Guía docente de la asignatura

Asignatura	**Técnicas en el desarrollo y mantenimiento de Software para incrementar la calidad.**		
Materia	AUDITARÍA, CALIDAD Y SEGURIDAD		
Módulo	TECNOLOGÍAS INFORMÁTICAS		
Titulación	MÁSTER EN INGENIERÍA INFORMÁTICA		
Plan	510	**Código**	53176
Periodo de impartición	2º CUATRIMESTRE	**Tipo/Carácter**	OPTATIVA
Nivel/Ciclo	MÁSTER	**Curso**	1º
Créditos ECTS	3 ECTS		
Lengua en que se imparte	CASTELLANO		
Profesor/es responsable/s	Yania Crespo González-Carvajal		
Datos de contacto (E-mail, teléfono…)	TELÉFONO: 983 185695 E-MAIL: yania@infor.uva.es		
Horario de tutorías	Véase www.uva.es → Centros → Campus de Valladolid → Escuela Técnica Superior de Ingeniería Informática → Tutorías		
Departamento	DEPARTAMENTO DE INFORMÁTICA		

1. Situación / Sentido de la Asignatura

1.1 Contextualización

Esta asignatura forma parte de la materia “Auditoría, Calidad y Seguridad”. La fabricación de sistemas informáticos de **calidad** de forma **económicamente competitiva** sigue siendo hoy en día un reto fundamental de la Ingeniería del Software. El desarrollo y evolución cada vez más rápido de sistemas cada vez más complejos requiere de herramientas, técnicas y métodos de diseño que resuelvan los problemas de calidad, fiabilidad, escalabilidad o mantenimiento del software.

Las características de calidad relacionadas con la **reusabilidad**, **adaptabilidad** y **mantenibilidad** del software son cruciales para la eficiencia de los procesos de desarrollo y mantenimiento de sistemas software. Los sistemas que se requiere construir son altamente cambiantes y las aplicaciones construidas deben estar preparadas para poder ser adaptadas con facilidad. La construcción eficiente de aplicaciones no se concibe actualmente si no se realiza descansando en desarrollos realizados previamente (desarrollar con reutilización). De ahí la importancia de dotar al diseño de un sistema de las características que lo hacen potencialmente reutilizable. Por otra parte, casi el 80% del tiempo/hombre empleado en la industria informática se invierte en el mantenimiento de sistemas, tanto en tareas de eliminación de defectos, como en tareas de adaptación. El diseño que subyace en el sistema software construido debe responder a características que lo hagan más mantenible. Las características fundamentales relacionadas con la mantenibilidad coinciden con las características que garantizan reusabilidad y adaptabilidad. En general, el software debe ser fácil de entender y estar preparado para el cambio y la extensión.

El soporte teórico y operativo para lograr dotar de las características de calidad mencionadas previamente podríamos dividirlo fundamentalmente en dos enfoques: el enfoque **a priori (o sistemático)** y el enfoque **a posteriori (o transformacional)**. Estos enfoques no son excluyentes porque cada uno ofrece ventajas difícilmente suplidas por el otro. El **enfoque sistemático** se propone dotar a la organización de los métodos y herramientas necesarios para producir software de calidad. En ese caso encontramos trabajos sobre estructuras, clasificación, almacenamiento y recuperación en repositorios de componentes reutilizables, métodos para desarrollar los elementos del software con reutilización y para reutilización, guías de diseño, etc. La incorporación de este enfoque sistemático en una organización es muy importante. Sin embargo, el software es altamente cambiante, y el cambio no siempre se puede realizar de forma controlada, los desarrolladores no son infalibles y el tiempo, que es oro, no siempre se puede consumir en hacer seguir todas las indicaciones que dotarían a un sistema de las características de calidad deseadas. Esto hace muy necesario contar con métodos y herramientas que permitan, a partir de un software construido, detectar los problemas que tiene, en el sentido mencionado (**detección de defectos**), y transformar (**refactorizar**) dicho software para hacerlo cumplir con las indicaciones. En esto consiste el **enfoque transformacional**.

En el enfoque transformacional es vital la utilización de herramientas que realicen las operaciones de diagnóstico y refactorización de forma automatizada. Estas tareas son complejas, en cuanto a tiempo y dificultad, y propensas a errores. Precisamente queremos introducir eficiencia en el proceso y minimizar el tiempo de desarrollo y mantenimiento. La refactorización automatizada es uno de los soportes fundamentales para lo que se ha dado en llamar “desarrollo ágil de aplicaciones” (*Agile software development*).

