Prácticas de Laboratorio en Ruby para 'Lenguajes y Paradigmas de Programación'

Casiano Rodríguez León, Coromoto León Hernández, Gara Miranda Valladares, Eduardo Segredo González y Carlos Segura González
Escuela Técnica Superior de Ingeniería Informática
Universidad de La Laguna

Este trabajo se ha realizado en el marco de la convocatoria de Proyectos de Innovación Docente 2012/2013 del Vicerrectorado de Calidad Institucional e Innovación Educativa de la Universidad de La Laguna. El trabajo de Eduardo Segredo y Carlos Segura ha sido financiado por el Ministerio de Educación con las becas de Formación del Profesorado Universitario con referencia FPU-AP2009-0457 y FPU-AP2008-03213 respectivamente.

La correspondencia relativa a este trabajo debe dirigirse a Casiano Rodríguez León, Dpto. de Estadística, I.O. y Computación. Universidad de La Laguna, Edificio de Física y Matemáticas. Av. Astrofísico Fco. Sánchez s/n. 38271 La Laguna, Tenerife, España (Spain). Teléfono: +34 922 318 187. Fax: + 34 922 318 170. E-mail: crguezl@ull.es

Publicado en:
INNOVACIÓN DOCENTE EN LA EDUCACIÓN SUPERIOR: UNA RECOPILACIÓN DE EXPERIENCIAS PRÁCTICAS APLICADAS

Editado por:
Vicerrectorado de Calidad Institucional e Innovación Educativa.
Universidad de La Laguna
La Laguna 2013

Coordinadoras: Carmen Inés Ruiz de la Rosa y Jacqueline O´Dwyer

ISBN: 978-84-695-9951-8

ISBN: 978-84-695-9951-8

Prácticas de Laboratorio en Ruby para 'Lenguajes y Paradigmas de Programación'

Resumen

En este trabajo se presenta el conjunto de actividades prácticas de la asignatura obligatoria de tercer curso del Grado en Ingeniería Informática de la Universidad de La Laguna denominada "Lenguajes y paradigmas de programación". En la asignatura se profundiza en el estudio de Paradigmas de Programación que incluyen: la 'Programación Orientada a Objetos', la 'Programación Declarativa' y la 'Programación Concurrente, Distribuida y Paralela'. Se plantea el uso del lenguaje de programación 'Ruby' para el desarrollo de las prácticas de laboratorio, haciendo especial énfasis en la aplicación de metodologías ágiles para el desarrollo de software como son: el uso de sistemas de control de versiones, el desarrollo dirigido por pruebas y por la conducta, la integración y comprobación continua y la elaboración de documentación. Cada una de las prácticas de laboratorio se asocia con una o varias herramientas, la mayoría de ellas en la nube, por ejemplo, para el control de versiones se usa *git* y los servicios de hospedaje *GitHub* y *Bitbucket*. Así mismo, la comunicación entre los estudiantes y el profesado se gestiona a través de un aula virtual *Moodle* hospedada en el campus virtual institucional. Las principales dificultades observadas en la implantación han sido la falta de motivación de una parte del alumnado y la escasez de tiempo para completar todo el conjunto de competencias, objetivos y contenidos definidos en el plan de estudio.

Palabras clave

Paradigmas de Programación, Lenguajes de Programación, Ingeniería del Software, Metodologías ágiles, Sistemas de control de versiones.

ISBN: 978-84-695-9951-8

Ruby Labs for "Programming Languages and Paradigms"

Abstract

This paper presents a set of practical activities for the subject 'Programming Languages and Paradigms'. This subject is taught in the third course of the University of La Laguna Computer Engineering degree. The course contents include the study of programming paradigms like 'Object Oriented Programming', 'Declarative Programming' and 'Concurrent, Parallel and Distributed Programming'. The programming labs are developed using Ruby, a pure Object Oriented language. We make particular emphasis on the application of agile methodologies to software development such as: the use of Version Control Systems, Behaviour and Test Driven Development, Continuous Testing, Continuous Integration and Documentation. Each practical exercise is associated with one or more tools, most of them in the cloud. For instance, *Git* is used for control version and *GitHub* and *Bitbucket* as hosting services. Also, a *Moodle* virtual classroom is used for communication among students and instructors. The main difficulties encountered in the implementation was the lack of motivation of some of the students and the lack of time to complete the full set of competencies, objectives and content defined in the curriculum.

