



FACULTAD DE INFORMÁTICA



UNIVERSIDAD
NACIONAL
DE LA PLATA

Especialización en Ingeniería de Software – Modalidad a distancia

Técnicas y Herramientas

Año 2021

Duración: 108 hs. Totales.

Cantidad de horas presenciales/VC: 30hs

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

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.



FACULTAD DE INFORMÁTICA



UNIVERSIDAD
NACIONAL
DE LA PLATA

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 orientados a objetos: v Maestriaarientes.

Lenguajes de modelado orientados a objetos

- El lenguaje de Modelado Unificado (Unified Modeling Language).
- Diagramas de Clases.
- Diagramas Dinámicos ó de Comportamiento: Diagramas de Interacción (Diagramas de Secuencia y Diagramas de Colaboración).

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.
- Validación en tiempo de ejecución.

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.

- El modelo de cascada tradicional.
- Procesos de desarrollo basados en prototipos.



FACULTAD DE INFORMÁTICA



UNIVERSIDAD
NACIONAL
DE LA PLATA

- Precepto fundamental sobre el cambio de código.
- Test Driven Development
- Elementos fundamentales de los métodos Ágiles: Usuarios, Testing, Integración Continua y Refactoring.
- Extreme Programming y Scrum.
- Herramientas de trabajo colaborativas que soporten las metodologías.

Calidad interna del software.

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

Aumentación Web.

- Introducción a la adaptación en aplicaciones Web
- Adaptación externa mediante aumentación Web
- Usos internos de adaptación Web

MODALIDAD DE EVALUACIÓN Y ACREDITACIÓN

La evaluación de la materia es a través de trabajos prácticos parciales y un trabajo práctico final integrador. La calificación final del alumno es en base al promedio de los mismos. Los trabajos prácticos parciales son de pequeña envergadura y tienen una función poner rápidamente en práctica los conocimientos que se van abordando. Estos son realizados durante la cursada de la materia. El trabajo final integrador involucra la totalidad de técnicas y herramientas vistas durante el curso de la materia, y tiene un plazo máximo de entrega de 3 meses luego de la última clase. Los trabajos prácticos se realizan usando el lenguaje de modelado UML y diferentes lenguajes de programación para su implementación, tales como JavaScript, Smalltalk, etc. que son los más apropiados de acuerdo a estos objetivos de aprendizaje/actualización.

Los trabajos prácticos pueden opcionalmente desarrollarse en grupos de 2 alumnos.

RECURSOS Y MATERIALES DE ESTUDIO

El curso propone 15 encuentros sincrónicos en total. De este total, 13 de ellos son por videoconferencia, mediante el uso de la plataforma virtual de enseñanza Big Blue Button que permite diferentes maneras de interacción inmediata con los alumnos. Se requiere 80% de asistencia a los encuentros por VC (10 encuentros). Los 2 encuentros restantes tienen carácter obligatorio y podrá ser presencial para los interesados.

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.



FACULTAD DE INFORMÁTICA



UNIVERSIDAD
NACIONAL
DE LA PLATA

- Material preparado por los docentes del curso.
- Presentaciones digitales y materiales multimediales sobre el tema (de producción propia)

ACTIVIDADES EXPERIMENTALES 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 están pensados para ser desarrollados en clases de taller, apuntando a una puesta en común.

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. Además, se trabajará con diagramas UML.

Actividad 2: luego de presentarse los contenidos sobre tests de unidad y la tecnología para programar y correr dichos tests en JavaScript, se realizará una actividad en la que los alumnos realizarán tests sobre código existente previamente utilizado en clase y también algunas funcionalidades con la metodología Tests-First.

Trabajo Final: se propone un trabajo final que abarca todos los temas trabajados durante las actividades prácticas intermedias. Se trata de un trabajo integrador donde se aplicarán patrones de diseño, técnicas ágiles de refactoring hacia patrones y test de unidad, en el contexto de artefactos de aumentación Web. Se tendrá en cuenta particularmente la calidad interna alcanzada en el artefacto desarrollado.

El plazo máximo de entrega del trabajo final es de 3 meses a partir de la finalización del cursado de la materia.

BIBLIOGRAFÍA BÁSICA

- 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". Addison-Wesley. (2008)
- Rebecca Wirfs-Brock. Object Design: Roles, Responsibilities, and Collaborations, Addison-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". Addison-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). Jsnoise: 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 Urbieto, Sergio Firmenich, Mauricio Witkin, Ramón Serrano, Viviana Elizabeth Cajas, Gustavo Rossi: *Improving Legacy Applications with Client-Side Augmentations*. *ICWE 2018*: 162-176