http://dx.doi.org/10.14201/eks20151648196
e-ISSN 2444-8729

Uso de GitHub en el diseño de e-actividades para la refactorización del software

Using GitHub in the design of e-activities for software refactoring

Carlos López, Jesús M. Alonso, Raúl Marticorena, Jesús M. Maudes

Área de Lenguajes y Sistemas Informáticos, Universidad de Burgos, Burgos, España.
{clopezno, rmartico, jmaudes} @ ubu.es, jesus.alonso.abad@gmail.com

Resumen

El objetivo de este trabajo es diseñar, planificar, aplicar y evaluar actividades docentes que ayuden en el proceso de enseñanza y aprendizaje del concepto de refactorización. La metodología didáctica seguida se basa en dos pilares. El primero es en un aprendizaje progresivo del concepto de refactorización mediante e-actividades definidas en diferentes niveles de conocimiento de la taxonomía de Bloom (conocer, comprender, aplicar, analizar, sintetizar). El segundo es la utilización de recursos en las e-actividades, que estén relacionados con los que el estudiante podrá encontrarse al ejercer su carrera profesional orientada al desarrollo del software: entornos de desarrollo integrados y repositorios de gestión de proyectos. El resultado del trabajo es la definición de un conjunto de e-actividades de refactorización de código Java, donde se utiliza de diferentes formas según el tipo de e-actividad, la funcionalidad del versionado del repositorio de proyectos de código abierto GitHub. Bajo estas premisas se diseñan cinco tipos de e-actividades: lecturas y comprensión, pruebas objetivas, aprendizaje basado en problemas, estudio de casos y seminarios virtuales. Las e-actividades diseñadas se aplican en el contexto de una asignatura de Ingeniería del Software del Grado en Ingeniería Informática de la Universidad de Burgos. El trabajo concluye analizando preliminarmente algunas consecuencias de la experiencia, tanto desde la perspectiva de utilizar estas nuevas e-actividades en el proceso de enseñanza aprendizaje, como desde la perspectiva de carga de trabajo que supone al docente y al estudiante.

Palabras Clave:

Refactorización; proceso enseñanza aprendizaje online; e-actividades; evaluación; taxonomía de Bloom; Github repositorio de proyectos software.

Abstract

The aim of this work is to design, plan, apply and assessment educational activities to help in the teaching-learning process of the concept of refactoring. The teaching methodology used is based on two pillars. The first is a progressive learning of the concept of refactoring by e-activities defined at different levels of knowledge of Bloom's taxonomy (knowledge, comprehension, application, analysis, synthesis, evaluation). The second is the use of resources in e-activities that are related to those the student may find during their professional career oriented towards software development: integrated development environments and software project repositories. The result of this work is the definition of a set of Java code refactoring e-activities. The version control functionality of software project repository, i.e.; GitHub, is used in different ways depending on the type of e-activity. Under these assumptions five types of e-activities are designed: Reading and comprehensions, objective tests, problem-based learning, case studies and webinars. The e-activities designed are applied in the context of the course Software Engineering of the Bachelor's Degree in Computer Science at the University of Burgos. The paper concludes analyzing some consequences of the experience from the perspective of using these new e-activities in the teaching-learning process as from the perspective of workload involved for both the teacher and the student.

Keywords:

Refactoring; online teaching-learning process; e-activities; Bloom's taxonomy; GitHub open source software repository.

Recepción: 03-03-2015 Revisión: 04-06-2015 Aceptación: 24-10-2015 Publicación: 01-12-2015

1. Introduction

En el área de conocimiento de la ingeniería del software (Bourque & Fairley 2014) las tareas de refactorización de código se encuentran dentro del sub área de mantenimiento del software. La refactorización es un tipo de tarea de corrección que se define como: una transformación del programa aplicada en fase de mantenimiento que no cambia su comportamiento externo (Fowler *et al.*, 1999). Los motivos para refactorizar son la mejora de algún atributo de calidad. En el contexto de refactorización también se describe y categoriza el concepto de defecto de código (*bad smell*). Es decir, el proceso de refactorización se inicia para eliminar una instancia concreta de un defecto de código y mejorar así algún atributo de calidad.

El resultado de una tarea de refactorización es un cambio en el código. Desde un punto de vista de ingeniería del software, la gestión de cambio está definida en el área de conocimiento de gestión y configuración del software. Parece interesante poder definir actividades de aprendizaje que tengan en cuenta esta relación.

Actualmente las tareas de refactorización son utilizadas en los equipos de desarrollo empresarial, y gozan de una gran popularidad entre los desarrolladores, especialmente en los principales países exportadores de desarrollos software (ver Fig. 1). Además tienen una multitud de documentación, tanto *online* y libre (catálogo de M. Fowler en refactoring.com), como bibliográfica (Brown, 1998), (Fowler *et al.*, 1999), (Lippert & Roock, 2006), (Wake, 2004).

Fig. 1. Tendencias término de búsqueda "refactor", categoría "informática electrónica".

El problema planteado es que ninguna de la bibliografía mencionada tiene un enfoque puramente pedagógico, donde se pueda guiar el proceso de enseñanza y aprendizaje utilizando en menor medida las metodologías centradas en el profesor (caracterizadas como expositivas y pasivas), para ir evolucionando hacia metodologías y actividades centradas en el estudiante (activas, dinámicas y participativas) (Meneses, Fernández & Ballesteros-Regaña, 2011). Según (Cabero & Román, 2005), las actividades para el aprendizaje se refieren a las diferentes acciones que los estudiantes llevan a cabo en completa relación con los contenidos e informaciones que les han sido ofrecidos. Si estas actividades son presentadas, realizadas o transferidas a través de la red, entonces se pueden considerar como e-actividades.