Keywords

Programming Paradigms, Programming Languages, Software Engineering, Agile methodologies, Control version systems.

ISBN: 978-84-695-9951-8

1. Introducción y objetivos

La asignatura 'Lenguajes y Paradigmas de Programación'– LPP –, del plan de estudios del Grado en Ingeniería Informática de la Universidad de La Laguna, tiene seis créditos y se imparte en el primer semestre. El objetivo principal de esta asignatura es estudiar los principales conceptos presentes en los lenguajes y modelos de programación. En ella se intenta responder a preguntas como: ¿Qué elementos son comunes a los lenguajes de programación? ¿Qué características tienen? ¿Cuáles son los elementos esenciales y los accesorios? ¿Qué hace que lenguaje de programación tenga un buen diseño?

Las competencias que se definen en el Plan de Estudios para LPP son las siguientes:

- Capacidad para analizar, diseñar, construir y mantener aplicaciones de forma robusta, segura y eficiente, eligiendo el paradigma y los lenguajes de programación más adecuados.
- Conocimiento y aplicación de los principios fundamentales y técnicas básicas de la programación paralela, concurrente, distribuida y de tiempo real.

Para el desarrollo de las mismas, se ha considerado el estudio de cinco temas que se vertebran en la idea de construcción de abstracciones. El elemento central de la programación es la abstracción. Un lenguaje de programación proporciona mecanismos de abstracción que permiten expresar una solución informática en un lenguaje cercano al dominio del problema que se pretende resolver. Se estudiarán las distintas formas de plasmar dichas abstracciones:

1. Modelos de Programación.
2. Programación Imperativa.
3. Programación Orientada a Objetos.
4. Programación Declarativa.
5. Programación Concurrente, Paralela y Distribuida.

Las asignaturas con un perfil similar impartidas en las titulaciones de Grado en Informática disponibles en otras universidades a nivel nacional tienen enfoques parecidos, si bien, tienen menor contenido puesto que no cubren la segunda de las competencias. Por ejemplo, en la Universidad de Alicante no se incluye el bloque de Programación Concurrente, Paralela y Distribuida. No es posible hacer un estudio

ISBN: 978-84-695-9951-8

comparativo porque no hay uniformidad en la distribución de las competencias y en muchos casos están dispersas en más de una asignatura. La mayor diferencia detectada no está en los contenidos, sino en las herramientas utilizadas para el desarrollo de las prácticas de laboratorio. Es decir, el asunto más polémico a la hora de implantar una asignatura de este corte estriba en qué lenguaje o lenguajes de programación utilizar.

El objetivo de este trabajo es describir el material docente elaborado para impartir las prácticas de laboratorio de la asignatura LPP. El mismo contempla ejercicios guiados y ejercicios autónomos que permiten la adquisición de las competencias. Particularmente, en los ejercicios guiados y autónomos de prácticas de laboratorio, se pretende introducir al alumnado en la metodología de desarrollo de una aplicación de código abierto. Concretamente, se propone utilizar: el lenguaje de programación de alto nivel *Ruby* de Matsumoto (2007), el sistema de control de versiones *Git* a través de la propuesta de Chacon (2009), los sistemas de hospedaje en la nube para el trabajo en grupo *GitHub* y *Bitbucket*, el sistema *RVM* de gestión de los intérpretes Ruby, la herramienta *Bundler* para la gestión de bibliotecas, la herramienta *Rake* para la ejecución de tareas, la biblioteca *Test::Unit* para el desarrollo de pruebas unitarias, *RSpec* para el desarrollo dirigido por el comportamiento, *Guard* para la comprobación continua, *Travis* para la integración continua, para la generación de documentación automática *Rubydoc* y para el seguimiento de la compatibilidad *Gemnasium*.

