Hs. semanales teoría: 2hrs
Hs. semanales práctica: 4hrs

FUNDAMENTACIÓN

La programación orientada a objetos es central para la formación del profesional informático. Los principios básicos del paradigma, cubiertos en Orientación a Objetos 1, brindan las bases para obtener micro-diseños donde los objetos/clases son módulos que exhiben características de bajo acoplamiento y alta cohesión. Sobre esas bases podemos construir diseños y arquitecturas de mayor envergadura, como aplicaciones, servicios, librerías y frameworks. Abstraer a ese nivel requiere nuevas herramientas conceptuales y tecnológicas. Los patrones de diseño orientado a objetos capturan y comunican soluciones probadas a problemas frecuentes de diseño OO. Los frameworks nos permiten capturar, en un artefacto de código bien diseñado e implementado, la complejidad de todo un dominio. El proceso de diseño da lugar al refactoring continuo de aplicaciones para lo cual el testing automatizado de unidad es una herramienta fundamental.

OBJETIVOS GENERALES

Profundizar los temas desarrollados por el alumno en Orientación a Objetos 1 e introducir conceptos fundamentales en la construcción de arquitecturas de software modulares, extensibles y reusables, a través de conceptos fundamentales como son: patrones de diseño, refactoring hacia patrones y frameworks orientados a objetos. Se profundizará también en el uso de un lenguaje de modelado gráfico orientado a objetos (UML), que le permitirá construir diagramas especificando distintos aspectos de un sistema. Los trabajos prácticos se realizarán usando el lenguaje de modelado y el lenguaje de implementación Smalltalk, que resulta el más apropiado de acuerdo a estos objetivos.

RESULTADOS DE APRENDIZAJE

1.2. Describir las características de los últimos avances en hardware y software y sus correspondientes aplicaciones prácticas (Adecuado).
1.3. Describir los avances informáticos actuales e históricos y demostrar cierta visión sobre tendencias y avances futuros (Adecuado).
1.4. Aplicar e integrar conocimientos de otras disciplinas informáticas como apoyo al estudio de la propia área de especialidad (o áreas de especialidad) (Adecuado).
1.5. Demostrar sensibilización ante la necesidad de contar con amplios conocimientos a la hora de crear aplicaciones informáticas en otras áreas temáticas (Adecuado).
2.2. Describir un determinado problema y su solución a varios niveles de abstracción (Adecuado).

2.3. Seleccionar y utilizar los correspondientes métodos analíticos, de simulación y de modelización (Adecuado).

2.4. Escoger los patrones de solución, algoritmos y estructuras de datos apropiados (Adecuado).
2.5. Analizar la medida en la que un determinado sistema informático cumple con los criterios definidos para su uso actual y desarrollo futuro (Básico).
3.1. Definir y diseñar hardware/software informático/de red que cumpla con los requisitos establecidos (Adecuado).
3.2. Describir las fases implicadas en distintos modelos de ciclo de vida con respecto a la definición, construcción, análisis y puesta en marcha de nuevos sistemas y el mantenimiento de sistemas existentes (Adecuado).
3.3. Elegir y utilizar modelos de proceso adecuados, entornos de programación y técnicas de gestión de datos con respecto a proyectos que impliquen aplicaciones tradicionales así como aplicaciones emergentes (Adecuado).
3.4. Describir y explicar el diseño de sistemas e interfaces para interacción persona-ordenador y ordenador-ordenador (Adecuado).
5.2. Describir y explicar las técnicas de gestión correspondientes al diseño, implementación, análisis, uso y mantenimiento de sistemas informáticos, incluyendo gestión de proyectos, de configuración y de cambios, así como las técnicas de automatización correspondientes (Adecuado).

COMPETENCIAS

- CGS6- Capacidad para interpretar la evolución de la Informática con una visión de las tendencias tecnológicas futuras.
- CGT2- Concebir, diseñar y desarrollar proyectos de Informática.
- CGT3- Gestionar, planificar, ejecutar y controlar proyectos de Informática.
- CGT8 Capacidad de interpretación y resolución de problemas multidisciplinarios, desde los conocimientos de la disciplina informática.
- LI- CE4 – Planificar, dirigir, realizar y/o evaluar proyectos de relevamiento de problemas del mundo real, especificación formal de los mismos, diseño, implementación, prueba, verificación, validación, mantenimiento y control de calidad de sistemas de software/sistemas de información que se ejecuten sobre equipos de procesamiento de datos, con capacidad de incorporación de tecnologías emergentes del cambio tecnológico. Capacidad de análisis, diseño y evaluación de interfases humano computador y computador-computador.
- LI- CE6 – Controlar las normas de calidad en el software o software integrado a otros componentes. Capacidad de evaluación de perfomance de sistemas de software y sistemas que integren hardware y software.
- LS- CE1 – Planificar, dirigir, realizar y/o evaluar proyectos de relevamiento de problemas del mundo real. Especificación formal, diseño, implementación, prueba, verificación, validación, mantenimiento y control de calidad de sistemas de software que se ejecuten sobre sistemas de procesamiento de datos, con capacidad de incorporación de tecnologías emergentes del cambio tecnológico. Capacidad de análisis, diseño y evaluación de interfases humano computador y computador-computador.
- LS- CE5 – Establecer métricas y normas de calidad y seguridad de software, contralando las mismas a fin de tener un producto industrial que respete las normas nacionales e internacionales. Control de la especificación formal del producto, del proceso de diseño, desarrollo, implementación y mantenimiento. Establecimiento de métricas de validación y certificación de calidad. Capacidad de evaluación de perfomance de sistemas de software y sistemas que integren hardware y software.

