FACULTAD DE INFORMATICA | UNIVERSIDAD NACIONAL DE LA PLATA

Maestría en Ingeniería de Software – Modalidad a distancia

Técnicas y Herramientas

Plan de estudios 2021
Año 2025

Duración: 108 hs. Totales.

Cantidad de horas VC: 30hs
Cantidad de horas de actividades en línea y de trabajo final: 78hs.

Docente Responsable: Dr. Grigera, Julián
Docentes Tutores: Dr. Urbieta, Matías (Académico/Tecnológico)
Dr. Díaz-Pace, Andrés (Académico/Tecnológico)
Tutores: **Dr. Gardey, Juan Cruz** (Académico/Tecnológico)
Maiten, Meza (Administrativo)

OBJETIVOS GENERALES:

El objetivo de la materia es introducir a los alumnos en técnicas modernas de la programación, en particular en aspectos de la programación orientada a objetos, diseño orientado a objetos, técnicas complementarias como test de unidad, refactorings e introducción a patrones de diseño.
Se introduce también al alumno en el uso de un lenguaje de modelado gráfico orientado a objetos (UML), que le permitirá construir diagramas especificando distintos aspectos de un sistema.
Se enfatiza en la construcción de arquitecturas de software modulares, extensibles y reusables, que son conceptos claves para aplicaciones de gran porte.

COMPETENCIAS A DESARROLLAR EN RELACION CON EL OBJETIVO DE LA CARRERA

C.1- Manejar y aplicar tecnologías actuales para el desarrollo de sistemas de software, incluyendo métodos, lenguajes, arquitecturas, frameworks y herramientas.

C.2- Tener capacidad para analizar diferentes modelos de proceso de desarrollo de software y evaluar su calidad tanto en aspectos del producto resultante como en la gestión de los individuos involucrados y sus interacciones.

C.4- Definir parámetros de calidad tanto interna como externa de un producto software, y establecer procesos de evaluación y mejora que atiendan la satisfacción de todos los

involucrados (el cliente, los usuarios y su experiencia, y el equipo de desarrollo).

C.6- Tener capacidad de analizar el estado del arte en los distintos aspectos de la ingeniería de software, así como producir conocimiento científico en el área.

CONTENIDOS MINIMOS:

- Principios del diseño orientado a objetos y lenguajes de modelado y programación orientados a objetos.
- Métodos ágiles. Testing y refactoring como prácticas esenciales de los métodos ágiles.
- Patrones de diseño.
- Mejora de la calidad interna del software

PROGRAMA

Programación Orientada a Objetos

- Principios del Diseño Orientado a Objetos. Reutilización de software.
- Definición de clases y responsabilidades.
- Herencia y polimorfismo. Binding dinámico.
- Relaciones de conocimiento y composición. Objetos contenedores.

Lenguajes de modelado orientados a objetos

- El lenguaje de Modelado Unificado (Unified Modeling Language).
- Diagramas de Clases.

Testing de Unidad.

- Factores que favorecen la aparición de errores.
- Comparación entre diferentes alcances en el testing: Unidad, Funcional, Regresión.
- Frameworks para implementar Test de Unidad. XUnit.
- Cobertura y buenas prácticas de Test de Unidad.
- Test-Driven Development

Patrones de diseño

- Introducción a patrones de diseño. Definición.
- Tipos de patrones de diseño de software.
- Catálogo “GoF” de patrones de diseño.

Refactoring.

- Definición.
- Análisis de los refactorings más importantes.
- Relación entre refactoring y patrones. Refactoring hacia patrones.

Métodos Ágiles.

- Precepto fundamental sobre el cambio de código.
- Elementos fundamentales de los métodos Ágiles: Usuarios, Testing, Integración Continua y Refactoring.

Calidad interna del software.