Los repositorios de proyectos, como GitHub, pueden ayudar a desarrollar e-actividades de refactorización y favorecer la interacción de

los estudiantes. La utilización de repositorios de control de versiones en docencia busca la mejora del proceso de enseñanza/ aprendizaje. En (Britton & Berglund, 2013) y en (Kelleher, 2014) se usan para mejorar el proceso de asignación, de evaluación y de retroalimentación en la entrega de tareas. Además, en (Kelleher, 2014) se usa como estrategia expositiva para diseminar ejercicios y presentar sus soluciones. En otros trabajos se analizan la estrategia del uso de estos repositorios, bien como habilidad especializada y encerrada en una asignatura, o como habilidad transversal aplicada en múltiples asignaturas. En (Lawrance, Jung & Wiseman, 2013) se propone empezar a utilizar los servicios de repositorios de proyectos en la nube, como Git, desde los primeros cursos del currículo CS (Computer Science), incluso en estudiantes de otras ramas de la ingeniería distintas a CS.

La taxonomía de Bloom ordena de manera jerárquica los diferentes niveles de aprendizaje de un nuevo concepto (ver Fig. 2). Esta taxonomía se puede utilizar para determinar el grado de maestría obtenido en los distintos módulos de un plan de estudios. (Bourque, Buglione, Abran & April, 2003) aplican la taxonomía sobre los temas del cuerpo de conocimiento de ingeniería del software (SWEBOK), su objetivo es establecer tres perfiles de los ingenieros software: nuevo graduado, graduado con cuatro años de experiencia y miembro experimentado de un equipo de procesos de ingeniería.

En concreto, en este trabajo se definirán un conjunto de e-actividades que persiguen cubrir los niveles de la taxonomía de Bloom: conocimiento, comprensión, aplicación, análisis, síntesis y evaluación. Cada

Fig. 2. Orden de pensamiento de la taxonomía de Bloom.

e-actividad utilizará plantillas pedagógicas para su descripción: tipo de actividad, roles del docente y el alumno, recursos TIC necesarios, resultados o salida esperada y criterios de evaluación. Además se usa GitHub para mejorar la distribución, la evaluación y la retroalimentación de estas e-actividades.

Este trabajo es una continuación del artículo presentado en XVI Simposio Internacional de Informática Educativa SIIE (López, Alonso, Marticorena & Maudes, 2014). Al trabajo original se le añaden algunos comentarios adicionales para ampliar la información de la definición de e-actividades de refactorización con GitHub como recurso TIC.

El resto de artículo se estructura de la siguiente forma. En la Sec. 2, se enumeran los tipos de e-actividades y se planifican en el contexto de la asignatura objetivo. En la Sec. 3, se detalla el diseño pormenorizado de las e-actividades para, en la Sec. 4, describir su aplicación y su relación con GitHub. Finalmente, en la Sec. 5 se presentan las

conclusiones obtenidas del presente trabajo, así como las líneas de trabajo abiertas.

2. Planificación de e-actividades

Antes de abordar su planificación, es necesario tener claro el conjunto de e-actividades a utilizar en la asignatura. A continuación se realiza la propuesta de las mismas, para posteriormente encuadrarlas en el proceso de enseñanza/aprendizaje.

2.1. Tipo de e-actividades utilizadas

El tipo de e-actividades que se proponen son:

- Lectura. Las lecturas son los textos y la documentación escrita, tanto de obras completas como fragmentos que se han recogido y editado como fuente de profundización en los contenidos trabajados.
- Ejemplos. Se trata de la exposición de ciertos elementos que, por analogía, pueden transferirse a otros contenidos o situaciones. Los ejemplos facilitan la comprensión de las informaciones y de los contenidos de aprendizaje, conocimientos, ideas o procedimientos. El abanico de objetivos que permiten trabajar es muy amplio, pero normalmente por sí solos los ejemplos no aseguran la consecución de ninguno de ellos. En este sentido, se caracterizan porque actúan como complementos de otros recursos.
- Pruebas objetivas. Son instrumentos de medida, elaborados rigurosamente, que permiten evaluar conocimientos, capacidades, destrezas, rendimiento, aptitudes, actitudes, inteligencia, etc. Suelen ser un recurso utilizado para la evaluación diagnóstica, formativa y sumativa. Podemos encontrarlas de naturaleza variada como por ejemplo: de respuesta breve, de completar, de discriminación, de ordenación, de localización o identificación.
- Aprendizaje basado en problemas. Es una técnica en la cual el estudiante debe resolver una situación problemática concreta (que puede tener más de una posible solución), a partir de los contenidos que se han trabajado.
- Estudio de caso (*Case-Based Learning*). El estudio de caso se basa en la presentación de una situación problemática, real o ficticia, que el estudiante tiene que resolver. Las competencias sobre las que se trabaja son: identificación y análisis de factores externos e internos, planificación de procesos y toma de decisiones argumentada.
- Seminario virtual. Es una técnica de grupo que promueve el estudio intensivo de un tema. Se caracteriza por la

discusión, la participación, la elaboración de documentos y las conclusiones compartidas por todos los componentes del seminario. Su objetivo es explorar sobre un tema concreto, reflexionar sobre un tema específico, transmitir informaciones.