CONTENIDOS MINIMOS (de acuerdo al Plan de Estudios)

- Diseño Orientado a Objetos
- Patrones de diseño
- Construcción de aplicaciones con frameworks orientados a objetos.
- Refactoring. Testing. Metodologías de Diseño Ágiles.

PROGRAMA ANALÍTICO

I - Diseño Orientado a Objetos

1. La filosofía del proceso de desarrollo de software. Las etapas del proceso de desarrollo de software. Procesos de desarrollo iterativos e incrementales, basados en modelos: utilidad de los modelos. Los modelos a través del proceso de desarrollo de software. Cualidades y clasificación de los modelos.

2. UML como Lenguaje de modelado. Diagrama de clases. Diagrama de estados. Diagrama de interacción.

II -Patrones de Diseño:

3. Introducción a Patrones. Definición de Patrón. Descripción de un patrón. Catálogo de Patrones.

4. Patrones de diseño. Definición. Descripción de un patrón de diseño. Organización del Catálogo de patrones de diseño. Utilidad de los patrones de diseño. Selección de los patrones de diseño. Uso de los patrones de diseño.

5. Patrones creacionales: Factory Method, Singleton.

6. Patrones estructurales: Composite, Decorador, Adapter, Proxy.

7. Patrones de comportamiento: State, Strategy, Template Method, Command.

III -Refactoring

8. Introducción a Refactoring. Su importancia como práctica en el contexto de los métodos ágiles de desarrollo. El concepto de "bad smells" como disparadores de refactorings. Catálogo de refactorings.

9. Manipulación de métodos largos: Extract Method. lnline Method, Replace Temp with Queries, Split Temporary Variable. Mover aspectos entre objetos: Move Method, Move Field, Extract class. Organización de datos: Self Encapsulate Field, Encapsulate Field, Replace Data Value with Objects. Simplificación de invocación de métodos: Rename Method, Replace Parameter with Method, Preserve Whole Object. Simplificación de expresiones condicionales: Replace Conditional with Polimorfism, Decompose Conditional. Manipulación de la generalización: Pull Up Method, Push Down Method, Extract Subclass, Extract Superclass, Collapse Hierarchy.

10. Refactoring hacia patrones. Form Template Method. Replace conditional logic with strategy. Replace State-Altering Conditionals with State. Unify interfaces with Adapter. Move Embellishment to Decorator.

IV -Frameworks

11. Introducción a Frameworks. Reutilización de software vs. reutilización de diseño. Clasificación de frameworks según su propósito.

12. Frameworks basados en herencia (white box frameworks). Frameworks basados en composición (black boxframeworks).

13. Elementos centrales en la implementación de un framework: Inversión de control, Hostposts, Frozenspots.

14. Las plantillas y los ganchos como generadores de hotspots e inversión de control. Su implementación con herencia y con composición.

15. Instanciación de frameworks; casos de estudio: Seaside, SUnit.

16. Diseño evolutivo de frameworks. El rol estratégico de los patrones, el refactoring, y los tests de unidad.

17. Documentación de frameworks: ejemplos, hotspot cards, y patrones

V - Test Driven Development

18. Testing. Importancia. Tipos de tests: de unidad, de integración, de aceptación. Metodología de desarrollo ágil TDD: "Test Driven Development". Relación entre refactoring y testing.

19. Patrones de tests de unidad: familia de frameworks XUnit.

20. El framework SUnit de test de unidad en Smalltalk.

BIBLIOGRAFÏA

BIBLIOGRAFÍA OBLIGATORIA

1. Design Patterns. Elements of Reusable Objects Oriented Software. Gamma, Helm, Johnson, Vlissides, Addison-Wesley, Professional Computing Series.
2. Refactoring: lmproving the Design of Existing Code. Fowler, Martin. Addison-Wesley, 1999.
3. Refactoring to Patterns. Joshua Kerievsky. Addison Wesley, 2004. ISBN: 0-321- 21335-1
4. Implementing Application Frameworks: Object-Oriented Frameworks at Work (Hardcover). Mohamed E. Fayad (Editor), Douglas C. Schmidt (Editor), Ralph E. Johnson (Editor).

