Facultad de Ciencias

Mantenimiento y modularización de EvalCode para su puesta en marcha en plataformas Moodle

(Maintenance and modularization of EvalCode for its implementation on Moodle platforms)

Trabajo de fin de grado
para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autor: Guillermo Quintana Pelayo
Director: Héctor Pérez Tijero
Co-director: Mario Aldea Rivas

Junio de 2019

Agradecimientos

A mi hermano Emérito, por haberme servido siempre de inspiración y ser un modelo a seguir. Gracias por creer en mí y en mis ideas.
A mis padres, por su apoyo sin medida en todo aquello que he querido emprender.
A mis amigos, por estar siempre a mi lado y ser mi principal fuente de sonrisas.
A la Universidad de Cantabria y en especial al Dpto. de Igeniería Informática de la Facultad de Ciencias por darme la oportunidad de desarrollar este trabajo con un excelente equipo y por sentime acogido y valorado desde el primer instante.

Mantenimiento y modularización de EvalCode para su puesta en marcha en plataformas Moodle

Resumen

palabras clave: Moodle, Aprendizaje automático, Evaluación automática

El uso de sistemas de gestión y aprendizaje *online* como Moodle está en constante auge y, cada vez más, se utiliza como medio principal de comunicación con los alumnos, ya sea para proporcionarles material de estudio o para la entrega de trabajos y prácticas. Este proyecto consiste en la recuperación, mantenimiento, mejora y puesta en marcha de una herramienta de evaluación automática de ejercicios de programación integrada en la plataforma Moodle: EvalCode. Esta herramienta permite evaluar las entregas de los alumnos desde su subida a la plataforma y generar datos de realimentación, tanto para alumnos como profesores. La utilización de un sistema como éste ayuda a los docentes a reducir el trabajo asociado a la evaluación continuada y, a su vez, es capaz de proporcionar a los alumnos una idea inmediata de las deficiencias de su ejercicio. El trabajo parte de una versión previa de EvalCode que no pudo quedar completa y funcional para su explotación real. Se plantea como objetivo añadir funcionalidades, corregir las que no estén completas y realizar un despliegue del sistema para hacer pruebas reales con los alumnos de Ingeniería Informática de la Universidad de Cantabria.

(Maintenance and modularization of EvalCode for its implementation on Moodle platforms)

Abstract

keywords: Moodle, Autonomous learning, Automatic evaluation system

The use of online learning and management systems such as Moodle is constantly growing and it is used as the main source of communication with students, either to provide them material for study or to submit work and practices. This project consists in the recovery, maintenance, improvement and start-up of an automatic evaluation tool for programming exercises integrated in the Moodle platform: EvalCode. This tool allows the students to evaluate submissions and generate feedback data, both for students and teachers. The use of a system like this helps teachers to reduce the work associated with continuous assessment and, in turn, is able to provide students with an immediate idea of the deficiencies of their exercise. The work is based on a previous version of EvalCode that could not be complete and functional for its real use. The aim is to add functionalities, correct those that are not complete and perform a deployment of the system to do real tests with the students of Computer Engineering of the University of Cantabria.

Méritos

El presente trabajo forma parte de un proyecto de innovación docente del departamento de Ingeniería Informática y Tiempo Real de la Facultad de Ciencias de la Universidad de Cantabria llamado "Fomentando el aprendizaje autónomo del estudiante: sistema de evaluación automática de ejercicios de programación informática.". Este proyecto tenía como objetivo final disponer de una versión funcional de EvalCode para poder utilizarlo como parte de la evaluación en varias asignaturas del Grado en Ingeniería Informática.

Además, durante el desarrollo del proyecto, se presentó el *paper* "Extensión de Moodle para la evaluación automática de ejercicios de programación en Java" en colaboración con Héctor Pérez, Mario Aldea, Julio Medina y Rubén Sebrango. Esta publicación fue aceptada para el XXV congreso de las JENUI 2019 (Jornadas sobre la Enseñanza Universitaria de la Informática) explicando el proyecto EvalCode.

Índice

1 Introducción

1.1. Contexto

A lo largo del proceso de enseñanza-aprendizaje de un lenguaje de programación resulta imprescindible que el profesor lleve un control continuo del trabajo de cada alumno, y que el alumno vaya identificando progresivamente las deficiencias de sus soluciones. Estas deficiencias no deben estar solo enfocadas al resultado funcional del problema presentado, sino que debe incluir aspectos de calidad (legibilidad, complejidad, eficiencia, etc). La evaluación de todos estos aspectos constituye un proceso largo y complejo, especialmente en cursos introductorios donde el número de alumnos es generalmente alto en relación al profesorado disponible.

En este contexto resulta interesante disponer de un sistema capaz de evaluar el resultado funcional como la calidad del código de manera automatizada y coherente, proporcionando información al alumno sobre las posibles mejoras para completar su trabajo, y también al profesor para que obtenga más datos con los que completar la evaluación. Este tipo de planteamiento fomenta el aprendizaje autónomo del estudiante [1], que pasa a obtener información inmediata sobre las deficiencias detectadas en sus ejercicios, y, en consecuencia, favorece la supresión progresiva de malos hábitos en la programación, y facilita la obtención de una calificación aproximada según los criterios de la asignatura en cuestión. En el caso de los profesores, se reduciría en parte el trabajo asociado a las actividades de evaluación continuada.

Por otro lado, los Sistemas de Gestión de Aprendizaje (SGA) como Moodle facilitan el seguimiento del proceso de aprendizaje, así como el desarrollo de actividades de evaluación que, generalmente, permiten vincular calificaciones con comentarios de realimentación. Una definición más completa sobre Moodle y sus posibilidades se verá en el apartado 3.1, pero por ahora es suficiente señalar que Moodle es una plataforma web pensada tanto para docentes como para alumnos que facilita la estructuración y seguimiento de múltiples cursos o asignaturas. Esta plataforma está diseñada para ser fácilmente extensible y personalizable, lo cual brinda la posibilidad de crear extensiones con todas las funcionalidades requeridas.

1.2. Trabajo relacionado

En la actualidad, el uso de SGA está totalmente integrado en el modelo educativo de la mayoría de universidades y los docentes están familiarizados con su entorno y funcionamiento. Extender este tipo de plataformas con nuevas herramientas garantiza una adaptación rápida de estudiantes y profesores [2]. A continuación se va a introducir el *software* en el que se centra este TFG, denominado EvalCode, así como otros trabajos relacionados con la creación de herramientas de evaluación automática.

1.2.1. Versión preliminar de EvalCode

El *software* EvalCode parte de un Trabajo Fin de Grado anterior [3], desarrollado por Rubén Sebrango, en el que se creó un marco para la evaluación automática de código a través de la plataforma Moodle. En este trabajo se desarrolló un prototipo con una funcionalidad básica, pero carecía de un grado de desarrollo suficiente para un despliegue real, ya que muchas funcionalidades estaban en progreso y había presentes fallos a nivel de *frontend* y *backend*.

EvalCode se toma, antes de comenzar este trabajo, en versión de *julio-2017* y, tras el desarrollo que se verá en los siguientes capítulos, se llega a la considerada versión final 1.0.0.

1.2.2. Otros trabajos

El uso de herramientas de apoyo en el proceso de enseñanza-aprendizaje de los conceptos fundamentales de programación es un tema que ha generado gran interés [4]. Existen multitud de trabajos disponibles relacionados con la evaluación automática de ejercicios de programación [5], aunque solo nos vamos a centrar en aquellas herramientas que se encuentran integradas en la plataforma Moodle. Dentro de estas se encuentra la extensión JavaUnitTest [6] que permite la evaluación de pequeños fragmentos de código Java utilizando test JUnit. Un enfoque similar propone CodeRunner [7], aunque este último soporta distintos lenguajes de programación.

Para la evaluación de códigos más complejos existe la extensión VPL [8]. La funcionalidad principal de esta extensión es similar a la proporcionada por EvalCode, aunque VPL presenta además prestaciones adicionales como soporte para distintos lenguajes de programación o detección de plagios en las entregas realizadas. El cálculo de métricas más elaboradas mediante herramientas, como SonarQube [9] por ejemplo, está contemplado en el proyecto para asignaturas más avanzadas.

1.3. Objetivos

El objetivo principal de este trabajo reside en el desarrollo y configuración de un sistema de evaluación automática de ejercicios de programación informática, así como su integración en Moodle por ser una de las plataformas SGA más utilizadas en la actualidad. En líneas generales, el sistema propuesto debe ser capaz de detectar deficiencias y generar comentarios sobre los ejercicios de programación en cualquier lenguaje entregados por los alumnos, así como establecer una calificación en función de unos parámetros preestablecidos por el profesor. A continuación se muestra un desglose completo de los sub-objetivos del proyecto:

- Conseguir una versión funcional del *plug-in* capaz de realizar evaluaciones sencillas en Java, proporcionando una calificación sobre la entrega de una alumno y realimentación sobre los posibles fallos o aspectos a mejorar.
- Modularizar el código para poder evaluar prácticas en cualquier lenguaje y con cualquier herramienta que requiera el profesor.
- Redactar una documentación del *plug-in* para futuros desarrolladores así como manuales de uso para todos los usuarios del sistema.
- Desplegar una versión operativa en un servidor de explotación de la Universidad de Cantabria accesible por los alumnos.
- Realizar pruebas reales con estudiantes del Grado en Ingeniería Informática a fin de poder detectar posibles deficiencias en el sistema de autoevaluación. Este apartado es particularmente importante ya que esta extensión nunca ha sido probada en un entorno real, con varios alumnos realizando simultáneamente entregas de diferentes ejercicios de programación.
- Adaptar el módulo a las nuevas versiones de la plataforma. Moodle está en constante crecimiento y aunque, debido a la estructura de desarrollo las nuevas versiones de la plataforma son totalmente compatibles con los módulos desarrollados, es necesario llevar un mantenimiento continuado para adaptarse a los nuevos requisitos.
- Añadir nuevas funcionalidades a los procesos de evaluación en Java, como utilizar librerías externas o dar soporte a jerarquías de paquetes complejas.