Otro de los objetivos es gestionar de manera eficiente una asignatura con un alto número de alumnos matriculados. Esto conlleva la implicación de un gran número de profesores. Estos números demandan un alto grado de sincronización y colaboración por parte del profesorado que se consigue mediante reuniones presenciales, el entorno institucional para la docencia virtual basado en *Moodle* como el que describe Rice (2008), así como herramientas en la nube, fundamentalmente: *Gmail, Google Calendar, Google Drive* y *Hangouts*.

El resto del presente trabajo se organiza tal y como sigue: en la Sección 2 se describen la metodología docente y las herramientas a utilizar para el desarrollo de las prácticas de la asignatura, el tipo de evaluación y la organización semanal. A continuación, en la Sección 3 se describen los recursos docentes desarrollados. Por último, se enumeran las conclusiones.

ISBN: 978-84-695-9951-8

2. Metodología

Bohem (1976) define la *Ingeniería del Software* como la aplicación práctica del conocimiento científico al diseño y construcción de programas de computadora y a la documentación asociada requerida para desarrollar, operar y mantenerlos. El desarrollo de software requiere la realización de numerosas tareas que generalmente son agrupadas en etapas. Al conjunto de estas etapas se le denomina 'ciclo de vida del software'. La *Programación* es la etapa común en todos los modelos de ciclo de vida. *Programar* consiste en plasmar mediante código para una computadora un diseño. La complejidad y la duración de esta etapa está íntimamente relacionada al o a los *lenguajes de programación* utilizados, así como al diseño previamente realizado. El objetivo de la asignatura LPP es estudiar los conceptos fundamentales de los lenguajes informáticos y de los modelos de programación.

Existen muchísimos *lenguajes de programación* y es muy complicado conocer a fondo las posibilidades que presentan cada uno de ellos. Hay lenguajes de propósito general, pero también los hay de propósito específico para áreas tan diversas como la gestión administrativa o la inteligencia artificial. *Ruby* es un lenguaje de guiones (en inglés, *scripts*), moderno y orientado a objetos que combina una importante flexibilidad con una alta productividad. Ruby fue diseñado por Matsumoto (2002) e incorpora algunas de las mejores características de otros lenguajes como *Smalltalk*, *Java* y *Perl*. Tiene un amplio alcance puesto que se utiliza en aplicaciones muy diversas que van desde el Desarrollo Web hasta la Simulación de entornos complejos.

Algunas de las razones que han llevado a la elección del lenguaje Ruby para el desarrollo de las prácticas de la asignatura LPP son las siguientes:

- Mediante su uso se pueden complementar las características del paradigma imperativo y el paradigma funcional.
- Simplifica declaraciones, estructuras de datos y modelos de sentencia sin perder potencia y permite que el programador, se desarrolle de forma adecuada.
- Es un lenguaje dinámico e interpretado.
- Promueve el uso de buenas prácticas de programación sin perder usabilidad.
- Es un lenguaje multiplataforma que se integra perfectamente en una gran cantidad de arquitecturas; puede ejecutarse, incluso, en dispositivos móviles.

ISBN: 978-84-695-9951-8

- Se puede extender su funcionamiento no sólo mediante bibliotecas escritas en Ruby, sino que puede ampliarse utilizando el lenguaje C.
- Permite utilizar la expresión más simple de un algoritmo mediante un programa.