2.2. Planificación del proceso de enseñanza y aprendizaje

En el diseño del proceso de enseñanza y aprendizaje mediante e-actividades es interesante diversificar para trabajar diferentes habilidades y niveles de conocimiento. Previamente es recomendable haber cursado una asignatura de diseño del software donde se haya tratado conceptos de patrones arquitectónicos y de diseño que permitan dirigir el proceso de refactorización. Bajo esta premisa, en la Tabla I se muestra nuestra propuesta de planificación para formar a alumnos del Grado de Ingeniería Informática de la mención de Ingeniería del Software de 8º semestre, en las tareas de mantenimiento del software relacionadas con refactorización de código.

Bloom [a]	**Planificación enseñanza/aprendizaje de refactoring**		
	Descripción	Tipo	Duración
C, Com	Lectura y comprensión de bibliografía sobre defectos de código	Lectura, Ejemplos, Pruebas objetivas	2 h
Com, Ap	Identificación de defectos de código	Aprendizaje basado en problemas	2 h
Ap, An	Detección de defectos de código en sistemas open-source	Estudio de casos	4 h
C, Com	Lectura y comprensión de bibliografía sobre catálogo de refactorizaciones	Lectura, Ejemplos, Pruebas objetivas	4 h
Com, Ap	Aplicación refactorizaciones aisladas en IDE	Aprendizaje basado en problemas	2 h
Ap, An	Proceso de refactorización mediante una secuencia de refactorizaciones	Estudio de casos	6 h
Com, Ap	Ejercicios de secuencia de refactorizaciones relacionadas	Aprendizaje basado en problemas	4 h
Sin	Visita virtual a una empresa de desarrollo para relacionar las revisiones de código con defectos y refactorizaciones	Seminario Virtual	2 h

[a] *C=Conocimiento, Com=Compresión, Ap=Aplicación, An=Análisis y Sin=Síntesis*

Tabla1. E-Actividades de refactorización

3. Diseño de e-actividades de refactorización

Cada descripción de e-actividad debería contener la siguiente información (Meneses, Fernández & Ballesteros-Regaña, 2011):

- Recursos asociados bien de elaboración propia, o en abierto, o de la biblioteca, u otros.
- Cómo se puede presentar en el entorno virtual, con qué tecnología o aplicación y con qué formato.
- Rol del docente y del alumno, o grupo de alumnos, si se plantean dinámicas colaborativas.
- Resultado u *output* esperado, cómo se evaluará y con qué criterios.

3.1 Compresión de conceptos: lecturas, ejemplos y pruebas objetivas

Como recursos para este nivel de conocimiento, además de la bibliografía (Fowler *et al.*, 1999) (Wake, 2004), catálogo de refactorizaciones y defectos de código disponibles en online (http://refactoring.com , http://sourcemaking.com/refactoring), se proporcionan videotutoriales de uso de herramientas propios de los autores que permiten contextualizar el entorno tecnológico para refactorizar: lenguaje de programación, herramientas de detección de defectos y refactorización.

Todas las actividades de comprensión de conceptos llevan asociado un cuestionario formativo con posibilidad de dos intentos. Las preguntas que se incluyen son de tipo relación de conceptos (*match*) y de selección múltiple (*quiz*), con valoración negativa en las respuestas incorrectas y distintos pesos en función de las respuestas. Las calificaciones positivas son proporcionales al número de respuestas correctas, y las negativas proporcionales al número de respuestas incorrectas. En el texto de las preguntas se incluyen referencias de lectura a los recursos y a ejemplos. Se propone utilizar los cuestionarios de Moodle.

Esta tarea es individual. Sirve de autoevaluación al alumno, y al profesor le permite realizar un seguimiento del aprendizaje del alumno.

3.2 Aprendizaje basado en problemas

Para poder realizar esta actividad es necesario haber superado las pruebas del nivel de comprensión. Los recursos utilizados son esqueletos de código compilados que recogen de manera sintetizada entre uno y tres defectos de códigos. Los códigos son de

pequeño tamaño, máximo 200 LOC (*Lines Of Code*). El recurso de código se suele presentar al alumno a través de una referencia a un repositorio de ejercicios de código públicos, cuyo autor es algún docente de la materia.

Respecto a la dinámica, el docente es el encargado de seleccionar los problemas de acuerdo con el resto de e-actividades. Especialmente relevante es elegir códigos con defectos y refactorizaciones que se vayan a utilizar en las e-actividades de estudio de casos. La actividad para los alumnos sigue siendo individual.

Las salidas de la actividad son dos. Por un lado, la identificación del tipo de defecto de diseño indicando su localización en el código. Por otro lado, el nuevo código resultado de aplicar la refactorización para eliminar el defecto. La evaluación está en función de la calidad de las respuestas y uso correcto de los conocimientos adquiridos.

