Actas XVIII JENUI 2012, Ciudad Real, 10-13 de julio 2012
I.S.B.N. 10: 84-615-7157-6 | I.S.B.N. 13:978-84-615-7157-4
Páginas 121-128

Evaluación y seguimiento de trabajos en equipo de desarrollo de software a través de la calidad del código fuente

Pablo Trinidad, Manuel Resinas, Sergio Segura, Antonio Ruiz-Cortés
Universidad de Sevilla
{ptrinidad, resinas, sergiosegura, aruiz}@us.es

Resumen

La enseñanza de buenas prácticas en el diseño e implementación de sistemas software es una tarea que exige al profesorado una docencia personalizada y un seguimiento continuo del aprendizaje. En este artículo presentamos nuestra experiencia en la implantación de una plataforma de evaluación de calidad del software para que estudiantes y profesores puedan obtener métricas objetivas de calidad y esfuerzo sobre los trabajos desarrollados. Para ello se ha utilizado la plataforma Sonar, que permite obtener información sobre la calidad de varios proyectos de desarrollo de software al mismo tiempo. Esta plataforma se ha integrado dentro de un sistema de gestión del ciclo de vida de aplicaciones (ALM) para que forme parte del día a día del estudiante. Esto nos ha permitido realizar un seguimiento detallado con poco esfuerzo y definir rúbricas de evaluación con términos objetivos y conocidos a priori por los estudiantes.

Summary

Teaching good practices in software design and implementation is a demanding task that requires a close evaluation of students' work. In this paper, we describe our experience in setting up a platform for software quality evaluation providing lectures and students with objective metrics about the software projects developed. We use the Sonar platform to track the quality of several software projects at the same time. This platform is integrated into an Application Lifecyle Management (ALM) environment so the students can use it in their everyday work. It has reduced the effort to evaluate the students' work by using objective and well defined criteria known by the students beforehand.

Palabras clave

Ingeniería del Software, Aprendizaje cooperativo, Diseño de software, TIC

1. Contexto

En la ingeniería del software, un diseño de calidad debe cumplir unos principios básicos que buscan mejorar factores a veces contrapuestos como la cohesión o el acoplamiento. Esto impide optimizar todos los factores al mismo tiempo y por tanto debe encontrarse una solución equilibrada que esté preparada para evolucionar en el tiempo.

Un estudiante habrá aprendido a realizar diseños de calidad cuando sea capaz de analizar un problema, proponer soluciones, criticarlas y tomar decisiones. Estas habilidades sólo se pueden desarrollar a través de la práctica por lo que la enseñanza del diseño de software debe orientarse en este sentido.

Esta experiencia se ha desarrollado para la asignatura ingeniería del software de gestión 2, con 6 créditos impartidos en el tercer curso de ingeniería en informática técnica de gestión. Para centrar el mayor esfuerzo posible en los conceptos prácticos, el cuerpo teórico se ha reducido a la enseñanza de los principios básicos del diseño en las primeras semanas del cuatrimestre. El resto del esfuerzo docente se centra en el seguimiento y evaluación de un trabajo práctico en grupo y de cierta envergadura. En este trabajo, los estudiantes deben diseñar e implementar un juego de mesa con licencia *print and play* [7] facilitado por el profesorado.

A cada grupo se le asigna un profesor-tutor que realiza un seguimiento semanal en horario de clase. Debido al elevado número de estudiantes, el seguimiento tiene una duración aproximada de 20 minutos por semana y grupo. En este tiempo el profesor debe ser capaz de contextualizarse, detectar y analizar el trabajo realizado por el grupo desde la última revisión, detectar puntos débiles y fuertes del diseño, proponer alternativas a través de otros puntos de vista, detectar factores no analizados y conocer y orientar las acciones siguientes que realizará el grupo para la siguiente sesión de seguimiento.