La elección del lenguaje Ruby ha condicionado la selección de la metodología de desarrollo de las prácticas de laboratorio de LPP de entre los disponibles en la Ingeniería del Software. Entre dichos *métodos* se encuentran los denominados *ágiles*, basados en un desarrollo iterativo e incremental, donde los requisitos y soluciones evolucionan mediante la colaboración de grupos auto-organizados y multidisciplinares. Existen muchos métodos de desarrollo ágil y la mayoría pone más énfasis en la adaptabilidad que en la previsibilidad. Al software desarrollado en una unidad de tiempo se le llama *iteración*, y su duración debe ser de una a cuatro semanas. Cada iteración del ciclo de vida incluye: *planificación*, *análisis de requerimientos*, *diseño*, *codificación*, *revisión* y *documentación*. El objetivo de cada iteración es tener una aplicación sin errores. Al final de cada iteración el equipo vuelve a evaluar las prioridades del proyecto. El más destacado de los procesos ágiles de desarrollo de software es la *Programación Extrema* – XP (el acrónimo proviene del nombre en inglés: *eXtreme Programming*). Se trata de una metodología de desarrollo de la Ingeniería del Software formulada por Kent Beck (2005) y es el que se seguirá en el desarrollo de los laboratorios de prácticas de LPP.

Cuando los programadores pueden *leer*, *modificar* y *redistribuir* el código fuente de un programa, éste evoluciona, se desarrolla y mejora. Los usuarios de dicho software lo adaptan a sus necesidades y corrigen sus errores a una velocidad impresionante. Esa velocidad es mayor a la aplicada en el desarrollo de software cerrado, por lo que el concepto de código abierto (del inglés, *open source*) da lugar a la producción de un mejor software. Los laboratorios de LPP tienen como objetivo el mostrar los pasos a seguir para desarrollar una biblioteca software de código abierto siguiendo los estándares. Se responderá a preguntas como las siguientes: ¿Cómo se debe estructurar el proyecto? ¿Qué debe incluir en términos de documentación? ¿Qué herramientas se pueden utilizar para desarrollar una biblioteca amigable en la que otros puedan contribuir? Así pues, el resultado final de las prácticas de laboratorio de LPP será una biblioteca de software (*library*, en inglés). En el lenguaje de programación Ruby a una biblioteca se le denomina *gema*.

ISBN: 978-84-695-9951-8

Finalmente, se ha de tener en cuenta que la competencia de trabajo en equipo se impone a la individualización en el desarrollo de software. Las tareas pueden tener tal grado de dificultad que hagan imposible su resolución de forma individual. Es por este motivo, que las organizaciones del trabajo reclaman, hoy más que nunca, la competencia transversal de trabajo en equipo. Por ello, las prácticas de laboratorio de LPP se realizarán en equipos de dos personas. Para que los miembros del equipo mantengan cada versión de un ejercicio de laboratorio dado han de utilizar un Sistema de Control de Versiones (*Version Control System* en inglés). El control de versiones es un sistema que registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo, de modo que se puede recuperar versiones específicas más adelante. La herramienta elegida, siguiendo los estándares de código abierto, es *Git*.

2.1. Sistema de evaluación

Las prácticas de laboratorio se evalúan de forma continua realizando una defensa oral del trabajo realizado en las sesiones de laboratorio. Puesto que se cuenta con múltiples grupos de prácticas asignados a profesorado diferente, para realizar una valoración objetiva se establece una rúbrica. En la Tabla 1 se presenta un ejemplo de rúbrica de corrección.

Tabla 1: Ejemplo de Rúbrica para la corrección de la práctica 'Programación imperativa con Ruby'

Criterio	Valores
¿El programa se ejecuta correctamente y sin errores?	Si/No
¿Es el número de subprogramas implementado suficiente para mantener la modularidad?	Si/No
¿Es el código simple y claro?	Si/No
¿Está bien documentado?	Si/No
Valore la participación de los miembros del equipo en el desarrollo del proyecto.	No adecuada/Adecuada
Valore el número de confirmaciones (*commits*) realizados.	No adecuada/Adecuada
Valore la información que proporcionan los mensajes de las confirmaciones (*commits*) realizadas.	No adecuada/Adecuada
Valore el número de ramas creadas	No adecuada/Adecuada
Valore el número de fusiones realizadas.	No adecuada/Adecuada
Valore el número de conflictos que se han resuelto.	No adecuada/Adecuada