Se plantean tres alternativas para presentar esta actividad en el entorno virtual. La primera es realizar un cuestionario cuyas preguntas sean de texto abierto. La segunda es definir una pregunta de texto incrustado (*cloze*), donde el texto de la pregunta incluye una descripción del problema de refactorización sobre un código y el código resultante después de aplicar la refactorización. Sobre este se seleccionan identificadores y palabras clave en partes de código afectadas por la refactorización. La tercera es un tipo de foros especial conocido como foros de preguntas respuesta. La dinámica en estos foros es la siguiente: el profesor plantea cada problema con un enunciado en un hilo de conversación y los alumnos contestan. En este tipo especial de foros los alumnos no ven las respuestas de sus compañeros si no han contestado. En ambas alternativas la escala de calificación de las respuestas será de tres niveles (Mal, Regular, Correcto).

3.3 Estudio de casos

Para poder realizar esta actividad es necesario haber superado las pruebas del nivel de comprensión y es aconsejable un nivel básico de aplicación.

En esta actividad se pretende aproximar al alumno a un contexto más real de desarrollo software. Se utilizan como recursos software entornos de desarrollo integrados, con funcionalidades que automaticen operaciones de refactorización y de detección de defectos de código presentados en las e-actividades de conocimiento y comprensión.

Los códigos de programas utilizados pertenecen a proyectos software reales o de carácter formativo. Los proyectos reales son seleccionados de algún repositorio de proyectos de código abierto (e.g.; *SourceForge, GitHub, GoogleCode, Bitbucket*). Los proyectos de formación suelen seleccionarse de fuentes bibliográficas concretas: Video club de

(Fowler *et al.*, 1999), *Refactoring Lab* de (Demeyer *et al.*, 2007) (Nierstrasz, Ducasse & Demeyer, 2009).

Esta actividad se realiza en grupo de dos participantes. El profesor proporciona una dirección web de un repositorio de proyectos de código abierto. También añade un enunciado donde se propone una nueva tarea de desarrollo, bien correctiva o adaptativa. Los alumnos, para realizar sus tareas, generan su propia versión, en el repositorio central, a partir de la proporcionada por el profesor (*fork*). Por cada operación de refactorización se realizan entregas en su rama del repositorio (*commit*), indicando en el texto descriptivo la refactorización realizada. El caso de estudio suele durar como mínimo un par de jornadas de dos horas.

El producto entregable está compuesto de dos componentes. En primer lugar, una dirección web del repositorio de código. Con ella el profesor puede ver la historia de creación de la nueva versión. Además obtiene el producto final, es decir, la nueva versión del código fuente. El segundo componente es una reflexión sobre la relación de los conceptos de refactorización con el desarrollo del software, y el uso de esta funcionalidad mediante herramientas de desarrollo. La reflexión se puede obtener con foros de tipo pregunta respuesta o con un documento/memoria con un tamaño máximo de dos folios.

La evaluación considera tres aspectos. El primero es el proceso seguido para obtener la solución, observando las entregas junto con sus textos descriptivos (*commits*) en el sistema de control de versiones centralizado. El segundo es la calidad de la solución basada en el conjunto de refactorizaciones, o tipo de defectos aplicados. El tercero es la calidad de las reflexiones a las preguntas de texto abierto basadas en la experiencia adquirida en el caso de estudio concreto.

3.4. Seminario virtual

Como colofón del aprendizaje se propone un seminario con una empresa de desarrollo del software que utilice refactorización y detección de defectos de código en su flujo de trabajo diario.

El profesor contacta con la empresa para fijar la temática del seminario, y negocia la duración dedicada al tema de refactorización, fechas y horas, ponentes que trabajen en tareas de desarrollo de código y los sistemas de video conferencia (*webmeeting* o *webminar*) a utilizar en la presentación.

En la dinámica de trabajo el profesor actúa de moderador. Toma el control los cinco primeros minutos para dar una introducción de la empresa junto con sus productos software desarrollados. Posteriormente presenta brevemente a los ponentes junto con sus funciones dentro de la empresa. Además, comenta brevemente a los ponentes los conocimientos adquiridos y e-actividades realizadas por los alumnos durante su

formación en refactorización. Los ponentes de la empresa toman el control de la exposición pudiendo ser interrumpidos con preguntas de los estudiantes durante la exposición.

La dinámica para el alumno en esta actividad se basa únicamente en la asistencia, sin considerar la participación en la evaluación.

4. Aplicación en un contexto docente

Una aplicación del diseño de las e-actividades se ha implementado en una asignatura optativa de la mención de Ingeniería del Software de 8º semestre del Grado de Ingeniería Informática de la Universidad de Burgos. Este ha sido el primer año que se imparte. En la asignatura estaban matriculados 12 alumnos y la imparte un solo profesor. Actualmente la asignatura del Grado en Informática solo se imparte de manera presencial, pero dispone institucionalmente de una plataforma de aprendizaje para definir e-actividades basada en Moodle llamada UBUVirtual.

En esta asignatura la carga docente dedicada a enseñar los conceptos de refactorización es aproximadamente el 50 %. Como prerrequisitos de conocimiento previos para cursar la asignatura se necesitan conocimientos medios de programación, de entornos de desarrollo integrados, y de otras actividades de desarrollo del software. Analizando la base curricular de los alumnos en la Universidad de Burgos se eligió el siguiente entorno tecnológico:

- Lenguaje de programación Java.
- Entorno de desarrollo integrado Eclipse (distribución Eclipse IDE f*or Java Developers*). Además se incorporan tres extensiones adicionales. La primera para analizar cobertura de pruebas (*EclEmma*), la segunda para ayudar a localizar defectos de código (*InCode*) y la tercera para obtener medidas estáticas de código (*RefactorIt*).
- Repositorio de proyectos de código abierto (*GitHub*).