Adicionalmente, se plantea también integrar en el sistema un generador de informes de evaluación para el profesor con un resumen de todas las entregas de un determinado trabajo, así como mejorar la visualización en la parte de *feedback* que se le proporciona a un alumno tras calificar una práctica.

El tiempo juega un papel importante también en el desarrollo del proyecto ya que se pretende que EvalCode forme parte de la evaluación de varias asignaturas del Grado en Ingeniería Informática de la Universidad de Cantabria para el curso 2019/2020.

2 Planificación y herramientas empleadas

En este punto se detalla la organización definida para el proyecto y una breve explicación de las diferentes herramientas y lenguajes que se han utilizado durante el desarrollo para una mejor comprensión cuando se haga uso de términos referentes a estas tecnologías.

2.1. Planificación

La planificación se estructuró en tres fases que se corresponden con una duración total de 8 meses de trabajo, cada una de las cuales consta de diferentes tareas, no necesariamente en el orden en el que aparecen dentro de su período de realización. La figura 1 representa el orden en el que se plantean las tareas, son de mayor o menor tamaño dependiendo de la estimación sobre el tiempo que llevará completar cada una.

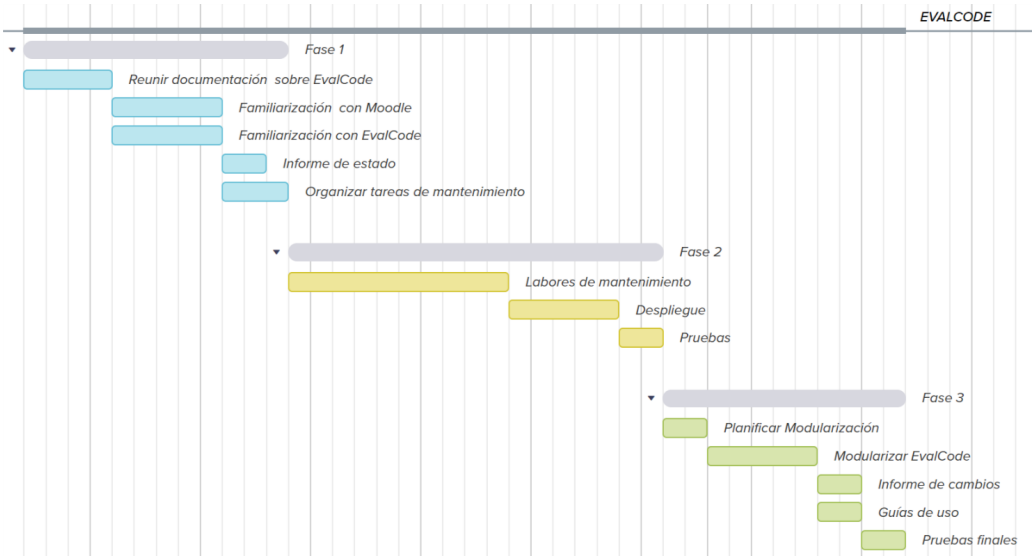


Figura 1: Diagrama de Gantt con la planificación del proyecto.

FASE 1

- Reunir toda la documentación y trabajo disponible sobre EvalCode.
- Familiarización con el funcionamiento interno de Moodle y su entorno web.
- Familiarización con el funcionamiento del *plug-in*.
- Realización un informe de estado y de errores.
- Priorización y organización las tareas de mantenimiento.

FASE 2

- Comienzo de las labores de mantenimiento y gestión de errores.
- Despliegue del *plug-in* en un servidor local al que puedan acceder los alumnos.
- Realización de pruebas reales con alumnos para poder detectar nuevos errores y volver elaborar un informe de errores y mejoras para repetir esta fase hasta un resultado de prueba óptimo.

FASE 3

- Partiendo de una versión estable y funcional de EvalCode para los viejos requisitos (funcionamiento correcto para Java con JUnit y Checkstyle), organizar la modularización del código para los nuevos (funcional para todos los lenguajes que se quiera).
- Modularización del sistema.
- Redacción de un informe de cambios.
- Creación de guías de uso para administrador, profesor y alumno.
- Pruebas finales.

Hay que tener en cuenta que, sobre esta planificación inicial, surgieron cambios a lo largo del desarrollo, debido a la aparición de nuevas tareas, como por ejemplo la redacción de un *paper* para enviar al congreso JENUI explicando el proyecto EvalCode, y también debido a que algunas de las tareas llevaron más tiempo del esperado, principalmente las pertenecientes a la FASE 1 por estar EvalCode en un nivel inferior al esperado.

2.2. Lenguajes

- PHP
  PHP (Hypertext Preprocessor) [10] es un lenguaje de programación libre y conocido con especial uso en desarrollo web por su facilidad para ser embebido en HTML (HyperText Markup Language). En concreto se hace uso de la versión 5 para este trabajo por ser la recomendada para este tipo de desarrollos.
- SQL
  El *Structured Query Language* (SQL) es un lenguaje declarativo estándar internacional dentro de las Bases de Datos (BD) que permite el acceso y manipulación de los datos [11]. Gracias a la posibilidad de integrar este lenguaje dentro de PHP, Moodle hace uso de él para gestionar usuarios, cursos, entregas, etc. Y, como se verá más adelante, cada extensión tiene su propia base de datos para almacenar entregas u otra información valiosa.
- Java
  Java es la base para prácticamente todos los tipos de aplicaciones de red, además del estándar global para desarrollar y distribuir aplicaciones móviles y embebidas, juegos, contenido basado en web y software de empresa [12]. Se ha utilizado como el primer lenguaje sobre el que probar EvalCode por su amplio uso y versatilidad.
- XML
  XML (*Extensible Markup Language*) [13] es un lenguaje de marcado similar a HTML (*Hypertext Markup Language*). El propósito principal del lenguaje es compartir datos a través de diferentes sistemas, como Internet. Es imprescindible ya que Moodle utiliza XML para definir la base de datos de cada extensión, y también resulta muy útil para almacenar información en la misma.

- Markdown
  Markdown es una herramienta de conversión de texto plano a HTML. Se considera un lenguaje que tiene la finalidad de permitir crear contenido de una manera sencilla de escribir, y que en todo momento mantenga un diseño legible [14].

2.3. Herramientas y tecnologías

- Moodle
  Moodle (*Modular Object-Oriented Dynamic Learning Enviroment*) es una plataforma de aprendizaje diseñada para proporcionarle a educadores, administradores y estudiantes un sistema integrado único, robusto y seguro para crear ambientes de aprendizaje personalizados [15]. Una definición más completa se detalla en el apartado 3.1.

- Visual Studio Code
  Es un editor de código fuente ligero pero potente que se ejecuta en modo escritorio y está disponible para Windows, macOS y Linux [16]. Dispone de extensiones para trabajar con todo tipo de lenguajes y entornos y ha sido el principal entorno de desarrollo para todo el proyecto.

- FileZilla
  Es un cliente de FTP (*File Transfer Protocol*) y SFTP (*Secure* FTP) multiplataforma rápido y confiable con muchas características útiles para transferir archivos entre máquinas y una interfaz gráfica de usuario intuitiva [17].

- PuTTy
  Cliente SSH (*Secure Shell*) y telnet, desarrollado originalmente por Simon Tatham para la plataforma Windows. PuTTY es un software de código abierto que está disponible con código fuente y está desarrollado y respaldado por un grupo de voluntarios [18].

- VirtualBox
  Software de virtualización completo de uso general para hardware x86, dirigido a servidores, y al uso de sistemas integrados [19]. Esta herramienta ha sido clave para poder crear el módulo de Moodle puesto que, gracias a ella, desde cualquier equipo se puede configurar una maquina con Moodle instalado para poder utilizarlo.

- Meld
  Es una herramienta que permite realizar un *diff* sobre dos archivos o directorios enteros. Meld [20] es útil para visualizar los cambios introducidos o comparar dos versiones de un mismo fichero. Se ha hecho uso del programa para comparar entregas, diferentes versiones del *plug-in* y directorios de trabajo.

- GitHub
  Se trata de una de las plataformas más usadas para el controlador de versiones Git. GitHub [21] permite hacer varias versiones de un repositorio para poder recular si nos hemos equivocado y nuestra aplicación ya no funciona, o para trabajar en funcionalidades nuevas sin necesidad de modificar la versión funcional. Ver apartado 4.3 para más información sobre el repositorio en el que se encuentra alojado el proyecto.

- Unix Shell
  La *shell* de Unix es, a la vez, un intérprete de comandos y un lenguaje de programación. Como intérprete de comandos, la *shell* proporciona la interfaz de usuario al amplio conjunto de utilidades de GNU.

- CheckStyle
  Checkstyle [22] es una herramienta de análisis de código utilizada en el desarrollo de software

para verificar si el código de Java cumple con las reglas de estilo que se especifique mediante un documento de reglas en XML.

- JUnit
  JUnit [23] es un *frameword* Java que permite ejecutar clases de manera controlada para poder comprobar que los métodos realizan su cometido de forma correcta.

3 Moodle

En este apartado se va a definir qué es Moodle de forma genérica, los aspectos que hay que tener en cuenta y los pasos a seguir para desarrollar una extensión. Todos estos conceptos son necesarios para comprender el desarrollo realizado sobre EvalCode y su funcionamiento interno.

3.1. Vision general

Moodle, es una plataforma *online* de aprendizaje altamente personalizable y de código abierto (licencia GNU GPL [24]) lanzada por primera vez en 2001. Esto supone que un desarrollador puede acceder directamente al código de la aplicación sin pagar por su descarga o por utilizarlo.