ISBN: 978-84-695-9951-8

2.2. Organigrama de las prácticas de laboratorio

La asignatura cuenta con 60 horas de trabajo presencial y 90 horas de trabajo autónomo, cumpliendo con las 150 horas de los 6 créditos ECTS (*European Credit Transfer System*) establecidos. Se debe impartir en 15 semanas lectivas y se plantea la realización de 13 ejercicios prácticos de laboratorio. El contenido de los ejercicios afianza los conceptos sobre modelos de programación tratados en teoría y permite la introducción gradual de las herramientas necesarias para cumplir con el objetivo de desarrollar una biblioteca Ruby de código abierto siguiendo los estándares. La Tabla 2 muestra la relación de prácticas de laboratorio, sus contenidos y los recursos elaborados y puestos a disposición del alumnado para el desarrollo de las mismas.

Tabla 2: Sesiones de prácticas, contenidos y recursos generados para cada una de ellas

Práctica	Contenidos	Recursos generados
1	Sistema de control de versiones *Git*. Creación de cuentas en *GitHub*.	Batería de ejercicios guiados
2	Creación y compartición de cuentas en *Bitbucket*.	Video tutorial
3	Compilación vs. Interpretación. Creación y fusión de ramas en Git.	Batería de ejercicios guiados
4	Primeros pasos en Ruby. Gestión de intérpretes de Ruby (*RVM – Ruby Version Manager*).	Batería de ejercicios guiados
5	Programación Imperativa.	Ejercicio individual
6	Programación Orientada a Objetos. Creación de gemas (*Bundler*).	Ejercicio de trabajo en grupo
7	POO. Pruebas Unitarias (*Test::Unit*).	Ejercicio de trabajo en grupo
8	POO. Desarrollo Dirigido por pruebas (*RSpec*). Definición de tareas (*Rake*).	Ejercicio de trabajo en grupo
9	POO. Comprobación continua (*Guard*).	Ejercicio de trabajo en grupo
10	POO. Integración continua (*Travis*).	Ejercicio de trabajo en grupo
11	Programación Funcional. Generación de documentación (*RDoc*).	Ejercicio de trabajo en grupo
12	PF. Compatibilidad y dependencias (*Gemnasium*).	Ejercicio de trabajo en grupo
13	Programación Concurrente.	Ejercicio de trabajo en grupo

2.3. Infraestructura

A continuación se relacionan las herramientas que se utilizan en la asignatura LPP para construir una gema que siga las mejores prácticas de la comunidad de código abierto:

- *Git* para el control de versiones distribuido.

ISBN: 978-84-695-9951-8

- RVM (*Ruby Version Manager*) para la gestión de los intérpretes de Ruby.
- *Bundler* para la gestión del arranque, los directorios, las dependencias y la liberación de una gema.
- *Test::Unit* (integrada en Ruby) para la definición de Pruebas Unitarias.
- *RSpec* para el Desarrollo Dirigido por Pruebas (en inglés, *Test Driven Development*).
- *Rake* para la ejecución de tareas.
- *Guard* para la automatización del proceso de desarrollo.
- *RDoc* para la generación de documentación automática.
- *Gemnasium* para el seguimiento de las dependencias que aseguren que la biblioteca siempre es compatible con las últimas versiones de las bibliotecas que utilice.

Además, es necesario tener acceso a los siguientes servicios de hospedaje en la nube:

- *GitHub* y *Bitbucket* para el alojamiento del código fuente, la emisión de informes y la documentación wiki de las gemas.
- *Travis* para la integración continua de proyectos de código abierto.
- *Rubygems* actuando como sevidor para distribuir gemas en la comunidad de código abierto.

Finalmente, la gestión on-line institucional de la asignatura se realiza mediante un Aula Virtual desarrollada en *Moodle*. La Figura 1 muestra una captura de pantalla de la misma.

