
	UNIVERSIDAD NACIONAL DE ITAPUA – U.N.I. <i>Creada por Ley N°:1.009/96 del 03/12/96</i> Facultad de Ingeniería	
Programa de Estudios		

Materia:	Ingeniería de Software II	Semestre:	Octavo	
Ciclo:	Ingeniería Informática			
Código de la materia:	109			
Horas Semanales:	Teóricas:			4
	Prácticas:			-
	Laboratorio:			2
Horas Semestrales:	Teóricas:	68		
	Prácticas:	-		
	Laboratorio:	34		
Pre-Requisitos:	Ingeniería de Software I			

I- OBJETIVOS GENERALES:

Los objetivos de esta materia es potenciar en el alumno las capacidades de:

1. Aplicar los principios de la Ingeniería del Software a un proyecto concreto
2. Utilizar técnicas, procedimiento y métodos propios de la disciplina.

II. OBJETIVOS ESPECÍFICOS

Al término de este curso los alumnos deberán haber desarrollado las siguientes capacidades:

1. Organizar, planificar y analizar los riesgos
2. Gestionar las configuraciones
3. Considerar técnicas de verificación y control de errores
4. Utilizar técnicas de desarrollo de software
5. Métricas y métodos analíticos de análisis del proceso de desarrollo
6. Obtener información relevante sobre un proyecto de software para poder medirlo
7. Gestionar los cambios que se realizan por mantenimiento y evolución del sistema


III. CONTENIDOS PROGRAMÁTICOS:



Unidad I

Organización, Planificación y Análisis De Riesgos

Incluye cómo se suele organizar proyectos grupales, cómo se elige el personal adecuado, cómo se divide las responsabilidades y el trabajo. Luego incluye cómo se realiza un análisis de riesgos y cómo se monitorea el proyecto.

Esta unidad también cubre las herramientas de gestión que se utilizan para los proyectos de software, principalmente el sistema de gestión de prestaciones (también llamado sistema de gestión de defectos).

Aprobado por _____	Actualización No.: _____	 Sello y Firma	Página 1 de 5
Fecha: _____	Resolución No.: _____		
	Fecha: _____		

	UNIVERSIDAD NACIONAL DE ITAPUA – U.N.I. <i>Creada por Ley N°:1.009/96 del 03/12/96</i> Facultad de Ingeniería	
Programa de Estudios		

Ej: Issuezilla, Bugzilla, etc.), y el sistema de control de código fuente. Pero también podría incluir herramientas como Microsoft Project, etc.

Unidad II

Gestión de Configuraciones (Configuration management)

Esta unidad se enfoca en las técnicas de gestión de configuraciones que aseguran la seguridad del código fuente, la reproducibilidad de la construcción del software, y la formalización del proceso de construcción de software.

Las herramientas principales que se utilizan para esta unidad debería incluir alguna variación de un Makefile (Ej: make, nmake, ant, Apache Avalon, etc.), y un sistema de control de código fuente.

En esta unidad también se pueden cubrir las implicaciones que estos procesos tienen en la certificación ISO 9001 y CMM (Capability Maturity Model).

Unidad III

Técnicas de verificación y control de errores



En esta sección se profundizará la noción de testeo de software. El fin será explicar al alumno que la responsabilidad es compartida por todo el equipo (no solamente el departamento de control de calidad) y cómo hacen todos los participantes para detectar errores en el sistema.

Cubre los siguientes puntos:

1. Conceptos sobre el objetivo de testear
2. La imposibilidad del testeo completo y el equilibrio entre el costo y el beneficio
3. Consideraciones de la Organización para asegurar el testeo eficaz: independencia de la organización de testeo de la de desarrollo, cooperación vs. rivalidad entre organizaciones, igualdad de rango entre programador y testeador vs. supremacía de uno sobre el otro (cooperación vs. competencia), etc.
4. La mentalidad del tester: mentalidad “hacker”, mentalidad del “usuario bobo”, testeo con caja negra, testeo con caja blanca, etc.
5. Tipos de testeo: por unidad, por integración, testeo de performance, testeo de interfaz de usuario, testeo de facilidad de uso (usability testing), testeo de seguridad, etc..
6. Técnicas de verificación para el programador: revisiones de diseño, testeo por unidad automatizado (Ej: con JUnit), testeo con el depurador, revisiones de código, inspecciones de código, testeo grupal, etc.
7. Técnicas de verificación para el tester: testeo de integración, testeo automatizado, etc.