La infraestructura utilizada se muestra en la figura 2 y se basa en una combinación de tecnologías llamada LAMP: **Linux**, para el sistema operativo; **Apache**, para configurar el servidor HTTP web y los protocolos de red; **MySQL**, utilizado en la gestión de las bases de datos junto con PostgreSQL; y **PHP** como principal lenguaje de programación en todo el entorno. Esta disposición está muy extendida para entornos de aplicaciones web. En el centro de todas estas tecnologías es donde se encuentra Moodle.

Figura 2: Diagrama de arquitectura LAMP de Moodle.

El impacto de esta herramienta educacional es cada vez mayor, en parte por ser fácilmente escalable y configurable, y también por ofrecer altas prestaciones en seguridad. Se utiliza en toda clase de instituciones de enseñanza, como institutos y universidades. La figura 3 es un ejemplo de *front page* de un sitio Moodle y muestra la página a la que se accede desde un navegador después de haber iniciado sesión. En ella aparecen los cursos a los que pertenece un usuario, bloques de información, próximos eventos, tareas pendientes, etc.

La estructura principal de la plataforma se basa en cursos. Se trata esencialmente de páginas privadas gestionadas por profesores y contienen recursos de aprendizaje y actividades para los estudiantes.

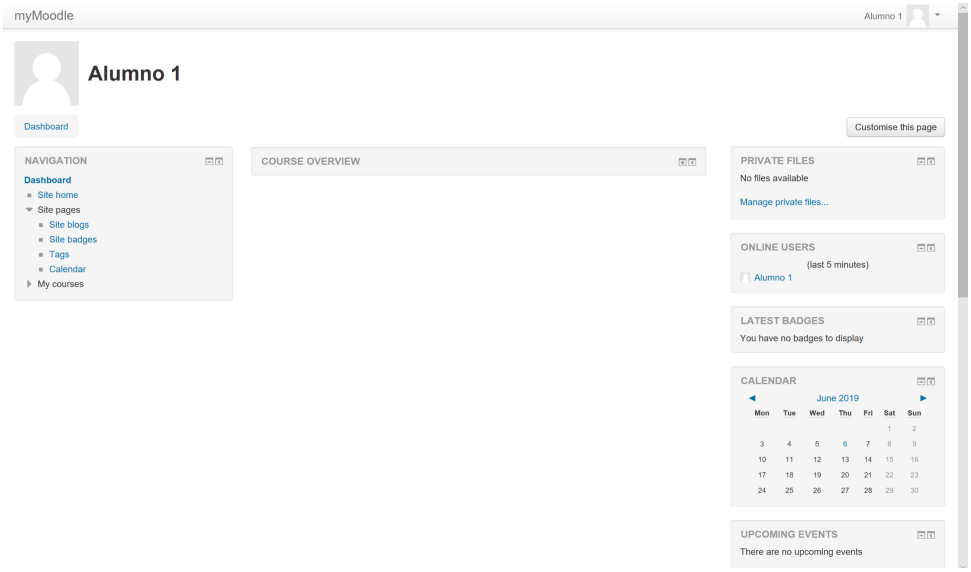


Figura 3: Ejemplo de una *front page* de Moodle.

Hay muchos tipos diferentes de actividades y de bloques que se pueden añadir a un curso, y si no se encuentra una herramienta que cumpla nuestras necesidades, podemos desarrollar la nuestra propia.

Moodle tiene una comunidad de desarrolladores muy activa que contribuye constantemente a mejorar sus prestaciones. Lo esencial para crear cualquier nueva funcionalidad en Moodle es entender su API (*Application Programming Interfaces*) de desarrollo basada en una filosofía orientada a objetos, en la que todo el sistema consiste en cajas negras interconectadas de las que no se conoce su comportamiento interno, pero sí su entrada, salida, estado particular en el que se encuentra en un determinado momento y su identificador único.

Por otra parte, Moodle presenta un tamaño interno considerable, con una gran cantidad de servicios que dependen unos de otros. Por ello, puede resultar abrumador para un desarrollador primerizo adentrarse dentro del código. Parte de ese tamaño se debe al considerable número de extensiones que posee. Las extensiones son piezas adicionales de *software* que añaden funcionalidades al propio Moodle. A continuación se introducen cada una de las capas de la arquitectura de Moodle y dónde se sitúan las extensiones en este contexto y cuál es su papel.

3.2. Extensiones tipo *Activity module*

Como el propio nombre de Moodle indica, nos encontramos ante una plataforma modular. Esto quiere decir que está preparada para añadir nuevas funcionalidades de forma sencilla. Si estamos ante un cliente que desea una funcionalidad concreta que no viene integrada por defecto en el sistema, la solución es crear un nuevo módulo o extensión para implementarla.

Una extensión (también llamada módulo o *plug-in* en inglés), propiamente dicha, es una aplicación que, en un programa o sistema informático, añade una funcionalidad o nueva característica al *software*. Si hablamos en concreto de las que se crean para Moodle, existen varios tipos, como por ejemplo: *Activity modules*, los más esenciales ya que proveen la creación de nuevas actividades en cursos; *Antivirus plugins*, capaces de escanear ficheros subidos por un usuario en busca de virus; *File Converters*, permiten convertir ficheros a diferentes formatos; *Machine learning backends*, para predicción y análisis de datos, etc. Este proyecto se va a centrar en los *Activity modules* ya que es el tipo más aproximado a las funcionalidades de EvalCode.

Tal y como se ilustra en la figura 4, cada nuevo módulo de este tipo que se desee incluir en la plataforma necesita tres cosas principalmente: los ***scripts*** que definen el comportamiento como tal, uno o varios **formularios** y una **base de datos** propia para almacenar toda la información referente

a los dos anteriores.

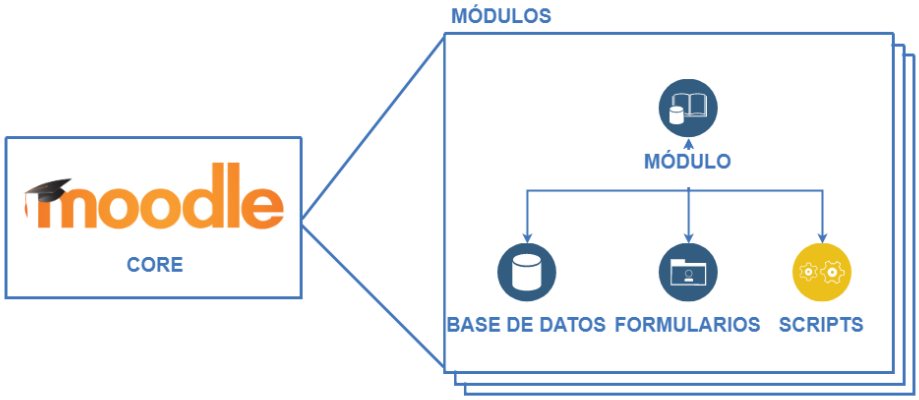


Figura 4: Módulos de Moodle.

Todos los módulos de este tipo se sitúan en subdirectorios dentro de `mod/` (en los archivos de Moodle). A continuación se detallan cada uno de los aspectos requeridos por un módulo para funcionar.

3.2.1. *Scripts*

Se entiende por *scripts* todos los archivos que contengan, en este caso, código PHP y que están en comunicación con los propios de Moodle, por ejemplo, algunos de estos ficheros se utilizan para instalar el módulo e integrarlo con el resto del sistema. Es en estos *scripts* en los que se programa la lógica de una extensión, cada uno tendrá una función concreta y se pueden añadir tantos como se quiera para la tarea que se le quiera encomendar. Sin embargo, hay algunos que son críticos para que todo pueda funcionar:

- mod/
  - db/
    - access.php
      Lugar donde se define cuáles son las capacidades del *plugin*. Las capacidades se cargan en la tabla de la base de datos cuando el módulo se instala o actualiza. Cada vez que se actualicen las definiciones de capacidades, el número de versión del módulo debe ser aumentado.
    - install.xml
      Este fichero XML es el que utiliza Moodle durante la instalación del módulo para crear tablas y campos en la base de datos principal. Es importante tener en cuenta que modificar este fichero a través de XMLDB_editor no modificará la BD hasta que no se actualice o reinstale el módulo. Es decir, el install.xml solo se ejecuta una vez, cuando el módulo se instala por primera vez.
    - upgrade.php
      Este archivo maneja la actualización del módulo para que coincida con la última versión ya que se han podido introducir cambios de funcionalidad que afecten directamente al resto. Es importante para mantener una organización sobre las dependencias de una extensión.
  - lang/
    Aquí es donde se almacenan las cadenas de texto que va a utilizar el plug-in. Cada idioma tiene una carpeta específica, por ejemplo “en”(English). Hay un marcador de posición obligatorio para los *plug-ins* llamado `<nombre_del_modulo>` que Moodle usará cuando incluya

este módulo como una opción para agregar a un curso y otras páginas. Para cumplir con los estándares de Moodle, las cadenas deben estar ordenadas alfabéticamente por el nombre del marcador de posición.

Es muy importante mantener actualizados los ficheros de texto y evitar incluir cadenas directamente en el código del *plug-in*. En su lugar hay que acostumbrarse a utilizar instancias del *array* aquí definido, como por ejemplo las que se muestran en la figura 5.

```
$string['addsubmission'] = 'Add submission';
$string['addattempt'] = 'Allow another attempt';
$string['addnewattempt'] = 'Add a new attempt';
```

Figura 5: Extracto de código para definir cadenas de texto.