Figura 1: Aula virtual Moodle de la asignatura Lenguajes y Paradigmas de Programación

ISBN: 978-84-695-9951-8

3. Prácticas de Laboratorio / Resultados

En esta sección se describe el proceso de desarrollo de los ejercicios prácticos de laboratorio desde un punto de vista general. El objetivo es ofrecer una vista panorámica inicial que permita fijar el mapa de prácticas de manera que desde un principio se conozca la ruta que se va a seguir.

Los ejercicios prácticos de laboratorio se llevarán a cabo en equipos de un máximo de dos personas. Puesto que el objetivo es el desarrollo de una gema, es decir, una biblioteca en Ruby, se va a programar código fuente y el mismo tiene que ser compartido por los miembros del equipo. Haciendo uso de *Git*, los miembros del equipo deben ir almacenando las diferentes versiones de todo el material desarrollado para las prácticas y compartirlo a través de un repositorio remoto situado en *GitHub* o en *Bitbucket*. De esta forma todos los miembros del equipo tienen disponible la última versión del material elaborado para su descarga. Además, en el repositorio remoto hay que dar de alta como colaboradores al profesorado de la asignatura de manera que puedan acceder al mismo para realizar las labores de seguimiento y de evaluación. Se ha programado las tres primeras sesiones de prácticas de laboratorio con ejercicios guiados para introducir el uso y configuración de *Git* y para la creación y compartición de cuentas en *GitHub* y *Bitbucket*.

La cuarta sesión de prácticas consiste en una batería de ejercicios guiados para dar los primeros pasos en Ruby. En ella se utiliza el intérprete interactivo de Ruby (en inglés, *irb – Interactive Ruby*) para jugar con cadenas, rangos, símbolos, *arrays* y *hashes*. Por lo tanto, es conveniente establecer como intérprete por defecto de Ruby la última versión liberada. Para gestionar las distintas versiones del intérprete se utiliza la herramienta *RVM* (*Ruby Version Manager*). *RVM* permite instalar, administrar y ejecutar con múltiples versiones de Ruby. Esto quiere decir que es posible tener instalado en el ordenador Ruby 1.8.7, Ruby 1.9.2 y Ruby 2.0 y usar en cualquier momento la versión más conveniente.

La programación imperativa es el modelo más antiguo y aunque Ruby sea un lenguaje de programación completamente Orientado a Objetos es posible utilizarlo en la quinta sesión de prácticas para introducir la sintaxis y la semántica de las sentencias condicionales y las iterativas.

ISBN: 978-84-695-9951-8

Todos los lenguajes de programación tienen soporte para las bibliotecas. Las bibliotecas son colecciones de código reutilizable que suelen agruparse en torno a un propósito común. Se pueden encontrar bibliotecas para casi cualquier cosa, desde el procesamiento avanzado de cadenas a funciones matemáticas de nivel superior. La integración de herramientas y plataformas se ha convertido en un estándar. La mayoría de los sistemas operativos, aplicaciones y herramientas permiten incorporar o quitar funcionalidades a partir de bibliotecas que se distribuyen a través de Internet. Mientras todos los lenguajes de programación tienen bibliotecas, no todos los lenguajes de programación suelen contar con un sistema para descargar e instalar esas bibliotecas tan simple y elegante como Ruby. Su gestor de bibliotecas se denomina *RubyGems*. Este sistema proporciona un formato estándar y auto-contenido con el objetivo de distribuir programas o bibliotecas en Ruby al que denomina *gema*. Además, se cuenta con la herramienta *Bundler* para gestionar la instalación y un servicio de hospedaje *Rubygems* que actúa como servidor para la distribución de las bibliotecas. *Bundler* permite administrar de forma sencilla y centralizada todas las dependencias de una gema. *Bundler* es en sí mismo una gema Ruby y puede ser instalado desde la línea de comandos con: `gem install Bundler`. El archivo donde se le especifican a *Bundler* los nombres de las bibliotecas que la aplicación necesita para funcionar se denomina `Gemfile`. *Bundler* encontrará automáticamente las versiones compatibles de las bibliotecas especificadas cuando se ejecute la línea de comandos: `bundle install`. Una gema es esencialmente un directorio con una estructura especial que contiene el código fuente, la descripción, la documentación y las pruebas de una biblioteca Ruby. La Figura 2 muestra la estructura de archivos normal de una gema. En la sexta sesión de prácticas de laboratorio se aborda la creación de una gema simple usando el paradigma de Programación Orientada a Objetos.