Dada la complejidad tecnológica de la asignatura, el profesor ha creado, como recurso docente, una lista de reproducción en un canal de *Youtube* que contiene cuatro videos para facilitar el uso concreto de las herramientas (disponible en http://goo.gl/abH1Oz). Los video tutoriales de herramientas son enlazados como recursos en UBUVirtual, próximos en tiempo y con formato apropiado para la e-actividad que se esté realizando.

Como evidencia de interacción con las e-actividades implementadas en el entorno virtual de aprendizaje, en la Fig. 3 se muestra

Fig. 3.. Analítica de uso de UBUVirtual.

la historia de uso de UBUVirtual durante el periodo de tiempo en el que se enseñaron los conceptos de refactorización. A modo de resumen, se ha utilizado el informe estadístico que proporciona Moodle por defecto. Los valles de la gráfica se corresponden con periodos vacacionales.

Las siguientes secciones describen con más detalle algunas de las e-actividades definidas.

4.1 Compresión de conceptos: lecturas, ejemplos y pruebas objetivas

La lectura de códigos de ejemplos y su evolución después de realizar alguna transformación es uno de los principales recursos didácticos para el proceso de enseñanza y aprendizaje del concepto de refactorización. En GitHub existe una funcionalidad especial llamada Gist. El objetivo de un Gist es compartir de manera simple fragmentos de código con otros pudiendo ser estos versionados y comentados. Estas dos funcionalidades (i.e. versionado y capacidad de añadir un diálogo mediante comentarios), permiten presentar los conceptos de manera dinámica a través de hiperenlaces donde se puede destacar la evolución del código y su justificación teórica. En el ejemplo disponible en el Gist https://gist.github.com/clopezno/2d5028391f1eea4f0c84 se muestra un Gist donde se aplican tres patrones de diseño (estrategia, adaptador y observador) sobre un código de ejemplo (Kerievsky, 2004).

El Gist referenciado tiene tres versiones de código y un diálogo con tres comentarios correspondientes a las justificaciones de aplicar cada patrón de diseño propuesto. En la versión inicial del Gist se plantea un comentario con una descripción de una clase Java (*Paragraph*) para aplicar el patrón de diseño *estrategia*, y se enlaza con la solución de la primera revisión de código. El segundo comentario describe una evolución del código con un nuevo requisito donde se aplica el patrón *adaptador*. Igual que en el caso anterior, el comentario tiene una referencia a la solución de la nueva versión de código. El proceso pedagógico es el mismo cuando se aplica el patrón de diseño *observador* en el tercer comentario.

La principal ventaja didáctica de aplicar esta solución es que permite gestionar de manera directa los cambios de código producidos resaltando mediante un lenguaje de colores las zonas cambiadas (rojo código eliminado y verde código añadido). En la Fig. 4 se muestra cómo cambia el código de la clase *Paragraph* cuando se aplica el patrón de diseño observador.

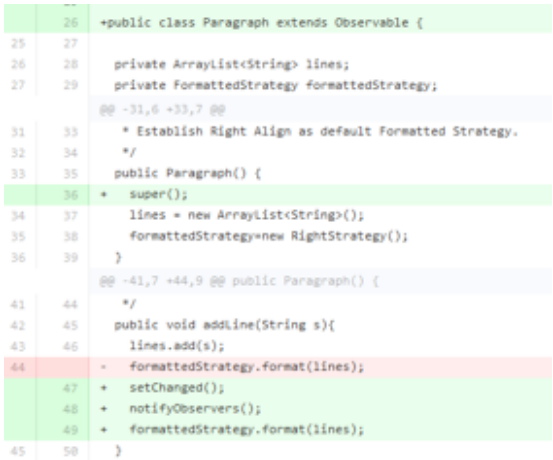

Fig. 4. Evolución de cambios de código mediante Gits.

4.2. Aprendizaje basado en problemas de refactoring

Una de las estrategias formativas para aplicar el conocimiento de refactorización en las etapas iniciales es fomentar la interacción controlada del estudiante. Con este objetivo, se han seleccionado las preguntas incrustadas. En la Fig. 5 se muestra una la visualización, desde la visión del estudiante, de una pregunta incrustada (*cloze*), para enseñar cómo funciona la refactorización *Move Method.* Tanto el código origen, como la descripción de la refactorización, se distribuyen con un enlace a Gist (ver https://gist.github.com/clopezno/10018544). Además del código, en el enunciado se proporciona como recurso el enlace a la descripción de la refactorización *Move Method* en el catálogo de Fowler (http://refactoring.com/catalog/moveMethod.html). La elección de huecos o palabras clave en el código se corresponde con cambios que sufre el código original después de realizar la refactorización.