- index.php
  Moodle utiliza esta página al enumerar todas las instancias del módulo que se encuentran en un curso en particular. Se entiende por instancia a una nueva tarea o actividad de tipo EvalCode que se crea dentro de un curso. Se obtienen a través de una petición SQL realizada con un identificador único. Dependiendo de cómo se haya configurado el módulo durante su instalación, mostrará más o menos información referente a cada una de esas actividades.
- lib.php
  Contiene tres funciones principales para controlar la creación, modificación y borrado de instancias del módulo en un curso, sus cabeceras son las listadas en la figura 6.

```
function <nombre_del_modulo>_add_instance($certificate);
function <nombre_del_modulo>_update_instance($certificate);
function <nombre_del_modulo>_delete_instance($id);
```

Figura 6: Cabeceras de las funciones principales de lib.php.

  - *add_instance*: recibe las variables del archivo `mod_form.php` (explicado más adelante) como un objeto cuando se crea una actividad por primera vez. Esta función toma los datos especificados para luego hacer lo que se quiera con ellos e insertarlos en la base de datos. Hay que tener en cuenta que solo se llama una vez.
  - *update_instance*: realiza operaciones similares a la función anterior pero cada vez que se actualiza la actividad, es decir, opera de nuevo con los datos introducidos y los vuelve a guardar. La principal diferencia es que necesita el ID de la instancia que se está editando (pasada como parámetro) y lo usa para editar cualquier valor existente en la base de datos para esa instancia.
  - *delete_instance*: se utiliza para borrar una instancia del módulo concreta especificando su ID. Se puede configurar un tratamiento distinto para cada valor almacenado.
- mod_form.php
  Este archivo se utiliza al agregar o editar un módulo a un curso. Contiene los elementos que se mostrarán en el formulario responsable de crear e instalar una instancia del módulo. La clase en el archivo debe llamarse mod_<nombre_del_modulo>_mod_form. La creación y uso de estos formularios se detalla en la siguiente sección (3.2.2).
- review.php
  Es una página específica para mostrar una revisión de un intento de entrega en particular. Puede ser utilizado por el alumno para comprobar sus intentos o por el profesor para realizar revisiones sobre la actividad.
- version.php
  Este es un archivo esencial para el módulo, y debe estar situado siempre en el directorio raíz.

Contiene una serie de propiedades que se utilizan durante la instalación o actualización del módulo, como por ejemplo asegurarse de que es compatible con la versión del sitio Moodle o detectar que necesita una actualización.

- view.php
  Cuando un curso renderiza la página de un curso y sus actividades, `view.php` genera los enlaces para verlos.

3.2.2. *Formularios*

Las nuevas extensiones de Moodle suelen necesitar cambiar los formularios de creación de nuevas actividades que vienen por defecto o crear otros nuevos. Estos *Web Forms* se crean usando elementos de la *Form API*, la cual soporta multitud de elementos HTML (*checkbox*, *radio*, *textbox*,etc.), pero modificados para proporcionar una mayor accesibilidad y seguridad, así como el estilo propio de Moodle. Para crear uno nuevo hace falta añadir una nueva clase que herede de `moodleform`, y para eso es necesario importar `/formslib.php`. Una vez hecho esto hacen falta dos fases: la **construcción del *formulario*** y la **validación de los datos**.

Para la fase de construcción del formulario se debe definir la función `definition()`, en ella simplemente se obtiene el objeto $mform y, sobre este objeto, se añaden todos los nuevos elementos que se quiera. Hay que tener en cuenta que cada uno de estos elementos necesitará un campo correspondiente en la base de datos para almacenar la información. Será llamada en el momento en el que un profesor o administrador pulse la opción "Añadir" con el módulo deseado seleccionado. Un extracto del código PHP con la forma de esta función se puede ver en la figura 7.

```
require_once("$CFG->libdir/formslib.php");
class simplehtml_form extends moodleform {
    function definition() {
        global $CFG;
        $mform = $this->_form; // No olvidar la _
        $mform->addElement()... // Añade elementos al form
            ...
    }
}
```

Figura 7: Extracto de código del método *definition*.

Después, para la fase de validación de los datos, hay dos formas disponibles para comprobar que la información facilitada sigue las normas deseadas, una consiste en utilizar `MoodleQuickForm::addRule()` (normalmente dentro del método de definición) para especificar la regla asignada a un campo según se ha creado [25] (como se muestra en la figura 8).

```
$mform->addElement('text','shortname', get_string('shortname'),'maxlength="15" size
    ="10"');
$mform->setHelpButton('shortname', array('courseshortname', get_string('shortname')),
    true);
$mform->setDefault('shortname', get_string('defaultcourseshortname'));
$mform->addRule('shortname', null, 'required', null, 'client');
$mform->setType('shortname', PARAM_MULTILANG);
```

Figura 8: Extracto de código de la función *definition* con reglas de validación.

Y otra forma es definir la función `validation()` (ver figura 9) a la que se le pasan dos parámetros: $data, *array* con los datos añadidos a cada uno de los campos del *form*; y $files, con los archivos adjuntados durante el proceso de creación de la actividad. A diferencia de la forma anterior, esta permite personalizar la validación en lugar de utilizar reglas predefinidas. Esta función será la encargada de asegurarse de que todos los campos del formulario cumplen con los requisitos de formato y que

están rellenados todos los que sean obligatorios; y otros especiales, como por ejemplo que una serie de campos numéricos sumen un valor concreto. Cuando no se cumple alguno de los requisitos o se detecta un error se suele añadir a un *array* de errores que posteriormente se retornará. Si no hay ningún error, se retorna el valor *null*. También hay que tener en cuenta que, si se decide hacer un *override* de este método desde la clase padre, será necesario hacer una llamada al `validate()` de `moodleform` para evitar fallos. Si se opta por esta opción, el método se llamará cuando se pulse una de las dos opciones de "Guardado".

```
class gorgeous_form extends moodleform {
    ...
    // Validacion personalizada
    function validation($data, $files) {
        $errors = parent::validation($data, $files);
        ...
    }
}
```

Figura 9: Extracto de código de la función *validation*

Los formularios de Moodle pueden extenderse de muchas maneras: sería posible añadir tus propios elementos de JavaScript con un estilo CSS propio, o incluirlos dentro de otros bloques que no tienen por que ser el formulario de creación de una actividad. Éste es un ejemplo de la página resultante del formulario del módulo *Assignment* (figura 10).

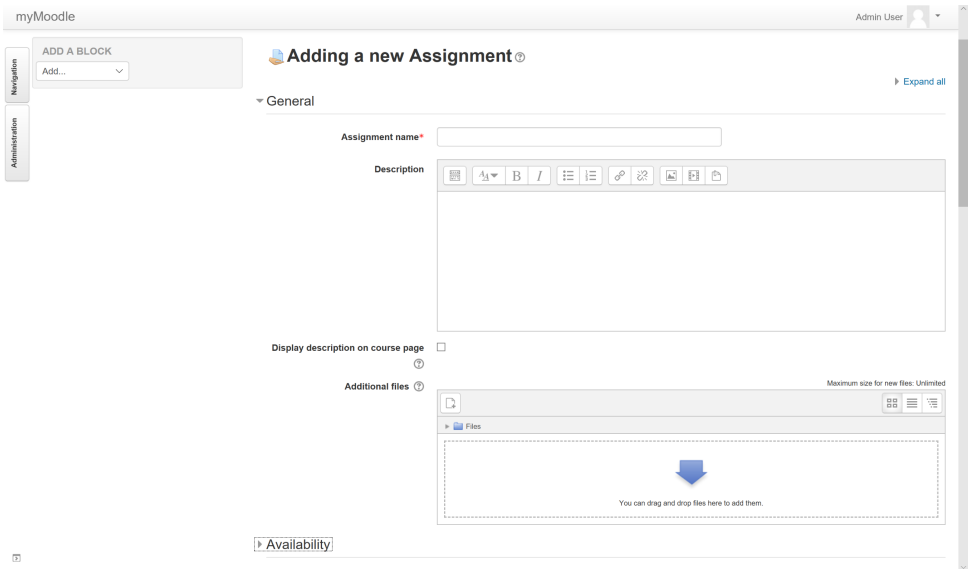


Figura 10: Ejemplo de formulario para el módulo *Assignment*.

Como hemos visto, cada uno de los campos se corresponde con un elemento creado en `definition()`. Después se comprobará que campos obligatorios como el nombre en este caso se hayan rellenado. Hasta que el proceso de validación de todos los campos no se haya completado correctamente, no se guardará la nueva instancia del módulo en la base de datos ni se mostrará en el curso. A esta vista solo pueden acceder los profesores designados o un administrador.

3.2.3. Base de datos

Moodle posee su propia base de datos para manejar usuarios y ficheros, pero cada extensión que se añada generará una nueva "ramificación" de la base principal. Esto se conoce como *database activity*. La API de manipulación de datos utiliza métodos públicos accesibles a través del objeto $DB, este

objeto global es una instancia de la clase `moodle_database` y se inicializa automáticamente durante la fase de configuración en el arranque. El núcleo de Moodle se encarga de configurar la conexión con la base de datos de acuerdo a los valores que se hayan especificado en el fichero principal `config.php`.

Como se ha mencionado en el apartado 3.2.1 la base de datos se define a partir del XML `install.xml` y puede editarse mediante la herramienta integrada en Moodle *XMLDB editor*. Este editor permite hacer todas las modificaciones que queramos manteniendo siempre los patrones estipulados por Moodle, por lo que es mucho más aconsejable usarlo en lugar de editar directamente sobre el XML. Para un módulo cualquiera aparecen varias ramificaciones, la figura 11 corresponde a una vista de dichas ramificaciones para el módulo *Assignment*.

Figura 11: Bases de datos existentes del módulo *Assignment*.

Si cargamos la principal, observaremos una vista de cada una de las tablas SQL para modificar sus campos (figura 12).