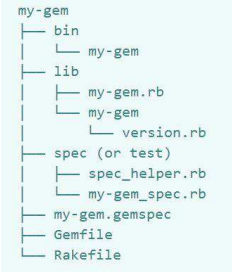

Figura 2: Estructura de una gema

ISBN: 978-84-695-9951-8

Diseñar y desarrollar pruebas para una gema es fundamental puesto que es una práctica generalizada su uso para controlar la calidad de un proyecto software. Si una gema no tiene pruebas, no hay manera de detectar la inclusión de errores o la regresión entre versiones. Las dos herramientas más utilizadas para la elaboración de pruebas en Ruby son *Test::Unit* y *RSpec*. En la séptima y octava sesiones de prácticas se introducen cada una de ellas. La filosofía de las pruebas unitarias consiste en definir mediante afirmaciones (*assert*, en inglés) certezas sobre cómo se espera que funcionen ciertas líneas de código. *Test::Unit* es la gema Ruby que proporciona los métodos necesarios para fijar las pruebas y se incluye de forma estándar en sus distribuciones. Por otro lado, el Desarrollo Dirigido por Pruebas (*Test Driven Development – TDD*, en inglés) se basa en definir en primer lugar la expectativa de cómo deberían funcionar ciertas líneas de código y a continuación implementarlas para que hagan lo esperado. La gema *RSpec* proporciona métodos de descripción, definición y comprobación para definir estos test. Es una práctica habitual entre todos los proyectos Ruby, tanto aplicaciones como gemas, el ejecutar pruebas para un proyecto a través de una tarea de *Rake*. La gema *Rake* permite a los desarrolladores definir tareas sencillas (*task*, en inglés) en un fichero `Rakefile` que se ejecutan mediante la línea de comando `rake`. La mayoría de las gemas establecen tareas para la ejecución de las pruebas unitarias, para la ejecución de las expectativas, para la generación de la documentación y para su construcción.

En la novena sesión de prácticas de laboratorio se introduce la Comprobación Continua (*Continuos Testing*, en inglés). La gema *Guard* permite observar los cambios que se producen en los archivos del proyecto que estén bajo vigilancia y realizar acciones basadas en ese cambio. En el fichero `Guardfile` se han de relacionar los archivos que se han de vigilar y la tarea a ejecutar cuando se produzcan cambios en el mismo. Para ponerlo en funcionamiento se ejecuta `bundle exec guard` que lanzará un proceso vigilante que trabajará de forma concurrente con los de desarrollo.

La Integración Continua (en inglés, *Continous Integration*) es el proceso de ejecutar automáticamente las pruebas de un proyecto cuando se empuja una nueva versión en el repositorio remoto central. Esta práctica se introduce en la décima sesión. *Travis - CI* es un servicio web que permite probar el código en distintas versiones de Ruby. Incluso si no se desarrolla en versiones anteriores o en implementaciones alternativas como *JRuby* o *Rubinius*, *Travis* puede ejecutar las pruebas sobre ellas y notificar cualquier fallo.