1.2 Relación con otras materias

La asignatura utiliza conceptos básicos de:

Ingeniería del Software

Programación

Calidad del Software

1.3 Prerrequisitos

Es recomendable que el alumno:

Conozca y sepa utilizar conceptos básicos sobre qué es un proceso de desarrollo del software

Conozca y sepa utilizar principios básicos de diseño, programación y pruebas de software

Conozca técnicas básicas de Calidad del Software, medición y métricas

2. Competencias

2.1 Generales

Código	Descripción
CG1	Capacidad para proyectar, calcular y diseñar productos, procesos e instalaciones en todos los ámbitos de la ingeniería informática.
CG8	Capacidad para la aplicación de los conocimientos adquiridos y de resolver problemas en entornos nuevos o poco conocidos dentro de contextos más amplios y multidisciplinares, siendo capaces de integrar estos conocimientos

2.2 Específicas

Código	Descripción
CET3	Capacidad para asegurar, gestionar, auditar y certificar la calidad de los desarrollos, procesos, sistemas, servicios, aplicaciones y productos informáticos.

2.3 Transversales

Código	Descripción
CT2	Capacidad para trabajar bajo presión.
CT3	Capacidad para afrontar tareas y situaciones críticas.
CT5	Conocimiento de otras lenguas, sobre todo la inglesa.
CT6	Capacidad de trabajo autónomo y toma de decisiones.
CT7	Capacidades asociadas al trabajo en equipo: cooperación, liderazgo, saber escuchar.
CT8	Capacidad analítica, crítica y de síntesis.
CT10	Capacidad de adaptación a situaciones cambiantes. Flexibilidad. Predisposición al cambio.
CT13	Motivación por la calidad.

3. Objetivos

Código	Descripción
DEF1	Conocer catálogos de defectos, antipatrones y malas prácticas en el código o el diseños.
DEF2	Conocer técnicas y herramientas de defección de defectos..
DEF3	Conocer las técnicas de refactorización de software
DEF4	Saber aplicar refactorizaciones para corregir o mitigar defectos en el código o los diseños.
DEF5	Ser capaz de evitar durante el desarrollo de software la aparición de defectos, antipatrones malas prácticas en el código y los diseños.
DEF6	Ser capaz de corregir o minimizar la presencia de defectos de diseño, antipatrones y malas prácticas durante el mantenimiento y/o evolución del software

4. Tabla de dedicación del estudiante a la asignatura

ACTIVIDADES PRESENCIALES	HORAS	ACTIVIDADES NO PRESENCIALES	HORAS
Clases teórico-prácticas (T/M)	10	Estudio y trabajo autónomo individual	20
Clases prácticas de aula (A)		Estudio y trabajo autónomo grupal	25
Laboratorios (L)	12		
Prácticas externas, clínicas o de campo			
Seminarios (S)	4		
Tutorías grupales (TG)			
Evaluación (fuera del periodo oficial de exámenes)	4		
Total presencial	30	**Total no presencial**	45

5. Bloques temáticos

Bloque 1: Detección de defectos, refactorización y reingeniería

Carga de trabajo en créditos ECTS: 3

a. Contextualización y justificación

La asignatura se desarrolla en un solo bloque.

b. Objetivos de aprendizaje

Código	Descripción
DEF1	Conocer catálogos de defectos, antipatrones y malas prácticas en el código o el diseños.
DEF2	Conocer técnicas y herramientas de defección de defectos.
DEF3	Conocer las técnicas de refactorización de software
DEF4	Saber aplicar refactorizaciones para corregir o mitigar defectos en el código o los diseños.
DEF5	Ser capaz de evitar durante el desarrollo de software la aparición de defectos, antipatrones malas prácticas en el código y los diseños.