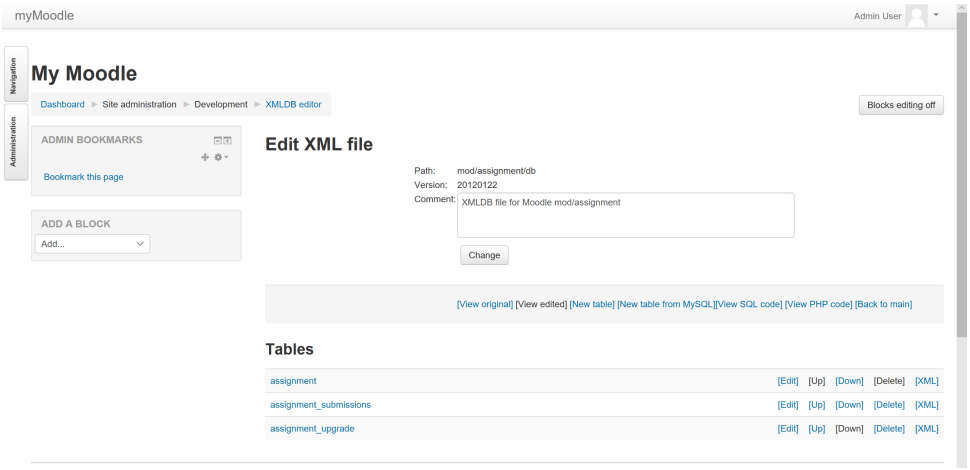


Figura 12: Vista de las tablas disponibles en *Assignment*.

XMLDB editor en ningún momento sirve para mostrar la base de datos en su estado actual, hacer consultas sobre tablas o editar celdas concretas. Para eso tenemos que acceder a través de un terminal o con un cliente visual específico con los credenciales adecuados. Desde el propio código se hacen llamadas a partir del objeto `\$DB`.

4 EvalCode

El sistema de evaluación automática EvalCode se ha desarrollado como una extensión que se integra como un nuevo módulo en Moodle, junto con el resto de recursos y actividades proporcionados por la plataforma. En la figura 13 puede observarse la relación entre la extensión y las distintas tecnologías sobre las que se apoya el servidor Moodle, LAMP (Linux, Apache, MySQL y PHP). Según la figura, los usuarios de la plataforma acceden a través de la red al servidor Apache en el que se encuentra alojado Moodle. La comunicación entre éste y los módulos disponibles se hace mediante clases y funciones PHP que, a su vez, ejecutan sentencias MySQL para realizar consultas e inserciones en la base de datos, y rutinas de Bash para la gestión de los archivos en el sistema. El proceso de análisis y evaluación automática se sirve de los módulos existentes para la interconexión con el resto de componentes de Moodle, detallados anteriormente.

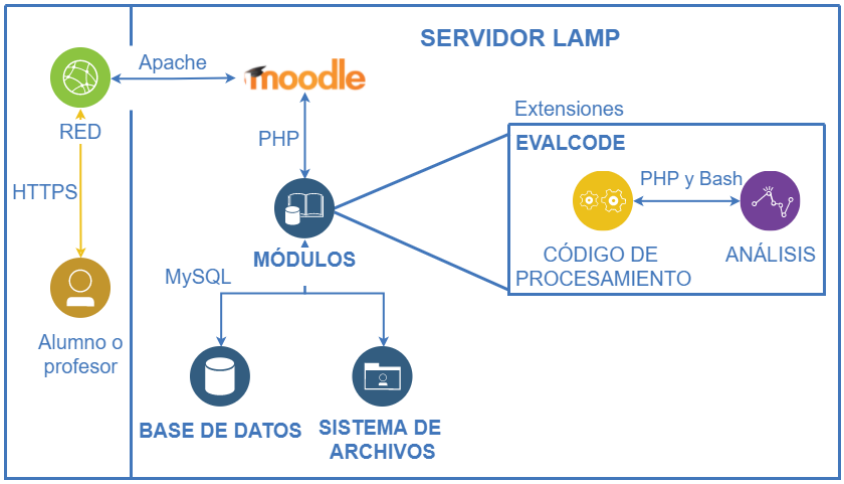


Figura 13: Diagrama de arquitectura de EvalCode.

Adentrándonos ahora en el desarrollo propiamente dicho del proyecto, se va a explicar la estructura interna de la que parte la extensión, el comportamiento que debería tener en base al proyecto inicial, las labores de mantenimiento realizadas y las mejoras implementadas durante el proceso de modularización.

4.1. Estructura del *plug-in*

EvalCode posee todos los ficheros necesarios para cualquier módulo vistos en el apartado 3.2, pero necesita nuevos *scripts* que se encarguen de las funcionalidades propias. En la figura 14 se pueden ver los detalles internos del *plug-in* original en el momento de la toma de contacto sin haber realizado aún modificaciones.

La estructura del módulo se basa en un diseño de tres capas [3]: capa de **datos**, en la que se encuentra el servidor de la base de datos; capa **lógica**, con todos los *scripts* que interactúan con el software de EvalCode y la plataforma Moodle; y la capa de **presentación**, correspondiente el desarrollo de formularios visto en el apartado 3.2.2.

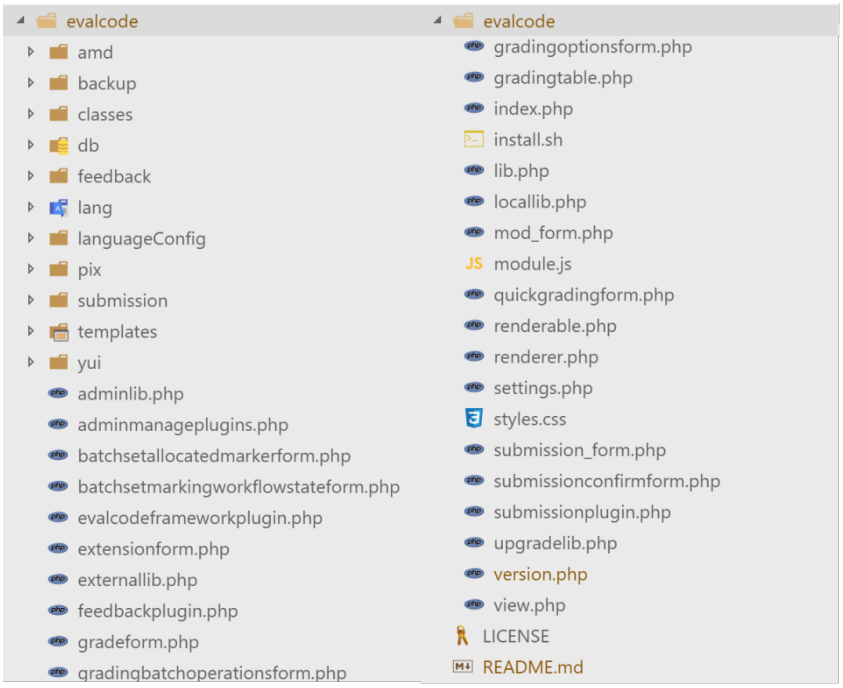


Figura 14: Jerarquía completa de ficheros (en dos columnas) de EvalCode *versión julio-2019*.

A la capa de datos pertenecen ficheros de la figura 14 como `install.xml`, en el que se almacena la estructura de tablas necesarias para el módulo en formato XML. La mayoría de *scripts* como `adminlib.php`, `lib.php` o `settings.php` pertenecen a la capa lógica de EvalCode que compone todo el comportamiento propiamente dicho del software, pero sin duda la mayor parte de la lógica del *plug-in* y sus funcionalidades se encuentran en `locallib.php`: funciones de almacenamiento de ficheros en la base de datos, proceso de evaluación, métodos de calificación y generación de comentarios de realimentación, etc.

Por último, en la capa de presentación se encuentran los formularios, descritos en `mod_form.php`, y aquellos ficheros necesarios para definir la visualización en Moodle del *plug-in*, como ocurre en `view.php`.

4.1.1. Comportamiento del *plug-in*

En su versión inicial, EvalCode solo fue pensado para evaluar prácticas de programación en Java mediante JUnit y CheckStyle. Para poder trasladar esta implementación a un sistema genérico capaz de evaluar cualquier tipo de lenguaje y utilizando una mayor diversidad de herramientas, son necesarios muchos cambios sustanciales. Todos los cambios se detallan en el apartado 4.2.1.

Pero antes conviene explicar aquellos conceptos necesarios sobre el comportamiento que sigue la versión inicial de EvalCode, para poder así comprender mejor todas las modificaciones que se van a aplicar. Cuando un profesor crea una nueva actividad, rellena los campos del formulario (nombre, descripción, plazos...) y sube dos ficheros comprimidos (.zip); uno con el material que se proporciona al alumno para realizar la práctica, y otro con los archivos de test de JUnit que se utilizarán durante el proceso de evaluación. Cuando el alumno tiene lista la entrega de su trabajo, debe subir un comprimido con el contenido del directorio `src/` del proyecto de Eclipse [26] correspondiente.

Una vez hecho esto se inicia el proceso de evaluación, que consiste en:

- Descargar en un directorio temporal la entrega del alumno
- Descomprimir la entrega.

- Descargar los ficheros de evaluación proporcionados por el profesor y sustituirlos, en caso de que haya documentos iguales en la entrega, por los del alumno.

Esto último puede ocurrir, ya que se puede haber facilitado con el enunciado de la práctica una o varias clases de prueba JUnit para que el alumno vaya probando la implementación que está desarrollando, pero las pruebas con las que quiere el profesor que se evalúe son más completas, por lo que deben ser reemplazadas.

A continuación se debe iniciar el proceso de compilación y, si ha finalizado correctamente, de ejecución de la herramienta CheckStyle, que comprueba si se han seguido las guías de estilo de programación estipuladas para la asignatura; y JUnit, para revisar las funcionalidades concretas del software.

Con toda la información generada por las herramientas de evaluación, se procede a calcular la nota del alumno y generar un comentario de realimentación que acompañará a la calificación, explicando en mayor detalle los fallos que se han cometido y un desglose del cálculo anterior.