ISBN: 978-84-695-9951-8

El paradigma de programación utilizado en las práctica de la seis a la diez ha sido el de Programación Orientada a Objetos. En la undécima sesión de prácticas se pasa al modelo de Programación Funcional y se introduce la gema *RDoc*. La documentación del código es otra de las tareas del desarrollo de un proyecto software y *RDoc* es el generador de documentación de Ruby estándar. *RDoc* realiza un análisis sintáctico recursivo de un directorio de proyecto Ruby en busca de comentarios en todos los archivos `.rb`, `.rbw` y `.c`. Los comentarios se especifican al comienzo de cada fichero y de cada función siguiendo un formato específico. *RDoc* es útil incluso si el código fuente no contiene comentarios explícitos puesto que se realiza un análisis del código fuente. El formato de salida es HTML y crea el archivo en el subdirectorio `doc`.

Gemnasium es una herramienta para el seguimiento de las dependencias que asegura que la gema en desarrollo siempre es compatible con las últimas versiones de las bibliotecas que utiliza. Se introduce en la décimo segunda sesión de prácticas.

Finalmente, en la última sesión de prácticas se introduce un ejercicio de programación concurrente para cubrir la segunda de las competencias de la asignatura LPP.

4. Discusión / Conclusiones

La acción innovadora de la asignatura Lenguajes y Paradigmas de Programación del Grado en Ingeniería Informática impartido en la Universidad de La Laguna se enmarca en el bloque de 'Innovación en metodologías y estrategias docentes'. La principal aportación es el estudio de los Paradigmas de Programación haciendo uso del lenguaje de programación Ruby. Así mismo, es novedoso el especial énfasis realizado en la aplicación de metodologías ágiles para el desarrollo de software como son: el uso de sistemas de control de versiones, el desarrollo dirigido por pruebas y por la conducta, la integración y comprobación continua y la elaboración de documentación.

Se ha diseñado una batería de ejercicios prácticos de laboratorio y elaborado un material reutilizable para cubrir las competencias de la asignatura. Cada una de las prácticas de laboratorio se asocia con una o varias herramientas. Concretamente, se utilizan: *Git* para el control de versiones distribuido. *RVM* para la gestión de los intérpretes de Ruby. *Bundler* para la gestión del arranque, los directorios, las dependencias y la liberación de una gema. *Test::Unit* para la definición de Pruebas Unitarias. *RSpec* para el Desarrollo Dirigido por Pruebas. *Rake* para la ejecución de tareas. *Guard* para la automatización

ISBN: 978-84-695-9951-8

del proceso de desarrollo. *RDoc* para la generación automática de documentación. *Gemnasium* para el seguimiento de las dependencias que aseguren que la biblioteca siempre es compatible con las últimas versiones de las bibliotecas que utilice. Además, se propone el uso de los siguientes servicios de hospedaje en la nube: *GitHub* y *Bitbucket* para el alojamiento del código fuente, *Travis-CI* para la integración continua y *Rubygems* como servidor para distribuir gemas.

El material elaborado también incluye un Aula Virtual *Moodle* hospedada en el campus virtual institucional. A través de ella es posible gestionar de manera eficiente el alto número de alumnos matriculados de la asignatura y mejorar el proceso de coordinación del alto número de profesores de la asignatura. Además, para la colaboración y sincronización del profesorado se ha propuesto el uso de herramientas en la nube como *Gmail*, *Google Calendar*, *Google Drive* , Google+ y *Hangouts*.

Las principales dificultades observadas en la implantación han sido la falta de motivación de una parte del alumnado y la escasez de tiempo para completar el conjunto completo de competencias, objetivos y contenidos definidos en el plan de estudio.

Referencias

Chacon, Scott. (2009) *Pro Git*. (Primera Edición). Apress.

Beck, Ken (2005) *Extreme Programming Explained: Embrace Change*, 2nd Edition. Escrito junto con Cynthia Andres. Addison-Wesley.

Matsumoto, Yukihiro. (2002) *Ruby in a nutshell - a desktop quick reference*. O'Reilly.

Rice, W. H. IV. (2008) *Moodle 1.9 E-Learning Course Development: A complete guide to successful learning using Moodle*. Packt Publishing.