DEF6	Ser capaz de corregir o minimizar la presencia de defectos de diseño, antipatrones y malas prácticas durante el mantenimiento y/o evolución del software

c. Contenidos

Tema 1 Heurísticas, Antipatrones y Defectos de Diseño

Tema 2 Refactoring

Tema 3 Refactoring y Patrones

Tema 4 Mantenimiento y Reingeniería

d. Métodos docentes

Actividad	Metodología
Clase de teoría	Clase magistral participativa Estudio de casos en aula
Clase práctica	Realización de trabajos prácticos guiados por el profesor.
Seminarios	Talleres de aprendizaje

e. Plan de trabajo

En esta asignatura se deberá desarrollará un trabajo práctico que consistirá en:

Detectar defectos en un sistema software, por inspección y utilizando una herramienta. (entrega 1)

Refactorizar un sistema software para corregir defectos

Refactorizar un sistema software para aplicar patrones (entrega 2)

Planificar y ejecutar la reingeniería de un sistema software (entrega 3)

Semana	Actividades a realizar
1	Presentación de la asignatura. Tema 1: Heurísticas, Antipatrones y Defectos de Diseño. Introducción a las herramientas del laboratorio. Formación de los grupos de trabajo del laboratorio.
2	Tema 1: Heurísticas, Antipatrones y Defectos de Diseño. (2ª parte). Trabajo de los grupos del laboratorio
3	Tema 2: Refactoring Trabajo de los grupos del laboratorio. Primera entrega de la práctica
4	Tema 2: Refactoring (2ª parte). Seminarios 1 y 2: Herramientas de defección de defectos
5	Tema 3: Refactorización y patrones.

	Trabajo de los grupos del laboratorio.
6	Tema 3: Refactorización y patrones (2ª parte). Trabajo de los grupos del laboratorio. Segunda entrega de la práctica
7	Tema 4: Mantenimiento y Reingeniería Seminarios 3 y 4: Herramientas, catálogos, big refactoring.
8	Tema 4: Mantenimiento y Reingeniería (2ª parte). Trabajo de los grupos del laboratorio. Tercera entrega de la práctica

f. Evaluación

Ver apartado 7.

g. Bibliografía básica

Lanza, M. & Marinescu, R. “Object-Oriented Metrics in Practice. Using Software Metrics to Characterize, Evaluate and Improve the Design of Object-Oriented Systems”, Springer, 2006

Fowler, M., “Refactoring. Improving the design of existing code”. Addison Wesley, 1999

Demeyer, S., Ducasse, S & Niertrasz, O. “Object-Oriented Reegineering Patterns”. Morgan Kaufmann, 2003

h. Bibliografía complementaria

Riel, A. “Object-Oriented Design Heuristics”, Addison Wesley, 1996

Brown, W y otros, “Antipatterns. Refactoring Software, Architectures and Projects in Crisis”, Wiley, 1998

Wake, W. “Refactoring Workbook”, Addison Wesley, 2003

Kerievsky, J. “Refactoring to patterns”, Addison Wesley, 2005

i. Recursos necesarios

El alumno deberá tener acceso a un ordenador personal para trabajo individual no presencial.

Aula virtual de la asignatura.

Documentación en el aula virtual de la asignatura (esquemas, artículos, ejercicios y supuestos etc.)

6. Temporalización (por bloques temáticos)

BLOQUE TEMÁTICO	CARGA ECTS	PERIODO PREVISTO DE DESARROLLO
Detección de defectos, refactorización y reingeniería	3 ECTS	Semanas 1 a 8

7. Sistema de calificaciones – Tabla resumen

INSTRUMENTO/PROCEDIMIENTO	PESO EN LA NOTA FINAL	OBSERVACIONES
Primera entrega práctica	15,00%	Semana 3
Segunda entrega práctica	20,00%	Semana 6
Tercera entrega práctica	25,00%	Semana 8
Presentación en Seminario	10,00%	Semana 4, Semana 7
Examen final escrito	30,00%	Periodo de exámenes

CRITERIOS DE CALIFICACIÓN
Convocatoria ordinaria: Se necesitará obtener un 3,5/10 en el examen escrito para hacer media con el resto de apartados. Se necesitará obtener un 5/10 en el trabajo práctico para hacer media con el resto de apartados. **Convocatoria extraordinaria:** En esta convocatoria se conservará la nota del examen escrito (si es mayor que 3,5/10) y la nota de las prácticas (si es mayor que 5/10). Se realizará un examen escrito que supondrá el 35% de la nota. Se necesitará sacar un 3,5/10 en este examen para superar la asignatura. Podrá volverse a realizar el o los encargos prácticos que decida el alumno. Se necesitará sacar un 5/10 en el trabajo práctico para superar la asignatura. Estos encargos prácticos constituyen en su totalidad el 55% de la nota. El 10% restante se corresponde con la presentación en seminarios no podrá volverse a obtener en convocatoria extraordinaria.