4.2. Desarrollo

Tras conocer el comportamiento que debe tener el módulo, se procede a evaluar el estado actual del mismo realizando una primera instalación en una máquina virtual de pruebas (para más información sobre el despliegue ver apartado 6.1) y simular entregas de prácticas para ver las posibles deficiencias (fase 2 del proyecto). Por un lado se va a hablar del mantenimiento y actualización del *plug-in*, en este apartado se verán los errores encontrados y las soluciones aplicadas en cada caso; y por otro lado se explicará la modularización del *plug-in* en la que se pretende aumentar sus capacidades y características (fase 3).

4.2.1. Mantenimiento y actualización del *plug-in*

En este punto se da paso a las labores de mantenimiento, entendiendo por esto: detectar las deficiencias existentes, subsanarlas, dejar constancia del problema y la solución desarrollada y cerciorarse de que no se ha afectado al correcto funcionamiento del resto de aspectos.

Existen cuatro tipos de mantenimiento [27]: correctivo, adaptativo, perfectivo y preventivo. El mantenimiento **correctivo** tiene por objetivo localizar y eliminar los posibles defectos de los programas que ocurren cuando el comportamiento de un programa es diferente del establecido en la especificación. El **adaptativo** consiste en la modificación de un programa debido a cambios en el entorno (*hardware* o *software*) en el que se ejecuta. El **perfectivo** son cambios en la especificación, normalmente debidos a cambios en los requisitos de un *software*, que llevan a la incorporación de nuevas funcionalidades. Por último, el mantenimiento **preventivo** consiste en la modificación del *software* para mejorar sus propiedades, por ejemplo, incluir comprobaciones de validez de los datos de entrada del programa.

La siguiente lista detalla la explicación extensa de todos los errores y problemas encontrados en EvalCode junto con la solución que se ha aplicado. En cada caso se especifica la función, clase y fichero en el que se han realizado las modificaciones pertinentes y, en los casos en los que corresponde, la documentación a la que se ha recurrido para hallar la respuesta al problema. Al final de cada solución aplicada se indican las refactorizaciones correspondientes siguiendo el catálogo de refactorizaciones disponible en [28]. Los errores se han clasificado en: errores de **instalación**, aquellos errores que puedan aparecer al instalar Moodle o EvalCode; **configuración**; **ejecución**; **evaluación**, durante el proceso de evaluar una entrega; y **migración** de Moodle, conflictos que puedan surgir entre diferentes versiones de Moodle.

Errores de instalación

- Después de instalar EvalCode y sin haber modificado aun el código, no se pueden visualizar los mensajes de error al intentar hacer una ejecución de EvalCode.

  Para solucionar este defecto fue necesario cambiar el modo de funcionamiento de Moodle de *Normal* a modo *Developer* y los permisos de escritura sobre los ficheros de *log*. Esto hace que cualquier error que surja de un *script* PHP se vea directamente en el navegador si es grave o en el *log* si es un error capturado de forma normal. Los errores se envían al fichero `/tmp/evalcode_error.log`. [1]

  Refactorizaciones: Cambiado modo de operación.

Errores de configuración

- No se pueden borrar actividades EvalCode una vez creadas dentro de un curso.

  En primer lugar, había clases PHP copiadas del módulo *assignement* que no habían sido adaptadas. Después de introducir las modificaciones necesarias, se detectó que las llamadas de borrar una actividad se dirigían hasta `evalcode_delete_instance(...)` dentro de `locallib.php` donde se originaba un error de SQL debido a que el comando de borrado de la base de datos utilizaba un campo que no existe cuando se crea (id_activ), tras ser sustituido por el correcto (id), el sistema permite borrar correctamente actividades EvalCode pero no lanza ningún mensaje de confirmación antes. Para activar esta opción era necesario configurar el elemento *Recycle Bin*[2] (lugar donde se almacenan los datos temporalmente cuando son eliminados) de Moodle para que funcione con el *plug-in* y que esta opción se haga por defecto para futuras instalaciones.

  Refactorizaciones: Cambiada declaración de funciones, variables renombradas y reemplazados literales.

- El profesor puede editar correctamente todos los apartados de una actividad una vez ha sido creada, pero si modifica los ficheros de test, las nuevas evaluaciones no se ejecutan con los nuevos sino con los viejos.

  La función `evalcode_update_instance()` definida en `lib.php` y utilizada en `locallib.php` no actualiza los valores en la base de datos, sino que contiene una copia del código de creación de una nueva actividad. Fue necesario cambiar las sentencias SQL para modificar los valores de una instancia existente en la BD [3] en lugar de intentar crear otra con los nuevos parámetros especificados en el formulario.

  Refactorizaciones: Reemplazados parámetros de *query*.

Errores de ejecución

- Error cuando un alumno intenta hacer una entrega. Se interrumpe el proceso de evaluación y Moodle abre una ventana en blanco sin información.

  Cuando se descarga la entrega de un alumno en el directorio temporal se debería combinar con un archivo llamado `TestRunner.java` que se convertirá en el *main* después de compilar, su única función es la de llamar a la clase JUnit que contiene los test de prueba. El problema es que esta

[1] Modos de funcionamiento de Moodle: https://docs.moodle.org/dev/Developer_Mode
[2] `Administration>Plugins>AdminTools>RecycleBin`
[3] https://www.w3schools.com/sql/sql_update.asp

clase no se añade al directorio y, por tanto, no se encuentra para el proceso de compilación. Se añadió un comando para mover este fichero al directorio de trabajo.

Después se vio que la ejecución fallaba porque EvalCode buscaba unas etiquetas dentro de `TestRunner` para "sustituirlas" (más adelante se verá que esta sustitución no era correcta) por los nombres del paquete del proyecto Eclipse y por el nombre de la clase JUnit que corresponde, pero estas etiquetas no existían. Se añadieron y ahora la ejecución se completa sin más errores y, tras completarse la ejecución, se retorna al usuario a la página en la que se visualiza su entrega.

Refactorizaciones: Literales reemplazados por variables.

- No se puede volver a intentar realizar una entrega después de haberlo hecho una vez.

  Este problema se originaba desde varios puntos. Por un lado, no se llamaba a las funciones de Moodle que finalizan el proceso de calificación, por lo que la nota y el comentario de *feedback* se muestran pero el hilo que la gestiona no ha terminado y, por tanto, no se puede volver a arrancar. Tras incluir las funciones necesarias para finalizar el proceso [4], saltó a la vista que el directorio temporal en el que se descarga la entrega y se hace toda la evaluación, se construía con el nombre del zip de la entrega, esto generaba un error porque "el directorio ya existía". Se modificó este comportamiento para que los nombres de los directorios temporales sobre los que trabaja EvalCode sean una composición del id asignado al alumno que hace la entrega y una traza temporal de día, hora, minutos y segundos.

  Refactorizaciones: Reemplazada construcción de variable.

- Error al intentar realizar entregas que dependen de librerías externas, por ejemplo, el paquete fundamentos.

  En la fase de compilación de una entrega no se ha tenido en cuenta que pueden existir dependencias con librerías externas. Se creó un directorio en la máquina virtual en el que el profesor o administrador puede incluir todas las librerías que se vayan a usar en un curso.

  Refactorizaciones: Añadida variable de ruta.

- Tras la primera prueba real con alumnos se ha detectado que la calificación no se suma correctamente y que aparecen errores inesperados de compilación solo en algunos casos sin motivo aparente.

  Los errores durante la compilación se deben a que no se compilaba con la directiva `UTF-8`, todas las palabras que contengan tildes o eñes generarán un error al compilar (aunque estén dentro de un comentario). El número de errores aumenta debido a la conversión de ficheros entre sistemas operativos si el alumno ha realizado la práctica en un sistema distinto de Linux. De nuevo, se arregla añadiendo la misma directiva[5] al comando usado para descomprimir las entregas.

  Refactorizaciones: Reescrito comando.

- La herramienta de estilo CheckStyle comprueba ficheros que no pertenecen a la práctica entregada por el alumno.

  La función de comprobación de estilo que se llama durante la evaluación de una práctica pasaba por todos los ficheros del directorio temporal en el que se está trabajando. Se añadieron *scripts*[6] para pasar la herramienta solo por aquellos ficheros Java que pertenezcan a la entrega del alumno.

[4] Gestión de *threads* en PHP https://www.aurigait.com/blog/php-threading/

[5] Codificaciones con las que se puede compilar en Java https://docs.oracle.com/javase/7/docs/technotes/tools/windows/javac.html

[6] Cómo configurar CheckStyle: http://checkstyle.sourceforge.net/config.html

Refactorizaciones: Cambiada configuración.

- Múltiples fallos al intentar evaluar prácticas con más de un paquete.

  Este defecto ha sido recurrente durante la mayor parte de las labores de mantenimiento. Se intentaron varias posibles soluciones, pero existían demasiadas cosas a tener en cuenta como para contemplarlas una a una. Por este motivo, se decidió comenzar la fase de modularización.

Errores de evaluación

- Se completa el proceso de evaluación pero no se guarda la calificación ni el comentario de realimentación. En la página de información sobre una entrega aparecen todos los campos en blanco pero la entrega sí está marcada como *Graded*.

  Cuando se hace la ejecución Java de todos los ficheros compilados, se redirige la salida del terminal a un fichero de texto del cual se extrae la información necesaria para calificar la entrega: el número total de test que se han ejecutado, la cantidad que han pasado correctamente y los que han fallado. Después, toda esta información se utiliza para calificar la entrega y generar un comentario de realimentación sobre la entrega. Dentro de las rutinas SQL[7] se intentaba guardar todos estos datos utilizando un id de usuario concreto en lugar de detectar el id del usuario que ha hecho la entrega. Como el curso de Moodle en el que se están haciendo estas pruebas es distinto al que se desarrolló, los ids de los usuarios y cursos no son los mismos. Se modificó la función process_save_submission para que, cuando se inicie el proceso de evaluación, se guarde el id del usuario[8] para usarlo en todas las llamadas posteriores.