BIBLIOGRAFÍA COMPLEMENTARIA

1. Extreme Programming Explained. Kent Beck and Cynthia Andres. Addison-Wesley, 2005.
2. Building Application Frameworks: Object-Oriented Foundations of Framework Design. Mohamed E. Fayad (Editor), Ralph E. Johnson (Author), Douglas C. Schmidt (Editor).
3. Domain-Specific Application Frameworks: Frameworks Experience by Industry (Hardcover). Mohamed E. Fayad (Editor), Ralph E. Johnson (Editor).
4. Johnson, R. E. 1997. Components, frameworks, patterns. In Proceedings of the 1997 Symposium on Software Reusability (Boston, Massachusetts, United States, May 17 - 20, 1997). M. Harandi, Ed. SSR ’97. ACM Press, New York, NY, 10-17. [PDF]
5. Designing Reusable Classes. B. Foote, R. Johnson. Journal of Object-Oriented Programming, 1998.
6. D. Roberts and R. Johnson. Evolving Frameworks: A Pattern Language for Developing Object-Oriented Frameworks. Proceedings of Pattern Languages of Programs, Allerton Park, Illinois, September 1996.[PDF]
7. Kent Beck. Simple Smalltalk Testing: With Patterns. http://www.xprogramming.com/testfram.htm
8. Stephane Ducasse. SUnit Explained. http://www.iam.unibe.ch/~ducasse/

METODOLOGÍA DE ENSEÑANZA

El curso está dirigido a alumnos que previamente cursaron Orientación a Objetos II, por lo cual, durante el redictado, se hará más énfasis en las actividades prácticas, repasando y reforzando los conceptos teóricos.

Habrá dos encuentros semanales en horario de la tarde. Cada uno de los encuentros se realizará la presentacion teorica corresponditente de la semana (1hora) y las siguientes 2 horas sera de resolucion de ejercicios

Se dispondrá de una plataforma para que los alumnos puedan estar en contacto y obtener respuesta de los docentes en todo momento.

En el caso de que la materia se desarrolle en **modalidad virtual**, las clases se ofrecerán mediante una combinación de material audio-visual publicado en la plataforma virtual y clases virtuales sincrónicas donde se enfatizará en la participación de los alumnos. Las presentaciones quedarán grabadas y disponibles. Del mismo modo se dispondrá de foros de discusión para la consulta y el intercambio de opiniones de manera asincrónica.

EVALUACIÓN

Trabajo con el fin de promocionar el final. El trabajo posee una fecha de entrega, y los alumnos podrán hacer consultas durante el desarrollo. Finalmente, en la entrega deberán realizar un coloquio para defender el trabajo.

En el caso que la materia se desarrolle en **modalidad virtual**, la entrega del trabajo de promoción se realizará de forma online, con la posibilidad de usar un repositorio de versionado de código. El coloquio de promoción y el examen final se desarrollará en forma individual y con entrega pautada a través de la modalidad virtual que determine el docente.

CONDICIONES DE CURSADA

Podrán inscribirse y cursar este redictado, aquellos alumnos que habiendo cursado Orientación a Objetos II en el semestre inmediato anterior, hayan desaprobado con calificación R (acceso a redictado).

CRONOGRAMA DE CLASES Y EVALUACIONES

Clase	Contenidos/Actividades
1	Presentación de la asignatura. Diseño OO. Características de un buen diseño OO.
2	Patrones de diseño: introducción. El patrón Adapter. Prácticas de diseño en implementación.
3	Patrónes Template Method y Composite. Práctica aplicando el patrón Adapter.
4	Patrones Strategy y State. Práctica con Patrón Composite y Template method.
5	Contrucción de soluciones con framework *antlr*.
6	Patrón Visitor

7	Introducción a refactoring y Test Driven Development. Patrones de XUnit. Práctica de construcción de aplicaciones con Seaside.
8	Refactoring: catálogo de refactorings de Fowler. El concepto de "bad smell". Bad smells clásicos. Práctica de testing de unidad.
9	Refactoring hacia patrones de diseño. Refactoring to Template method. Refactoring to Strategy. Práctica de refactorings.
10	Reuso; Librerías. Introducción a Frameworks Orientados a Objetos. Ejemplo de framework: Seaside, SUnit, entre otros. Práctica de refactoring hacia patrones.
11	Plantillas y ganchos utilizando herencia y composición. Plantillas y ganchos en los hotspots. Prácticas integradoras.
12	Integración de contenidos: Testing, Patrones, Refactoring, Frameworks. Prácticas integradoras.
13	Prácticas integradoras. Clase de consulta. Parcial
14	Prácticas integradoras. Clase de consulta.
15	Clase de consulta. Recuperatorio
16	Clase de consulta
17	Recuperatorio

Evaluaciones previstas	Fecha
Parcial	Semana del 25/11
Recuperatorio	Semana del 9/12
Recuperatorio	Semana del 16/12

Contacto de la cátedra (mail, sitio WEB, plataforma virtual de gestión de cursos):

federico.balaguer@lifia.info.unlp.edu.ar

Firma del/los profesor/es