En el curso pasado, la dificultad de realizar todo este trabajo en tan poco tiempo provocaba que

la atención en horario de tutorías fuera muy demandada, llegando en las entregas finales a superar la demanda a las horas ofertadas por el profesorado. Por tanto, aunque consideramos que el enfoque docente era adecuado, implicaba un consumo excesivo de recursos docentes. Además, en torno a la práctica grupal detectamos dos problemas adicionales. En primer lugar, el baremo a utilizar en la calificación de los trabajos no estaba claramente definido. En segundo lugar, a cada trabajo se le presupone una complejidad que afectará al número de horas a invertir por parte del alumnado y por ende a la ponderación en la calificación final. En ambos casos, la valoración la realiza el tutor en base a criterios subjetivos y difícilmente cuantificables. Esto dificulta la homogeneidad en la calificación entre los distintos tutores y la falta de certeza en la valoración de la complejidad real de los trabajos.

En base a los problemas anteriormente definidos nos planteamos reestructurar la docencia de la asignatura con objeto de cumplir los siguientes objetivos de cara al curso 2011/2012:

1. Reducir la carga docente por el seguimiento de los trabajos fuera del horario de clase. Para ello se debe aumentar la autonomía del alumno en el aprendizaje del diseño de software mediante el estudio y seguimiento de métricas de calidad del software.
2. Homogeneizar criterios de evaluación entre los profesores mediante el uso de criterios objetivos y cuantificables.
3. Mejorar la atención del profesor en las revisiones semanales de grupos.
4. Controlar que el análisis de dificultad previo por parte del profesor se corresponde con la dificultad real del problema a resolver.

En este artículo, presentamos una plataforma de trabajo y evaluación que ha contribuido a lograr los objetivos anteriores en la Sección 3 dentro del marco metodológico descrito en la Sección 2. En la Sección 4, se indica cómo esta plataforma ha permitido mejorar la docencia logrando aumentar la autonomía del estudiante, definiendo rúbricas de evaluación [2] y facilitando el seguimiento semanal de los trabajos en equipo. En la Sección 5 analizamos los resultados obtenidos durante este curso académico y extraemos algunas conclusiones. Por último en la Sección 6 discutimos brevemente la posible aplicación de la plataforma de trabajo en otros contextos.

2. Mecanismos de Evaluación

Al comienzo del curso, cada estudiante es responsable de firmar un contrato de aprendizaje [5] que defina la ponderación de cada uno de los tres aspectos evaluables: trabajo en equipo, prueba de conocimiento general y porfolios. La ponderación afecta al número de horas que se estima que cada estudiante debe invertir en cada una de las partes. Al tratarse de una asignatura de 6 créditos ECTS, el estudiante debe realizar un esfuerzo de 150 a 180 horas entre clases y estudio.

El trabajo en equipo es la columna vertebral de esta asignatura asignándose un porcentaje entre el 50 % y el 80 % de la nota final. La ponderación indica la dificultad estimada del trabajo propuesto por el profesorado y por ende el número de horas que deberá invertir el grupo para realizarlo. Así un porcentaje del 50 % implica que el estudiante debe invertir entre 75 y 90 horas en la realización del trabajo.

Los trabajos se realizan en grupos de 5 a 7 personas que deben haber elegido la misma ponderación para el trabajo. Aunque el trabajo se evalúa en su conjunto, la calificación final es individual. Esto se implementa mediante un sistema de recompensas y castigos definido en [7] en el que el grupo asigna en cada entrega una calificación por estudiante en función del trabajo realizado.

El trabajo consiste en el diseño e implementación de un juego de mesa. Se definen dos entregables: una memoria y el código fuente. La memoria debe contener el diseño realizado en cada entrega así como todas las decisiones tomadas al respecto y sus justificaciones. El código fuente debe residir en un sistema de control de versiones que forma parte de la plataforma de trabajo y que utiliza Sonar para su análisis.

La evaluación final del trabajo en equipo se realiza en una sesión pública en la que deben presentarse los resultados obtenidos. En esta sesión el tutor debe emitir una calificación en la que se evalúen los entregables del trabajo y la propia presentación.

La prueba de conocimiento general significan entre el 10 % y el 40 % de la calificación final. Cada ponderación implica la realización de una prueba diferente en cada caso y acorde al número de horas que se deben invertir en el estudio. De esta forma se realizan pruebas orales, informes técnicos o exámenes

de diseño tradicionales según el caso.