- Análisis de problemas de diseño y su solución mediante refactorings.
- Deuda técnica: costeo de rectificar los problemas

MODALIDAD DE EVALUACIÓN Y ACREDITACIÓN

La evaluación de la materia es a través de un trabajo práctico integrador con entregas parciales. Las entregas parciales están diseñadas con la función de poner rápidamente en práctica los conocimientos que se van abordando durante la cursada de la materia. El trabajo integrador involucra la totalidad de técnicas y herramientas vistas durante el curso de la materia. En el trabajo se utiliza JavaScript como lenguaje orientado a objetos, junto con diferentes herramientas o frameworks para cada objetivo de aprendizaje, como NodeJS (para programación de backend) o Jest (para programar tests de unidad).

La cursada se acreditará con la aprobación de la primera entrega parcial. La calificación final del alumno es en base al promedio de las 3 entregas.

El trabajo práctico se desarrolla en grupos de 2 o 3 alumnos.

RECURSOS Y MATERIALES DE ESTUDIO

El curso propone 13 encuentros sincrónicos en total. Éstos se realizan por videoconferencia, mediante el uso de la plataforma virtual de enseñanza Webex que permite diferentes maneras de interacción inmediata con los alumnos. Se requiere de asistencia a 10 de los 13 encuentros por VC (aproximadamente 75% de asistencia).

Los materiales de estudio son:

- Textos digitales: textos de lectura de referencia en en las temáticas tratadas durante el curso. Los textos de referencia son recuperados de la biblioteca del postgrado, revistas y/o de repositorios.
- Material preparado por los docentes del curso.
- Presentaciones digitales y materiales multimediales sobre el tema (de producción propia)

ACTIVIDADES EXPERIMENTALES Y DE INVESTIGACIÓN PLANIFICADAS PARA LA APROPIACIÓN DE LOS SABERES Y LA EVALUACIÓN

Actividades prácticas:

Desarrollo de trabajos prácticos parciales luego de cada eje temático de la materia. Estos trabajos conforman un único trabajo integrador, y se complementan con actividades de tipo taller durante las clases.

Actividad 1: luego del dictado de los primeros temas, se trabaja sobre un modelo para que los alumnos puedan realizar un diseño e implementarlo sobre la tecnología a utilizar en el curso (JavaScript) para empezar a familiarizarse con la misma. Se trabajará con diagramas UML. Programarán tests de unidad sobre su código, aplicando las buenas prácticas dadas en clase y preparando el campo para los temas siguientes como refactoring.

Actividad 2: luego de presentarse los contenidos patrones de diseño, se extenderá el modelo previamente trabajado para incorporar funcionalidad. Esta funcionalidad requerirá del descubrimiento e implementación de patrones de diseño clásicos. Se deberá explorar variantes de implementación y justificar su aplicación. Adicionalmente, se prepararán ejercicios puntuales sobre patrones para trabajar en clase, de manera de reforzar y fijar los contenidos.

Actividad 3: la última entrega del trabajo integrador se enfocará en refactoring. Se realizará un análisis de deuda técnica sobre lo entregado hasta el momento, evaluando los puntos que podrían traer problemas de mantenimiento en el corto o mediano plazo. Se priorizarán dichos problemas, y luego se elegirán los más importantes para realizar una tarea de refactoring. Esta actividad estará además sostenida por los tests de unidad realizados previamente.

Investigación:

Los alumnos analizarán artículos de investigación de la bibliografía complementaria con el objetivo de estudiar y comparar métodos ágiles y técnicas de programación orientada a objetos, y herramientas actuales de desarrollo, testing y refactoring. Estos artículos serán discutidos luego durante los encuentros sincrónicos y a través del espacio de foro de consultas de la plataforma virtual de enseñanza.