  Refactorizaciones: Algoritmo reescrito.

- Solo se puede evaluar una práctica de prueba concreta.

  En multitud de funciones en `locallib.php` se hace uso de nombres concretos, que se refieren a una práctica de la asignatura de Estructuras de Datos de 2017, para todo el proceso de evaluación. Por ejemplo, cuando se tiene que sustituir dentro de `TestRunner.java` una etiqueta por el nombre de la clase que toca probar, se añadía directamente, es decir, había un *string* con lo que se tenía que poner en lugar de detectarlo para cada entrega. Se implementaron *scripts* para detectar: los nombres de las clases Java que hay que compilar, la clase Test de JUnit para la ejecución de las pruebas y el nombre del paquete en el que se encuentra esa clase.

  Refactorizaciones: Cambiados literales por variables.

- Cuando aparecen errores de compilación por subir una práctica incorrecta, la calificación obtenida es un 10.

  Antes de iniciar el proceso de calificación, había una variable auxiliar iniciada al valor entero "10"que se decrementaba con cada test JUnit fallido. Este comportamiento se sustituyó por completo de tal forma que cuando surja un error durante el proceso de calificación, se notifique pero no se guarde ninguna calificación, y solo cuando haya finalizado se calcule la nota en base a los porcentajes estipulados por el profesor en la entrega y a los resultados de cada una de las herramientas de evaluación (CheckStyle y JUnit). De modo que ahora se lanzan excepciones controladas [9] al detectar fallos en PHP.

  Refactorizaciones: Algoritmo reescrito.

[7] Sintaxis de inserción en SQL: https://www.w3schools.com/sql/sql_insert.asp

[8] Cómo conseguir el id del usuario en Moodle dentro de PHP: https://moodle.org/mod/forum/discuss.php?d=187465

[9] https://www.php.net/manual/es/language.exceptions.php

Errores de migración de Moodle

- Con la versión funcional en la máquina de desarrollo, no se ejecuta en la misma versión en el servidor del departamento.

  Este defecto fue particularmente difícil de detectar su origen. La versión de Moodle de las dos máquinas virtuales (ver 6.1) difiere, y en Moodle se cambiaron los permisos y el funcionamiento del usuario con el que se ejecutan las rutinas en la Bash. Todos los procesos arrancados por este usuario solo pueden operar dentro de `/var/www/moodledata/`. Tras comprobar las guías de uso de este directorio, se decidió sustituir todas las rutas que antes se dirigían a `/tmp/` hacia `/var/www/moodledata/temp/filestorage/`. De modo que, a partir de ahora, los directorios temporales de trabajo y los ficheros de *log* aparecerán en esta ruta. Tras estos cambios, el proceso de evaluación se completa correctamente en ambas máquinas.

  Refactorizaciones: Reemplazadas variables.

La siguiente tabla 1 sirve a modo de resumen de cada uno de los defectos encontrados mencionados anteriormente. En ella se indica además el usuario con el cual se origina el problema, este puede ser: alumno, profesor o desarrollador y el tipo de mantenimiento al que pertenece el defecto. Se listan en orden de descubrimiento durante el desarrollo.

Tabla 1: Defectos encontrados en EvalCode y soluciones aplicadas en cada caso.

Defecto encontrado	**Mantenimiento**	**Usuario**	**Solución resumida**
Imposible visualizar trazas de error.	Correctivo.	Desarrollador.	Modo de funcionamiento *Developer*. Cambiados permisos sobre ficheros de *log*.
No se pueden borrar actividades EvalCode.	Correctivo.	Profesor.	Cambiadas llamadas SQL.
Error al hacer una entrega.	Correctivo.	Alumno.	Añadido y arreglado `TestRunner.java`.
No se guarda la calificación ni el *feedback*.	Correctivo.	Alumno.	Reescritas las llamadas SQL de guardado.
Solo se puede evaluar una práctica de prueba concreta.	Correctivo.	Alumno.	Se hacía uso de nombres concretos que no estaban relacionados con la entrega. Se implementaron nuevos *scripts* de detección.
Modificar una actividad no guarda los nuevos datos.	Correctivo.	Profesor.	Comportamiento de modificación añadido.
No se puede volver a intentar realizar una entrega.	Correctivo.	Alumno.	Añadida finalización del hilo encargado. Modificada creación del directorio temporal.
Si hay errores de compilación se califica con un 10.	Correctivo.	Alumno.	Modificado por completo el proceso de calificación.
No funciona en el servidor del departamento.	Adaptativo.	Desarrollador.	Cambiados permisos y funcionamiento del usuario encargado de ejecutar rutinas en la Bash. Modificadas todas las rutas de trabajo y permisos.
La calificación no se suma correctamente y, a veces, aparecen errores de compilación.	Correctivo.	Alumno.	El cálculo de la nota se cambió por completo y se pasó a compilar con `UTF-8`.

Continúa en la siguiente página.

Tabla 1 – Continuación de la página anterior.

Defecto encontrado	Mantenimiento	Usuario	Solución resumida
CheckStyle comprueba ficheros que no pertenecen a la práctica entregada.	Correctivo.	Alumno.	Se comprobaban todos los archivos del directorio de trabajo en lugar de aquellos específicos de la entrega del alumno.

Dos de los defectos encontrados dieron lugar a nuevas funcionalidades añadidas a esta versión de EvalCode, por lo que se ha decidido separarlas en la tabla 2.

Tabla 2: Defectos encontrados en EvalCode que llevaron a crear nuevas funcionalidades.

Defecto encontrado	Mantenimiento	Usuario	Solución resumida
Error si la entrega depende de librerías externas.	Perfectivo.	Alumno.	Se añadió esta nueva funcionalidad incluyendo en el proceso de compilación las dependencias necesarias.
Múltiples fallos con prácticas con más de un paquete.	Perfectivo.	Alumno.	Defecto recurrente durante todas las labores de mantenimiento, se intentaron varias soluciones que funcionaron temporalmente pero finalmente se decidió pasar a la fase de modularización.

4.2.2. Modularización de EvalCode

Antes de comenzar la modularización del código es necesario conocer las características del sistema deseado. Los requisitos son los siguientes:

- Disponer de un *plug-in* capaz de evaluar una entrega con diferentes lenguajes y con una o varias herramientas que seleccione el profesor.
- El profesor debe ser capaz de seleccionar aquellas herramientas de evaluación que desee pasar sobre la entrega de un alumno y el porcentaje de la nota final de la práctica que irá asignado a cada herramienta.
- Un profesor (junto con la ayuda del administrador de Moodle) debe poder crear y añadir herramientas de evaluación con facilidad sin necesidad de tocar la estructura del propio Moodle y comprometer su seguridad.

Teniendo esto en mente, el primer paso a llevar a cabo es detectar aquellas funciones PHP que componen todo el proceso de evaluación de una entrega en EvalCode. Estas funciones están dentro de `locallib.php`, la principal (llamada por Moodle) es process_save_submission, que a su vez llama a: prepareFiles, prepareEvalFeedbackComment, saveEvalCodeGrade, unZipFile, findFileInDir, executeJunitTest, checkCodeQuality, parseFile, prepareNote y saveEvalCodeGradeFile. Estas funciones suman en total unas 642 líneas de código. Este dato es relevante para tener en cuenta que se pretende que la nueva versión sea más eficiente. En la figura 15 se representan los cambios que sufrió EvalCode durante esta fase del proyecto, cada una de estas modificaciones se describen en este apartado.

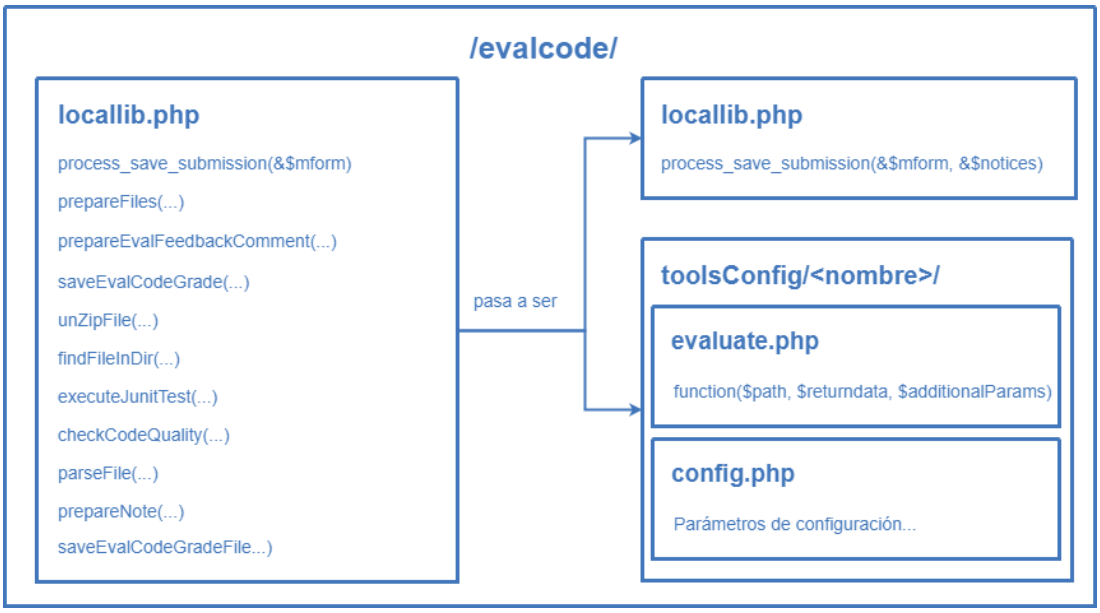


Figura 15: Diagrama con los cambios aplicados durante la modularización de EvalCode.