Para esta unidad algunas herramientas útiles pueden ser un framework de desarrollo de tests automatizados por unidad (Ej: JUnit) y una herramienta de automatización de testeo de interfaz de usuario (Ej: Microsoft Test, java.awt.Robot, etc.)

Aprobado por _____ Fecha: _____	Actualización No.: _____ Resolución No.: _____ Fecha: _____	_____ Sello y Firma	Página 2 de 5
--	---	------------------------	----------------------

	UNIVERSIDAD NACIONAL DE ITAPUA – U.N.I. <i>Creada por Ley N°:1.009/96 del 03/12/96</i> Facultad de Ingeniería	
Programa de Estudios		

Unidad IV

Técnicas de desarrollo

Esta unidad se enfoca en técnicas de desarrollo que pueden utilizar los programadores para desarrollar programas con menos errores, más mantenibles, de mejor diseño, y en menor tiempo.

Algunas técnicas importantes incluyen:

1. Gestión formal del tiempo del programador a través de un programa de gestión de prestaciones (Ej: Issuezilla, Bugzilla)
2. Revisión de diseño (Design reviews) formal e informal
3. Programación en pareja (Pair Programming)
4. Programación dirigida por tests (Test driven development)
5. El uso de un depurador para verificar el software al desarrollarlo y otras técnicas de verificación de errores
6. El uso de logs (Ej: log4j) para determinar rápidamente la causa de errores
7. Programación defensiva con ASSERT (asertos) tanto en modalidad de depuración como en modalidad de producción
8. Técnicas avanzadas de manejo de errores
9. Convenciones de programación
10. Técnicas específicas para lenguajes comunes (Ej: C, C++, Java, etc.), que son aplicables para todo lenguaje

Unidad V

Métricas y métodos analíticos de análisis del proceso de desarrollo

El objetivo de esta unidad es aprender a obtener información relevante sobre un proyecto de software para poder medirlo de alguna forma con el fin de ir mejorándolo continuamente. Es importante que el alumno entienda las debilidades además de los beneficios de

Algunas métricas comunes para proyectos [de Wiegers]:



1. Tamaño de código: líneas de código, número de funciones, número de clases, número de requisitos, o número de elementos GUI
2. Tiempo estimado y tiempo real (tiempo calendario) y esfuerzo (horas de trabajo): registrar por tareas individuales, etapas del proyecto, y sobre proyecto en general.
3. Distribución de esfuerzo: grabar el tiempo que demoró cada actividad de desarrollo (gestión del proyecto, especificación de requisitos, diseño, programación, testeó) y actividades de mantenimiento (adaptativo, perfeccionamiento, correcciones)
4. Defectos: contar número encontrado al testear, por cliente y por su tipo, severidad y estatus (abierto o cerrado).

Unidad VI

Evolución y Mantenimiento Efectivo de Software

Esta unidad reconoce que el software es siempre cambiante. Aquí se hablará sobre la gestión de los cambios que se realizan por mantenimiento y evolución del sistema para que el sistema siga siendo

Aprobado por _____ Fecha: _____	Actualización No.: _____ Resolución No.: _____ Fecha: _____	_____ Sello y Firma	Página 3 de 5
--	---	------------------------	---------------

	UNIVERSIDAD NACIONAL DE ITAPUA – U.N.I. <i>Creada por Ley N°:1.009/96 del 03/12/96</i> Facultad de Ingeniería	
Programa de Estudios		

relevante en un futuro. Idealmente, el instructor podría dar un ejemplo de un ciclo de vida de un proyecto hasta su muerte e indicar los casos prácticos que causaron que el costo de mantenimiento exceda el costo de reimplementación.