Por último, los porfolios consisten en la realización de trabajos individuales a propuesta del estudiante o el profesor con un peso de entre el 10 % y el 30 % de la calificación final.

3. Plataforma de Trabajo y Evaluación

Un entorno ALM (Application Lifecycle Management) es un conjunto integrado de herramientas que dan soporte al desarrollo y mantenimiento de aplicaciones software a lo largo de su ciclo de vida [6]. Las principales características asociadas a este tipo de entorno son la automatización de procesos, la monitorización del desarrollo y la trazabilidad entre los distintos artefactos software. La Figura 1 ilustra el entorno ALM empleado en nuestra propuesta. Sus principales componentes son:

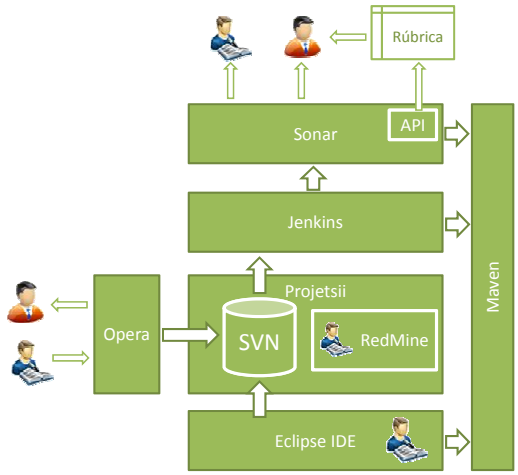

Figura 1: Estructura del ALM

Sonar: Evaluación de la calidad. Sonar [1] es una plataforma para la evaluación de la calidad del software a través del código fuente. La plataforma integra un conjunto de herramientas de análisis estático del código fuente para extraer métricas que permitan evaluar y mejorar la calidad del software. Entre otros, Sonar proporciona información sobre tamaño del código, métricas para medir la cohesión y el acoplamiento, errores potenciales, violaciones de estándares de codificación y código duplicado. Además, Sonar permite la conexión con herramientas de integración continua como Jenkins proporcionando así información sobre el resultado de las pruebas unitarias y la cobertura de código. En la asignatura, se empleó una instalación de Sonar para Java conectada a Jenkins que permitía a profesores y alumnos tener información constantemente actualizada sobre la calidad del código fuente y el resultado de las pruebas unitarias en cada proyecto. La Figura 2 muestra el cuadro de mando de Sonar con información sobre uno de los proyectos de la asignatura.

Subversion: Control de versiones. Subversion (SVN) [2] es un sistema de control de versiones ampliamente usado por la comunidad de software libre. Este sistema permite centralizar el almacenamiento del código fuente de un proyecto, registrando los cambios realizados en el mismo y permitiendo volver a versiones anteriores si es necesario. Además, la herramienta da soporte al trabajo concurrente sobre los ficheros de código aumentando así la flexibilidad y productividad en el desarrollo. En la asignatura, todos los grupos contaban con un repositorio SVN para la gestión del código fuente. Concretamente, se hizo uso de ProjETSII [3], una aplicación Web para la gestión de proyectos que integra Subversion y Redmine, gestionada por la E.T.S. de Ingeniería Informática de Sevilla.

Jenkins: Integración continua. Jenkins [4] es un software de integración continua de código abierto. La integración continua consiste en hacer integraciones frecuentes y automáticas de un proyecto para detectar errores lo antes posible. En la asignatura, Jenkins fue configurado para descargar el código fuente del repositorio SVN de cada grupo con una frecuencia de 24 horas. En cada iteración, la herramienta se encargaba de compilar el código fuente, ejecutar las pruebas Junit contenidas en el proyecto y generar informes con los resultados.