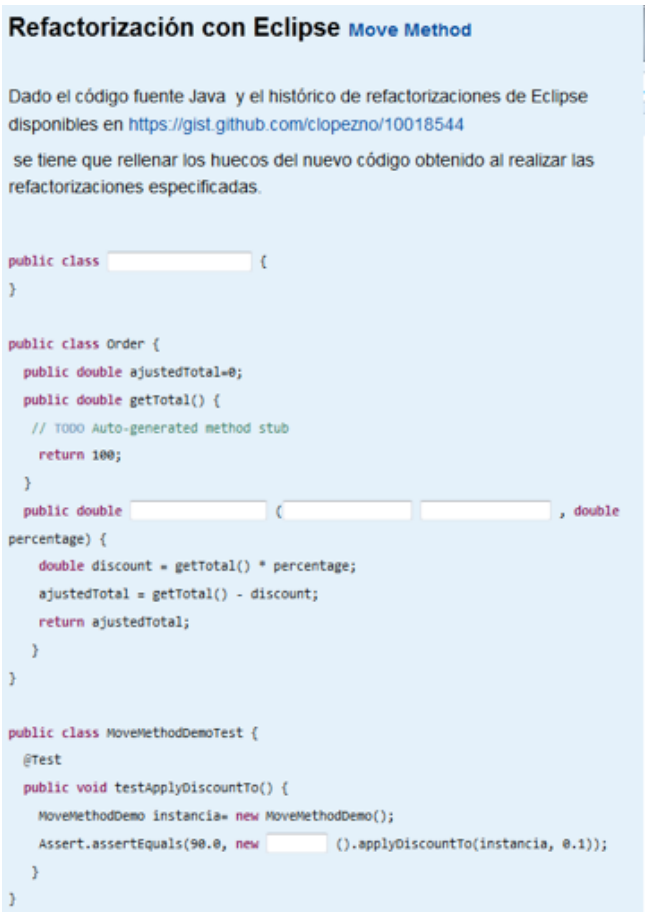

Fig. 5.Problemas de refactoring con preguntas cloze.

A medida que el alumno va aprendiendo, estas preguntas evolucionan en preguntas de texto libre o ensayo. La evolución consiste en ir eliminando del enunciado toda información relacionada con el código final después de refactorizar, para que la genere completamente el alumno. Como último paso evolutivo, en el diseño de problemas solo se proporciona el código fuente original, para que identifique algún defecto asociado y la secuencia de refactorizaciones que puede ayudar a eliminarlo. En el caso particular de la Fig. 5, la refactorización *Move Method* está motivada por existir el defecto de código denominado envidia de características (*Feature Envy*).

La funcionalidad denominada *fork*, asociada a cada Gist, permite crear ramas de desarrollo independientes del código original. Esto puede ser utilizado didácticamente para poder llevar a cabo ese proceso evolutivo del aprendizaje donde cada alumno desarrolle su solución a partir del enunciado inicial proporcionado por el profesor, y entregar como resultado el enlace de su evolución, como respuesta a una pregunta de texto libre o ensayo.

4.3. Estudio de casos basado en proceso de refactorización

En la asignatura se han seleccionado dos casos de estudio, ambos son referencia básica en el campo de refactorización orientadas a diseño. El primero es guiado por el profesor y con una documentación detallada paso a paso basada en el ejemplo del capítulo 1 del libro (Fowler *et al.*, 1999). El caso de estudio es monitorizado en GitHub. El profesor crea un repositorio con el código original y cada estudiante crea su propia rama de desarrollo (*fork*). Cada paso se corresponde con la aplicación de una refactorización, y se pide que se haga una revisión en su propia rama de desarrollo con un texto descriptivo de la revisión (*commit*).

El segundo caso de estudio es el propuesto en (Demeyer *et al.*, 2007), *Refactoring Lab Session*, y basado en el libro (Nierstrasz, Ducasse & Demeyer, 2009). Este caso de estudio es guiado mediante conjunto de tareas en su enunciado, y se realiza un seguimiento de las preguntas sobre la resolución de cada tarea con un foro de tipo pregunta respuesta. La actividad se realiza en grupos de dos participantes que realizan su clonación del repositorio proporcionado por el profesor (disponible en https://github.com/clopezno/reafactoring_lab_session.git). Los productos a entregar resultado del caso de estudio son dos: (1) un histórico de refactorizaciones realizadas con Eclipse, obtenido mediante la funcionalidad disponible en el menú refactor → *history y*, (2) un documento memoria con la dirección web al repositorio de control de versiones, junto con las respuestas a las preguntas de reflexión:

- *¿Se puede automatizar completamente el proceso de refactorización a través de herramientas?*
- *¿Qué relación encuentras entre el proceso de refactorización y la utilización de sistemas de control de tareas y versiones?*

En la Fig. 6 se describe la rúbrica utilizada en UBUVirtual para evaluar esta actividad. En ella se consideran tres dimensiones del trabajo: el producto software, el proceso para obtener el producto y las reflexiones obtenidas.

Proceso de refactoring	Sin entregar *0 puntos*	Refactorizaciones de un tipo *1 puntos*	Refactorizaciones de dos tipos *3 puntos*	Refactorizaciones de tres tipos *6 puntos*	Refactorizaciones de cuatro tipos *8 puntos*
Calidad de la solución	Sin entregar *0 puntos*	Existen las subclases Workstation y Printer *1 puntos*	Workstation y Printer tienen implementados correctamente al menos uno de estos tres métodos:printOn, printHtmlOn, printXmlOn *3 puntos*	Workstation y Printer tienen implementados correctamente tres métodos: printOn, printHtmlOn, printXmlOn *6 puntos*	Se elimina el código de tipo de la clase Nodo *8 puntos*
Preguntas de reflexión	No se justifica con la experiencia adquirida en el caso de estudio *1 puntos*		Son correctas parcialmente y se justifican con la experiencia adquirida en la práctica *3 puntos*		Son correctas y se justifican con la experiencia adquirida en la práctica *6 puntos*

Fig. 6. Rúbrica de evaluación del caso de estudio.