Proyecto:

Paralelo al estudio de la teoría de la Ingeniería de Software y los ejercicios que eso pueda implicar, se realizará un proyecto grande que abarque las materias de Ingeniería de Software I y II en la que los alumnos practican los conceptos mientras los van aprendiendo culminando en un producto terminado. El enfoque será de trabajar efectivamente en grupo realizando un sistema un poco más complejo de lo que hayan visto anteriormente en la universidad. El proyecto requerirá de trabajo grupal para completarse.

Durante el primer semestre de Ingeniería de Software los alumnos aplicarán una metodología de desarrollo mientras lo aprenden para lograr planificar, desarrollar, codificar, testear, y medir un proyecto de software.

A lo mínimo el proyecto debería requerir el uso de un sistema de gestión de prestaciones (Ej: Bugzilla, Issuezilla, etc.), para coordinar y un sistema de control de versiones (Ej: CVS, Perforce, SourceSafe) para que varios alumnos puedan trabajar en un mismo archivo a la vez.

El proyecto debe ser una parte importante de los trabajos parciales para este semestre.

IV. METODOLOGÍA

Las actividades de los alumnos comprenderán:

- Clases teóricas
- Estudio de casos
- Prácticas supervisadas en laboratorio
- Elaboración de Trabajos prácticos
- Elaboración y presentación de trabajos
- Investigaciones

V. CRITERIOS DE EVALUACIÓN

Conforme al Reglamento Académico y Reglamento de Cátedra vigentes.

VI. BIBLIOGRAFÍA:



DeMarco, T., & Lister, T. (2013). Peopleware: productive projects and teams. Addison-Wesley.

Fowler, M. (1999). Refactoring: improving the design of existing code. Pearson Education India.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design patterns: elements of reusable object-oriented software. Pearson Education.

Fuggetta, A., & Ghezzi, C. (1994). State of the art and open issues in process-centered software engineering environments. Journal of Systems and Software,26(1), 53-60.

Aprobado por _____ Fecha: _____	Actualización No.: _____ Resolución No.: _____ Fecha: _____	_____ Sello y Firma	Página 4 de 5
--	---	------------------------	---------------

	UNIVERSIDAD NACIONAL DE ITAPUA – U.N.I. <i>Creada por Ley N°:1.009/96 del 03/12/96</i> Facultad de Ingeniería	
Programa de Estudios		

Kendall, K. E., & Kendall, J. E. (2005). Análisis y diseño de sistemas. Pearson educación.

Krasteva, I., & Ilieva, S. (2008, May). Adopting an agile methodology: why it did not work. In Proceedings of the 2008 international workshop on Scrutinizing agile practices or shoot-out at the agile corral (pp. 33-36). ACM.

Maguire, S. (1994). Debugging the development process. Microsoft Press.

Paulk, M. C. (2002). Agile methodologies and process discipline. Institute for Software Research, 3.

Pressman, R. S. Mc Grah-Hill, 7a. edición, 2010. Ingeniería del software, un enfoque práctico.

University of Illinois at Urbana-Champaign, College of Engineering, Department of Computer Science."Computer Science 327". 12/01/01. <http://www.cs.uiuc.edu/education/courses/cs327.html>

University of Illinois at Urbana-Champaign, College of Engineering, Department of Computer Science."Computer Science 329". 12/01/01. <http://www.cs.uiuc.edu/education/courses/cs329.html>

University of Washington, Department of Computer Science and Engineering. CSE403: Software Engineering. Winter 2004. <http://www.cs.washington.edu/education/courses/403>

University of Washington, Department of Computer Science and Engineering. CSE403: Software Engineering. Spring 2003. <http://www.cs.washington.edu/education/courses/403>

Yourdon, E., & Armitage, A. T. (1993). Análisis estructurado moderno (Vol. 5). Prentice hall.

Aprobado por _____ Fecha: _____	Actualización No.: _____ Resolución No.: _____ Fecha: _____	_____ Sello y Firma	Página 5 de 5
--	---	------------------------	---------------