Maven: Gestión de la construcción. Maven [5] es una herramienta software para la configuración y construcción de proyectos Java. Para ello, Maven se basa en el uso de documentos XML que permiten la automatización de tareas habituales como la compilación, ejecución de pruebas, resolución de dependencias y despliegue de aplicaciones. En la asigna-

[1] www.sonarsource.org

[2] subversion.apache.org

[3] projetsii.informatica.us.es

[4] http://jenkins-ci.org/

[5] maven.apache.org

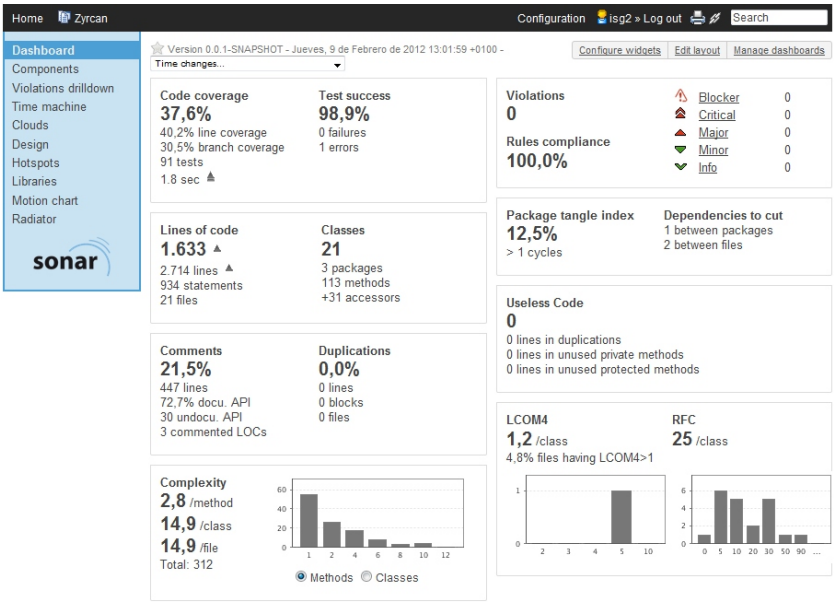

Figura 2: Cuadro de mando de Sonar con información sobre unos de los proyectos de la asignatura

tura, Maven fue empleado para definir la estructura de código fuente de los proyectos desarrollados por los grupos de alumnos permitiendo así su integración directa en Jenkins y Sonar.

Eclipse: Entorno de desarrollo de software. Eclipse [6] es un entorno de desarrollo con una arquitectura basada en *plugins*. En el contexto de nuestra asignatura, Eclipse se ha usado como entorno de desarrollo Java y elemento integrador con el resto de herramientas del ALM. Eclipse ya integra en sus últimas versiones con SVN y Maven. Para la integración con Sonar, se hizo uso del plugin Sonar Eclipse.

Redmine: Gestión de proyectos. Redmine [7] es una herramienta de código abierto para la gestión de proyectos que permite la gestión de tareas, wikis, calendarios y noticias entre otros. En la asignatura, esta herramienta fue empleada para dar soporte al trabajo colaborativo del alumnado quienes lo usaron fundamentalmente para la asignación y seguimiento de tareas. El profesorado tenía acceso a todos los proyectos creados dando así la posibilidad de evaluar el funcionamiento del grupo a nivel de tareas y roles.

Opera: Publicación de resultados. Opera [3] es una aplicación Web para la gestión de trabajos en grupo desarrollada en el contexto de la E.T.S.I. Informática de Sevilla. La aplicación permite a los alumnos proponer trabajos y formar grupos de forma sencilla. Además, actúa como una red social donde cada trabajo tiene su propia página con información descriptiva y enlaces así como comentarios y votaciones de otros alumnos. Se ha utilizado Opera para gestionar los grupos de trabajos y sus entregables.

4. Mejoras obtenidas

4.1. Aumento de la autonomía del estudiante

Las métricas calculadas por Sonar han ofrecido no sólo a los tutores sino también a los grupos una información vital para detectar deficiencias en el código fuente y el diseño. La compilación diaria ha permitido a los grupos conocer su evolución y mejorar constantemente sus trabajos. Además Sonar permite una compilación desconectada del servidor desde el entorno Eclipse, con lo que se puede realizar un seguimiento en tiempo real de la evolución individual.