En la Fig. 7 se muestra las evidencias de seguimiento de las reflexiones realizadas por los grupos de alumnos. El profesor comenzaba indicando unas preguntas procedentes del enunciado indicando un duración en minutos por respuesta. Los grupos de alumnos después de realizar la tarea solicitada contestaban las preguntas planteadas en el foro. El profesor solo intervenía para corregir reflexiones erróneas que pudieran llevar a confusiones

a otros compañeros. La participación de los grupos fue casi completa y el profesor solo tuvo que aportar dos correcciones.

La Fig. 8, disponible en el repositorio de GitHub, se presenta un resumen gráfico del versionado del proceso de todos los grupos de prácticas.

Tema	Comenzado por	Grupo	Réplicas	Último mensaje
Extract Method 30 min + 5	LOPEZ NOZAL CARLOS		7	LOPEZ NOZAL CARLOS mar, 3 de jun de 2014, 19:28
Conclusión (20 min)	LOPEZ NOZAL CARLOS		6	IZQUIERDO AMO ROBERTO vie, 23 de may de 2014, 19:20
Transformar códigos de tipo (30 min + 5)	LOPEZ NOZAL CARLOS		5	ALAMO OLIVE JORGE vie, 23 de may de 2014, 11:48
Eliminar código de navegación (30 min + 5)	LOPEZ NOZAL CARLOS		6	ATIENZA GONZÁLEZ DAVID jue, 22 de may de 2014, 21:51
Mover el comportamiento cerca de los datos (30 min + 5)	LOPEZ NOZAL CARLOS		6	RAMOS SÁNCHEZ DAVID jue, 15 de may de 2014, 19:30
Habla con los de mantenimiento (10)	LOPEZ NOZAL CARLOS		6	RAMOS SÁNCHEZ DAVID jue, 15 de may de 2014, 18:31
Hacer una instalación de prueba (20 + 10)	LOPEZ NOZAL CARLOS		7	RAMOS SÁNCHEZ DAVID jue, 8 de may de 2014, 19:27
Lee todo el código en 5 minutos (10 +5)	LOPEZ NOZAL CARLOS		6	MIGUEL DE LA FUENTE DAVID jue, 8 de may de 2014, 18:50
Ojear la documentación 15 + 5	LOPEZ NOZAL CARLOS		6	RAMOS SÁNCHEZ DAVID jue, 8 de may de 2014, 18:22

Fig. 7. Foro pregunta respuesta para realizar un seguimiento de las reflexiones.

Cada línea se corresponde con la evolución temporal de una determinada versión del sistema software de un grupo de prácticas.

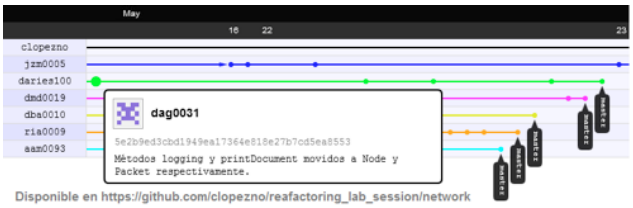

Fig. 8. Síntesis del versionado del proceso de todos los grupos de práctica.

Todos los grupos parten de una versión inicial que es la proporcionada por el profesor, que en este caso se corresponde con la primera línea. Se observa que a principios de mayo todos los grupos de prácticas hicieron su clonación. Cada punto dentro de la línea representa un cambio en el código ocasionado por una tarea de refactorización. Por cada cambio, se puede tener acceso al autor del cambio y su texto descriptivo. En el gráfico también se observa como la descripción textual del cambio utiliza correctamente la terminología de refactorización.

4.4. Seminario virtual

Después de adquirir un nivel de conocimiento analítico, como última actividad, se propone a los estudiantes la asistencia no obligatoria a un seminario impartido por una empresa. Parte del contenido del seminario está orientado a reflexionar cómo utilizan en la empresa los conocimientos adquiridos.

La historia del desarrollo de la actividad fue la siguiente. Primero se anunció la actividad en abierto en varias asignaturas y a profesores de asignaturas relacionadas. La publicidad del evento se realizó a través de la página de Escuela Politécnica Superior de la Universidad Burgos y se utilizó un cartel interactivo (disponible en el enlace http://goo.gl/n6LbdP). A la actividad asistieron presencialmente el 91 % (11 de 12) de los estudiantes de la asignatura y dos profesores. El seminario de impartió sincronizadamente por dos ponentes, uno presencial desde la Universidad de Burgos y otro desde la empresa ubicada en Valladolid. El sistema de videoconferencia utilizado fue *GoToMeeting*. A pesar de ser público el

enlace a la videoconferencia no se unió nadie desde Internet. El seminario se grabó y está publicado en *Youtube* (ver http://goo.gl/aW5vhh).

5. Conclusiones y Trabajos Futuros

El presente trabajo propone y describe e-actividades para la enseñanza de las tareas de refactorización a alumnos de último curso de un grado de Ingeniería Informática. La propuesta usa la taxonomía de Bloom para definir e-actividades con una complejidad gradual, lo que se plasma en un plan en el que el primer mes el esfuerzo del profesor (e.g. exposición de conceptos, secuenciación, creación y corrección de actividades de problemas) es muy alto. Después, ese esfuerzo disminuye, siendo los alumnos los que realizan el mayor desempeño al enfrentarse a los casos de estudio (ver Fig. 3).

