FACULTAD DE INFORMÁTICA

UNIVERSIDAD NACIONAL DE LA PLATA

TÉCNICAS Y HERRAMIENTAS Año 2020	**Carrera:** Maestría en Ingeniería de Software **Docentes Responsables:** Dr. Sergio Firmenich Dr. Matías Urbieta Dr. Julián Grigera **Duración:** 108 hs.

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.

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: variantes.

Lenguajes de modelado orientados a objetos
- Historia y variantes.
- 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).
- Diagramas de Casos de Uso.
- Uso de Casos de Uso para estimar esfuerzo.

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 módelo de cascada tradicional.
- Procesos de desarrollo basados en prototipos.
- 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.
- Analisis de problemas de diseño y su solución mediante refactorings.
- Deuda técnica: costeo de rectificar los problemas

ACTIVIDADES EXPERIMENTALES y DE INVESTIGACION

Actividades prácticas:

Desarrollo de trabajos prácticos parciales luego de cada eje temático de la materia.

Desarrollo de un trabajo final integrador que incluye conceptos de teoría y práctica.

Investigación:

Los alumnos analizarán papers 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.

METODOLOGIA DE EVALUACION

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 y aprobados 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 6 meses luego de la última clase. Los trabajos prácticos se realizan usando el lenguaje de modelado y diferentes lenguajes de programación para su implementación, tales como JavaScript, Smalltalk, etc. qué son los más apropiados de acuerdo a estos objetivos de aprendizaje/actualización.

Opcionalmente al trabajo final integrador, los alumnos podrán optar por un trabajo que requiera investigación en alguna de las temáticas abordadas durante el curso, sumado a una propuesta de aplicación de la misma.

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.

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.