Las métricas sólo se han explicado a los grupos bajo demanda. En general, los grupos han buscado información sobre aquellas métricas sobre las que

[6] www.eclipse.org
[7] www.redmine.org

Sonar ha establecido alertas y han buscado soluciones. En este sentido Sonar ha impuesto un factor adictivo importante, puesto que los grupos han perseguido en todo momento ofrecer una imagen de corrección y han buscado optimizar al máximo todos los factores posibles. Esto ha aumentado la autonomía de los grupos a la par que ha reducido la necesidad de supervisión por parte del tutor. En algunos casos, los grupos han considerado las sesiones de seguimiento como un mero trámite, indicando al tutor que ya conocen cuáles deben ser las actuaciones siguientes y los puntos de mejora de sus trabajos.

4.2. Rúbricas para la evaluación de trabajos

Disponer de una rúbrica [2] para la evaluación de trabajos en grupo supone varias ventajas muy interesantes. Por un lado, permite a los profesores de la asignatura homogeneizar los criterios de evaluación y, por otro, permite a los estudiantes conocer cuáles son los aspectos que se consideran más importantes para su evaluación y, por lo tanto, centrarse en ellos.

No obstante, en el contexto del diseño de software, resulta complicado proporcionar un conjunto de criterios objetivos y homogéneos y en muchos casos se tiene que acudir a evaluaciones cualitativas basadas en el análisis de la cohesión y el acoplamiento. Esto es problemático para los estudiantes pues, en muchos casos, la inexperiencia en analizar distintos diseños hacen que no sean capaces de hacer una evaluación acertada de estos aspectos. Esto provoca que la rúbrica no les proporcione, en muchos casos, información realmente útil para mejorar su diseño.

Por tanto, una de las mejoras más significativas que se obtienen al disponer de un análisis de la calidad del código fuente es que proporciona un conjunto de criterios objetivos y cuantificables que resultan de gran utilidad para complementar los criterios cualitativos anteriormente mencionados a la hora de definir rúbricas para la evaluación de las prácticas.

En nuestro caso, la rúbrica abarca cinco aspectos diferentes de la práctica: diseño, verificación, documentación, funcionalidad y presentación. De esos cinco aspectos, funcionalidad y presentación son evaluados cualitativamente en base a una presentación realizada por los estudiantes en clase que incluye una demostración de la práctica desarrollada de manera que nos centraremos en los tres primeros, que aparecen reflejados en el Cuadro 1. Con respecto al diseño, los factores que hemos tenido en cuenta han sido los siguientes:

Rules compliance: Sonar permite definir un conjunto de reglas, implementadas en varias herramientas (PMD, FindBugs, Squid, CheckStyle), que se comprueban sobre el código fuente en busca de posibles errores, expresiones demasiado complejas, convenciones de estilo de código y malas prácticas como métodos muy grandes o clases con muchos métodos. Existen distintos catálogos predefinidos de reglas y, además, Sonar permite definir catálogos propios en base a los predefinidos o añadiendo nuevas reglas. En nuestro caso, hemos creado nuestro propio catálogo basándonos fundamentalmente en el llamado *Sonar Way with Findbugs*. La métrica elegida refleja la proporción de conformidad de las reglas frente al número de líneas de código de la aplicación.

Package tangle index: Índice que refleja la cantidad de dependencias cíclicas existentes entre los paquetes de la aplicación. El valor ideal es $0\,\%$ que indica que no existen dependencias cíclicas entre los paquetes. La razón de incorporar esta métrica es para evaluar la manera en que se han repartido las clases entre los distintos paquetes de la aplicación.

Useless code: Número de líneas no útiles del código. Se calcula como la suma de líneas de código duplicadas, líneas de código en métodos privados no utilizados y líneas de código en métodos protegidos no utilizados. La razón de incorporar esta métrica es para combatir la tendencia de los estudiantes a copiar y pegar líneas de código y a dejar código no utilizado conforme van desarrollando la práctica.

LCOM4 density: LCOM4 es una métrica de cohesión de las clases basada en calcular el número de "componentes conexas" de una clase. Una componente conexa es un conjunto de métodos y atributos relacionados de una clase. En cada clase debería haber una única componente conexa. Si hay dos o más, la clase debería dividirse en otras clases más pequeñas. LCOM4 refleja el porcentaje de clases que tienen un $LCOM4 > 1$. Esta métrica permite evaluar la cohesión de las clases desarrolladas.