Aunque la experiencia de aplicación de las e-actividades solo se ha realizado en el curso 2013-2014, se observan las siguientes consecuencias:

1. que las funciones del profesor son quizás más dinámicas de lo que serían con una metodología tradicional, pues seguirá creando nuevas e-actividades en los siguientes cursos. A todo ello se suma la renovación de las herramientas software, búsqueda de empresas implicadas en los seminarios virtuales, y su papel de moderador y guía de las actividades más interactivas. Esto último viene refrendado por la elevada asistencia a clase.
2. que los resultados de aprendizaje son buenos (100 % tasa de rendimiento en primera convocatoria, pese a existir notas de corte). Cuestiones, como que la propuesta permite homogeneizar conocimientos y habilidades en las primeras semanas, o motivar a los alumnos mediante la competitividad en la realización de los casos de estudio, así como la utilización de software profesional, apuntan como factores de éxito.

Consideramos que desde el punto de vista de las TIC el trabajo puede mejorar en varias líneas. Por un lado, utilizando más funcionalidades ofrecidas por GitHub para fines pedagógicos. En este sentido, en este trabajo no se han utilizado dos funcionalidades interesantes que GitHub asocia a cada proyecto: la gestión de tareas (*issues*) y los comentarios de revisión (*pull request*). Por otro lado, se pueden incorporar análisis visuales de la evolución del software de la interacción de los estudiantes con las e-actividades de refactorización. Para ello se puede usar algún *framework* como el propuesto en (González-Torres, García-Peñalvo & Therón, 2013).

6. Agradecimientos

Este trabajo ha sido realizado por el Grupo de Innovación Docente de la Universidad de Burgos DIGIT y financiada con cargo a la Convocatoria de Apoyo a Proyectos de Innovación y Mejora Docente, convocada por el Vicerrectorado de Profesorado y Personal de Administración y Servicios de la UBU. Clave orgánica [30.18.10.A2].

A la Empresa Códice Software por su colaboración y disponibilidad en la organización del seminario. A los alumnos que han participado en la experiencia por dar autorización escrita a presentar parte de su información personal con fines de divulgación educativa.

7. Referencias

Bourque, P., & Fairley, R. E.. (2014). *Guide to the Software Engineering Body of Knowledge SWEBOK, Version 3.0.* IEEE Computer Society. http://www.swebok.org/

Bourque, P., Buglione, L., Abran, A., & April, A. (2003). Bloom's Taxonomy Levels for Three Software Engineer Profiles. En *STEP* (pp. 123-129). http://dx.doi.org/10.1109/step.2003.6

Britton, J., & Berglund, T. (2013). Using version control in the classroom. En *Proceeding of the 44th ACM technical symposium on Computer science education* (pp. 753-753). Denver, Colorado, USA: ACM.

Brown, W. J. (1998). AntiPatterns: refactoring software, architectures, and projects in crisis. New York: Wiley.

Cabero Almenara, J, & Román Graván, P. (2005). *E-actividades: un referente básico para la formación en Internet.* Madrid: Eduforma Editores.

Demeyer, S., Rysselberghe, F. V., Gîrba, T., Ratzinger, J., Marinescu, R., Matthias R., & El-Ramly, M. (2007). The LAN-simulation: A Refactoring Lab Session. En *WRT* (pp. 52-53).

Fowler, M., Beck, K., Brant, J., Opdyke, W., & Roberts, D. (1999). *Refactoring: Improving the Design of Existing Code.* Addison-Wesley.

González-Torres, A., García-Peñalvo, F. J., & Therón, R. (2013). Human–computer interaction in evolutionary visual software analytics. *Computers in Human Behavior, 29*(2), 486-495. http://dx.doi.org/10.1016/j.chb.2012.01.013

Kelleher, J. (2014). Employing git in the

classroom. *Computer Applications and Information Systems (WCCAIS), 2014 World Congress on*, 1-4. http://dx.doi.org/10.1109/WCCAIS.2014.6916568

Kerievsky, J. (2004). *Refactoring to patterns.* Addison-Wesley. http://dx.doi.org/10.1007/978-3-540-27777-4_54

Lawrance, J., Jung, S., & Wiseman, C. (2013). Git on the cloud in the classroom. En *Proceeding of the 44th ACM technical symposium on Computer science education* (pp. 639-644). Denver, Colorado, USA: ACM. http://dx.doi.org/10.1145/2445196.2445386

Lippert, M., & Roock, S. (2006). Refactoring in Large Software Projects: Performing Complex Restructurings Successfully. John Wiley & Sons.

Lopez, C., Alonso, J. M., Marticorena, R., & Maudes, J. M. (2014). Design of e-activities for the learning of code refactoring tasks. En *2014 International Symposium on Computers in Education (SIIE)* (pp. 35-40). http://dx.doi.org/10.1109/SIIE.2014.7017701

Meneses, E. L., Fernández, G. D., & Ballesteros-Regaña, C. (2011). E-actividades: elementos constitutivos para la calidad de la praxis educativa digital. En *La práctica educativa en la Sociedad de la Información: Innovación a través de la investigación* (pp. 267-282). Editorial Marfil.

Nierstrasz, O., Ducasse, S., & Demeyer, S. (2009). *Object-Oriented Reengineering Patterns.* Square Bracket Associates.

Wake, W. C. (2004). *Refactoring workbook.* Boston: Addison-Wesley.