BIBLIOGRAFÍA BASICA

- Fowler, Martin. Refactoring: Improving the Design of Existing Code. Addison-Wesley, 2018.
- Rubin, K. Essential Scrum: A practical guide to the most popular Agile process. Addison Wesley. 2012.
- Antani, V., & Stefanov, S. Object-Oriented JavaScript. Packt Publishing Ltd. 2017.
- Timothy Budd. “An Introduction to Object-Oriented Programming”. Addisson-Wesley. (2008)
- Rebecca Wirfs-Brock. Object Design: Roles, Responsibilities, and Collaborations, Addisson-Wesley 2003.
- Gamma, Helm, Johnson, Vlissides. Design Patterns. Elements of Reusable Objects Oriented Software. Addison-Wesley, Professional Computing Series. 1995.
- Joshua Kerievsky. Refactoring to Patterns. Addison Wesley, 2004.
- Meszaros, G. xUnit test patterns: Refactoring test code. Pearson Education. 2007

BIBLIOGRAFÍA COMPLEMENTARIA

- Matt Weisfeld. The Object-Oriented Thought Process, Third Edition, Pearson Education, Addison Wesley. (2008), ISBN-13: 978-0-672-33016-2
- Rebecca Wirfs-Brock. Design for Test. IEEE Software 26(5): 92-93 (2009)
- Gregor Hohpe, Rebecca Wirfs-Brock, Joseph W. Yoder, Olaf Zimmermann: Twenty Years of Patterns' Impact. IEEE Software 30(6): 88 (2013)
- Rebecca Wirfs-Brock: Designing with an Agile Attitude. IEEE Software 26(2): 68-69 (2009)
- Rebecca Wirfs-Brock: Looking for Powerful Abstractions. IEEE Software 23(1): 13-15 (2006)
- Martin Fowler, Kendall Scott. “UML Distilled”. Addisson-Wesley. (2004)
- Kent Beck and Cynthia Andres. Extreme Programming Explained. Addison-Wesley, 2005.
- Crockford, D. Javascript: the good parts: the good parts. " O'Reilly Media, Inc.". 2008.
- Kent Beck. Simple Smalltalk Testing: With Patterns. http://www.xprogramming.com/testfram.htm
- Stephane Ducasse. SUnit Explained. http://www.iam.unibe.ch/~ducasse/
- Chen, R., & Miao, H. (2013, June). A selenium based approach to automatic test script generation for refactoring javascript code. In 2013 IEEE/ACIS 12th International Conference on Computer and Information Science (ICIS) (pp. 341-346). IEEE.
- Burchard, E. (2017). Refactoring JavaScript: Turning Bad Code Into Good Code. " O'Reilly Media, Inc.".
- Fard, A. M., & Mesbah, A. (2013, September). Jsnose: Detecting javascript code smells. In 2013 IEEE 13th International Working Conference on Source Code Analysis and Manipulation (SCAM) (pp. 116-125). IEEE.
- Silva, L. H., Valente, M. T., & Bergel, A. (2017, May). Refactoring legacy JavaScript code to use classes: The good, the bad and the ugly. In International Conference on Software Reuse (pp. 155-171). Springer, Cham.
- Alves, N. S., Mendes, T. S., de Mendonça, M. G., Spínola, R. O., Shull, F., & Seaman, C. “Identification and management of technical debt: A systematic mapping study”. Information and Software Technology 70, 100-121. 2016.
- Kruchten, P., Nord, R. L., & Ozkaya, I. “Technical debt: From metaphor to theory and practice”. Ieee Software 29(6), 18-21. 2012.
- Letouzey, JL. “The SQALE Method for Evaluating Technical Debt”. In Third International Workshop on Managing Technical Debt (MTD) (pp. 31-36). IEEE. 2012

- Li, Z., Avgeriou, P., & Liang, P. “A systematic mapping study on technical debt and its management”. Journal of Systems and Software, 101, 193-220. 2015.
- Lim, E.; Taksande, N. & Seaman, C. “A Balancing Act : What Software Practitioners Have to Say about Technical Debt”, IEEE Software 29 (6), 22-27. 2012.
- Díaz, O., & Arellano, C. (2015). The augmented web: rationales, opportunities, and challenges on browser-side transcoding. ACM Transactions on the Web (TWEB), 9(2), 1-30.
- José Matías Rivero, Matias Urbieta, Sergio Firmenich, Mauricio Witkin, Ramón Serrano, Viviana Elizabeth Cajas, Gustavo Rossi:Improving Legacy Applications with Client-Side Augmentations. ICWE 2018: 162-176