Complexity by method: Media de la complejidad ciclomática por método. La razón de incorporar esta métrica es para evaluar la forma en la que se ha dividido el código en métodos combatiendo la tendencia del estudiante de crear métodos con varios bucles o condicionales anidados.

En cuanto a las pruebas, los factores que se han

	Calificación		
Métrica	A	B	C
Diseño			
Rules compliance	> 95	$80 - 95$	< 80
Package tangle	< 0	$0 - 40$	> 40
Useless code	0		> 0
LCOM 4 density	$< 15\,\%$	$15 - 30$	$\geq 30\,\%$
Compl./method	< 3		≥ 3
Verificación			
Line coverage	>40	20-40	<20
Unit test success	$100\,\%$		$<100\,\%$
Documentación			
Comentarios	$>25\,\%$	10-25	<10

Cuadro 1: Rúbrica para la evaluación cuantitativa referida a diseño, verificación y documentación

tenido en cuenta en la rúbrica han sido:

Line coverage: Es la proporción de líneas de código ejecutadas por las pruebas unitarias frente al total de líneas de código ejecutables de la aplicación. La razón de incorporar esta métrica es para evaluar la cantidad de código fuente que está siendo ejecutado por las pruebas unitarias.

Unit test success: Refleja el porcentaje de pruebas unitarias que dan resultado satisfactorio. La razón de incorporar esta métrica es para hacer énfasis en la necesidad de que el código debe pasar todas las pruebas unitarias definidas.

Por último, para la documentación se ha tenido en cuenta el porcentaje de líneas de comentarios en el código frente a la totalidad del código escrito.

Además, la rúbrica utilizada incluye también otros elementos cualitativos no mostrados en la Tabla 1. En concreto, para el diseño se realiza una evaluación de cohesión y acoplamiento del diseño completo, se analiza la asignación de responsabilidades realizada, se valora la legibilidad del código y se penaliza la existencia en el código de malos olores [1]. Para las pruebas se valora su legibilidad, el uso de *mocks* y las pruebas realizadas sobre métodos complejos. Por último, en cuanto a la documentación, se valora la legibilidad y el uso correcto de los diagramas UML, la documentación de las pruebas realizadas y la descripción realizada a los problemas de diseño encontrados por medio de memorandos técnicos. Todos estos aspectos cualitativos de la rúbrica son valorados en una escala de la A a la F.

Con respecto a la calificación, la nota final se obtiene como la media aritmética entre los aspectos cuantitativos y los aspectos cualitativos de la rúbrica. De este modo, el 50 % de la nota de la práctica viene determinada por aspectos cuantitativos a los que el alumno tiene acceso.

4.3. Seguimiento de trabajos

La mejora en la atención y el aprovechamiento de las sesiones de seguimiento semanal ha sido uno de los objetivos de nuestra actuación. El tutor debe ser capaz de revisar en 20 minutos el diseño realizado, el código fuente producido, la memoria y la estructura organizacional del grupo de trabajo. Sonar recoge la evolución temporal de las métricas con lo que se tiene numerosa información sobre la progresión del trabajo y la mejora en el uso de técnicas de diseño. Esto ha permitido una puesta en contexto rápida por parte del tutor y ha ofrecido la posibilidad de realizar un análisis previo a las sesiones de seguimiento sobre el diseño y el código fuente.

Además se han homogeneizado las memorias haciendo uso de plantillas LaTeX, lo que ha facilitado la generación de documentación por parte de los estudiantes y la revisión por parte del tutor. Se ha propuesto una estructura incremental en la que cada entrega se corresponde con un capítulo lo que se facilita la detección de los cambios realizados sobre el diseño.

La experiencia obtenida por el profesorado durante las sesiones de seguimiento ha servido para validar las rúbricas definidas y aumentar la confianza en el uso de criterios objetivos y su impacto en las calificaciones final.

5. Resultados obtenidos

Una vez finalizada la asignatura podemos afirmar que la plataforma presentada ha ayudado a lograr los objetivos definidos de la siguiente forma:

1. Sonar proporciona información útil al grupo de trabajo para detectar puntos débiles y definir acciones de mejora del diseño. Esto dota al grupo de una mayor autonomía lo que ha conseguido reducir la demanda de tutorías drásticamente.
2. Las rúbricas que se apoyan en métricas obtenidas automáticamente a partir de Sonar han permitido objetivar y homogeneizar los criterios de evaluación.
3. La información ofrecida por Sonar permite al tutor contextualizarse rápidamente en las sesiones de seguimiento y analizar la evolución diaria de los trabajos. Además permite al tutor realizar un estudio previo que ayuda a mejorar la organización de las sesiones de seguimiento.
4. La complejidad y el tamaño de los diseños son dos métricas básicas ofrecidas por Sonar. En base a ellas se puede analizar y comparar la dificultad entre los trabajos, pudiendo ajustarlas el esfuerzo que el estudiante debe realizar.

A fin de obtener datos cuantitativos de la mejora producida con nuestra actuación, hemos realizado un análisis de los trabajos del curso pasado mediante Sonar, aplicando la rúbrica de evaluación de este curso. Atendiendo únicamente a criterios objetivos, los trabajos del curso 2011 han obtenido un 6,3 de media frente al 3,9 de los trabajos del curso 2010. Atendiendo sólo a métricas de diseño la calificación media es de 7,3 frente al 4,8. Esto señala una mejora significativa en la calidad de los diseños. Si bien la calidad de los diseños es similar en lo que respecta a la cohesión de las clases y su acoplamiento, se aprecia una desviación importante en métricas a nivel de línea de código como son el número de líneas duplicadas y el cumplimiento de reglas avanzadas. Esto se debe a que el tutor no puede detectar este tipo de problemas durante una supervisión en un tiempo razonable.

Podemos afirmar por tanto que la calidad global de los proyectos ha mejorado a la vez que se ha logrado reducir drásticamente el tiempo invertido por parte del tutor en la supervisión de los proyectos, dotando a los estudiantes de una importante capacidad de autogestión de la calidad. Además, el estudiante conoce y aprecia los beneficios que ofrecen las ALM, considerándolos una herramienta fundamental para el desarrollo de futuros proyectos y que les permitirá mejorar sus habilidades como diseñadores de software.

Por contra, la instalación, configuración y mantenimiento del ALM necesita de personal especialista. El profesor no debe asumir esta carga. En este proyecto se ha obtenido financiación para un becario que asistiera en las tareas de mantenimiento de la plataforma y se ha contado con el soporte gratuito por parte de Klicap[8], suministrador de parte de la plataforma. En futuros proyectos consideraremos el uso de plataformas ALM en la nube que son una alternativa interesante tanto desde el punto de vista económico como de disponibilidad.

Hasta donde sabemos, este trabajo es novedoso en su aplicación a la enseñanza de la ingeniería del software. Existen no obstante otras propuestas que han hecho uso de métricas para evaluar código fuente [4, 8] pero no así para su seguimiento. No obstante se trata de una propuesta *ad-hoc* no basado en herramientas existentes.

5.1. Análisis de resultados académicos

En años anteriores la evaluación combinaba la realización de un trabajo práctico con dos pruebas de conocimiento teórico y práctico respectivamente. La ponderación del trabajo era del 50 % frente a un 20 % y 30 % de las pruebas de conocimiento teórico y práctico respectivamente. Hemos entendido que separar el aprendizaje entre teoría y práctica no ayuda a lograr los objetivos de la asignatura y provoca un cambio de contexto innecesario en el estudiante. La práctica implica conocer y aplicar correctamente los principios teóricos con lo que con ambas pruebas de conocimiento se produce una redundancia en la valoración de conceptos. Además, la prueba teórica consistía en un test cuyas calificaciones eran bastante discretas debido al cambio de contexto y a la dificultad de diseñar una prueba equilibrada. Por esta razón se ha optado por unificar estas pruebas e incorporar una evaluación de porfolios que permita al estudiante profundizar en algunos aspectos del diseño, adaptando la docencia a sus preferencias.

Adicionalmente, la calificación del trabajo iba en

[8] www.klicap.es

	2010			2011
Convocatoria	1C	2C	3C	1C
M.Honor	2	0	0	2
Sobresalientes	1	0	0	3
Notables	16	6	1	53
Aprobados	45	8	10	22
Suspendidos	36	17	4	21
No Presentados	22	23	29	29
Total	122	54	44	130

Cuadro 2: Resultados académicos de 2010 y 2011

consonancia a la dificultad, lo cual era inadecuado. Se puede realizar un diseño excelente en un trabajo de poca envergadura y obtener una calificación inferior a un grupo que realice un diseño pobre en un problema de grandes dimensiones. Entendimos que este sistema no valora el aprendizaje de habilidades de diseño, por lo que la complejidad afecta actualmente a la ponderación en lugar de a la complejidad. Esto ha permitido mejores calificaciones y más cercanas al rendimiento y aprendizaje real del alumnado. El Cuadro 2 muestra una comparativa de resultados académicos en las últimas convocatorias.

6. Futuro y Aplicabilidad de Resultados

La asignatura en la que se ha desarrollado esta experiencia se extingue con este curso. No obstante, pretendemos aplicar lo aprendido en otras asignaturas de los nuevos planes de estudio donde se desarrolle software. A tenor de la experiencia y de las opiniones vertidas por el alumnado, entendemos que plataformas como la presentada en este articulo deben formar parte del día a día del ingeniero/a en informática. Trataremos de incorporar nuestra propuesta a la plataforma ProjETSII.

En cuanto a los aspectos a mejorar, la objetividad en las rúbricas puede incrementarse aún más en base a la experiencia. Si bien la mejora ha sido significativa, creemos que con un análisis exhaustivo de los cerca de 40 trabajos realizados hasta la fecha podemos definir una rúbrica mucho más objetiva.

Por último, creemos que el uso de métricas y el factor adictivo que produce en el alumno la mejora constante de las mismas, abre la puerta a explorar la gamificación del proceso de aprendizaje del diseño, introduciendo competiciones entre grupos y un sistema automático de recompensas y castigos.

Agradecimientos

Queremos agradecer su contribución a Manuel Recena de Klicap, Antonio M. Gutiérrez y Juan Jesús Pérez Luna. Este trabajo ha sido financiado por el proyecto de innovación docente n^o 249 del Plan Propio de Docencia de la Universidad de Sevilla.

Referencias

[1] *Refactoring: improving the design of existing code*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[2] I. Gallego, J. López, E. Rodríguez, E. Salamí, E. Santamaría, and M. Valero. Presentaciones orales a un coste razonable. In *Actas de JENUI 2010*, pages 25–32, 2010.

[3] M. Jiménez, P. Fernández, and R. García. Opera: Una herramienta soporte para el aprendizaje basado en proyectos. In *Actas de JENUI 2011*, Sevilla, 2011.

[4] S. A. Mengel and V. Yerramilli. A case study of the static analysis of the quality of novice student programs. *SIGCSE Bull.*, 31:78–82, March 1999.

[5] T. F. S. nes, D. M. Consarnau, M. Marqués, and E. R. Nieto. El contrato de aprendizaje en la enseñanza universitaria. In 4^o *Congreso Internacional Docencia Universitaria e Innovación*, pages 1–16, Barcelona, 2006.

[6] J. Pérez-Jiménez, A. Durán, and B. Bernárdez. Fundamentos para un entorno de application lifecycle management dirigido por procesos. In *Actas del II Taller de Procesos de Negocio e Ingeniería de Servicios*, pages 41–48, San Sebastián, España, 2009.

[7] P. Trinidad, M. Resinas, C. Müller, J. Parejo, and A. Ruiz-Cortés. Aprendiendo a diseñar software usando juegos de mesa como enunciado de prácticas. In *Actas de JENUI 2011*, Sevilla, 2011.

[8] N. Truong, P. Roe, and P. Bancroft. Static analysis of students' java programs. In *Proceedings of the Sixth Australasian Conference on Computing Education - Volume 30*, ACE '04, pages 317–325, Darlinghurst, Australia, Australia, 2004. Australian Computer Society, Inc.