Como puede verse, se eliminaron la mayoría de las funciones menos una, process_save_submission, que pasa a ser la encargada de gestionar todo el proceso de evaluación.

Se introdujeron las llamadas "Herramientas de evaluación". Se trata realmente de dos *scripts* PHP que evalúan las entregas de los alumnos utilizando diferentes tecnologías y que indican a EvalCode las configuraciones necesarias para funcionar. Crear una nueva herramienta de evaluación para EvalCode es realmente sencillo gracias al modelo de implementación seguido. Como puede verse en la figura 16, dentro de los archivos de la extensión se añadió el directorio `/evalcode/toolsConfig/`. Aquí es donde se encuentran todas las herramientas de evaluación que se quieran incluir. Un profesor simplemente crea un nuevo directorio dentro de este y añade dos *scripts*: `config.php` y `evaluate.php`.

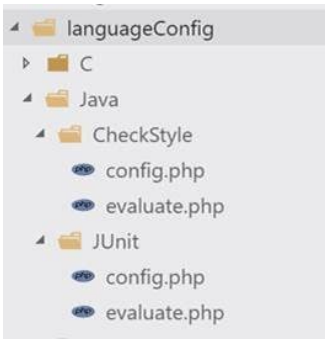


Figura 16: Localización y ejemplo de herramientas de evaluación.

En `config.php` se especifican una serie de parámetros: el nombre de la herramienta, la versión actual, una descripción breve y el lenguaje de programación que evalúa.

Por último solo queda programar el comportamiento de la herramienta en `evaluate.php`. EvalCode facilita toda esta codificación ofreciendo ya un directorio temporal en el que trabajar con el contenido de la entrega de un alumno y los ficheros de configuración que haya subido. De modo que el docente solo necesita añadir (por ejemplo) rutinas de compilación y ejecución y retornar por un lado la calificación y por otro el comentario de realimentación. Un desarrollo mucho más extenso y detallado sobre la creación de herramientas de evaluación para EvalCode se ha incluido en el Anexo 1, en el punto 9.1.2.

La función process_save_submission se encarga de: ejecutar cada una de las herramientas median-

te el uso de funciones anónimas PHP [29], aplicar la ponderación que haya especificado el profesor para una actividad a cada una de las herramientas de evaluación y de concatenar los comentarios de realimentación y darles un estilo legible. La figura 17 muestra un pseudocódigo de este comportamiento. Tras la implementación, esta función tiene una longitud de 262 líneas, lo cual ha supuesto una importante mejora en rendimiento y legibilidad dentro del propio código

```
función process_save_submission:
    inicia el proceso de evaluación de la entrega
    descargar XML con las herramientas de evaluación de la BD
    para cada herramienta de evaluación seleccionada hacer
        si la herramienta no tiene un script de evaluación
            lanzar mensaje de error
        fin si
        guardar entrega en la BD
        crear un directorio temporal
        descargar y descomprimir entrega
        descargar y descomprimir ficheros de evaluación del profesor
        si la entrega tiene el formato correcto
            llama a la función de evaluación de la herramienta: retorna nota y comentario
            multiplica la nota por el porcentaje asignado a esa herramienta
            guarda la nota y el comentario en una variable local
        si no
            lanza mensaje de error correspondiente
        fin si
    fin para
    guarda la nota total y el comentario de todas las herramientas en la BD
    finaliza el proceso de evaluación
fin función
```

Figura 17: Pseudocódigo de la nueva función encargada del proceso de evaluación.

El formulario de creación de una nueva actividad de EvalCode se cambió por completo. Además se trata de una implementación dinámica, porque cada vez que se va a añadir una nueva instancia a un curso se lee el directorio de herramientas y se incorpora cada una al formulario con la información correspondiente. Un detalle visual sobre cómo ha quedado el formulario puede consultarse en el apartado 5.2.

La base de datos del *plug-in* tuvo que ser modificada para cumplir con las nuevas especificaciones. Se borraron los campos que usaba la anterior versión para controlar información para la evaluación únicamente mediante JUnit y CheckStyle: *maxtestnumber*, *maxerrornumber*, *percentagetest* y *percentagequality* de la tabla `evalcode`. En su lugar se añadió el campo *selectedtools* de tipo texto, que se usará para almacenar un XML con las herramientas de evaluación y sus ponderaciones seleccionadas en una instancia de EvalCode en el formulario de creación. La figura 18 es un ejemplo del texto que se almacenaría en este campo de la BD si se seleccionasen las herramientas JUnit y CheckStyle con una ponderación de 70 % y 30 % respectivamente. Este XML es usado posteriormente por la función de control del proceso de evaluación para determinar qué herramientas tiene que pasar sobre la entrega del alumno y cuánto porcentaje de la nota final corresponde cada una.

```xml
<?xml version="1.0" encoding="utf-8"?>
<selected_tools>
    <tool>
        <name>JUnit</name>
        <language>Java</language>
        <percentage>70</percentage>
    </tool>
    <tool>
        <name>CheckStyle</name>
        <language>Java</language>
        <percentage>30</percentage>
    </tool>
</selected_tools>
```

Figura 18: Ejemplo de texto XML guardado en el campo *selectedtools* de la BD.

4.3. Versión final

En este punto, y con todo lo descrito anteriormente, se presenta la primera versión funcional final (1.0.0) de EvalCode, que cumple todos los requisitos estipulados en el punto anterior. Puesto que se trata de una herramienta de código abierto que seguirá en desarrollo, se invita a la comunidad docente a descargarla y contribuir a la misma. Para esto, se ha decidido alojar el proyecto en GitHub para que se pueda acceder con facilidad y aportar mejoras: https://github.com/aldeam/evalcode (figura 19).

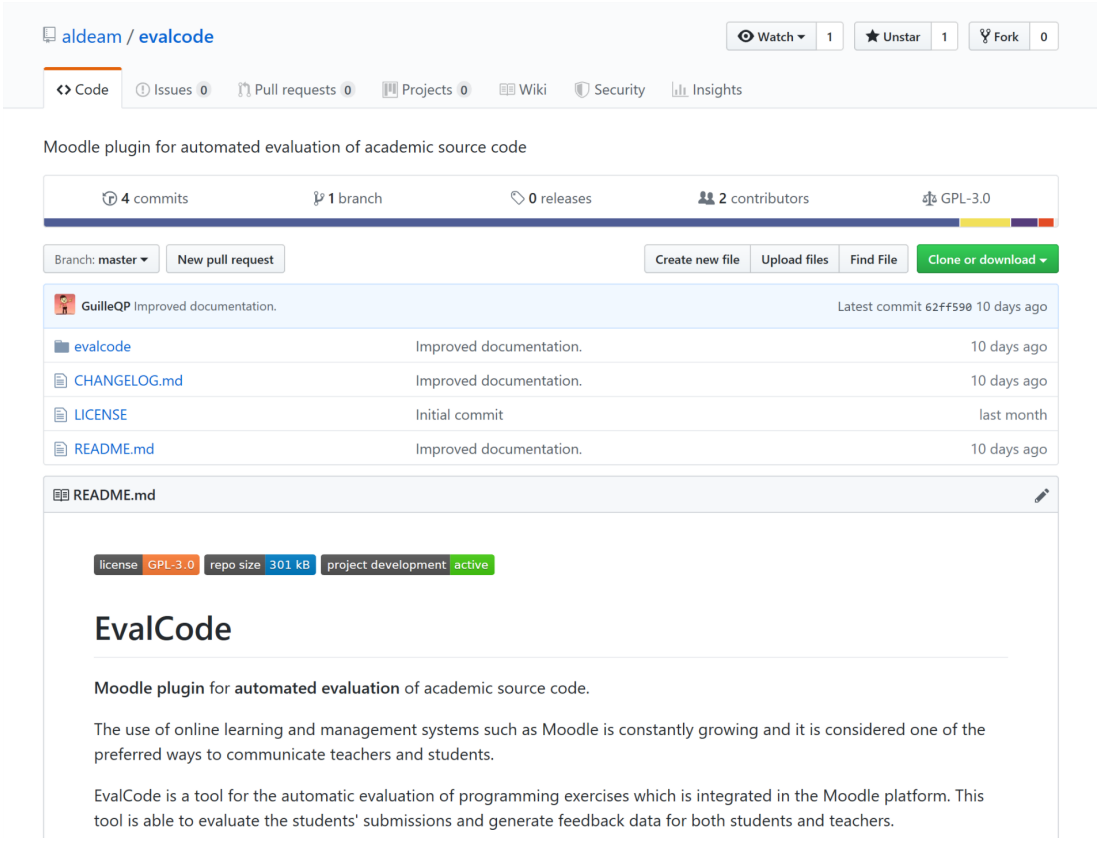


Figura 19: Repositorio GitHub de EvalCode.

Se ha creado documentación de EvalCode en *Markdown* para que aparezca en el repositorio, así

como un *log* de cambios en el que se detalla la versión actual y las funcionalidades que se han añadido, modificado o borrado. Con esto, se dispone de un sistema de versiones organizado que seguir a partir de ahora y fácil de comprender para alguien ajeno al desarrollo del *plug-in*.

En la figura 20 aparece una lista de ficheros que han sido añadidos, ~~eliminados~~ o modificados sobre la implementación inicial tras las labores de mantenimiento mencionadas en el apartado 4.2.1 y la modularización explicada en 4.2.2. Los archivos que no han recibido cambios sustanciales, o de ningún tipo, no se han incluido para una mayor claridad. Se añadieron un total de 11 archivos, la mayoría relacionados con las nuevas herramientas de evaluación implementadas. Sobre los que han sido modificados se ha aligerado el número de líneas de código en 512, dando lugar a *scripts* más legibles. Descontando los ficheros comunes a todo módulo de Moodle para funcionar, nos quedan 37 que son específicos de EvalCode; tras las labores de mantenimiento y la modularización un total de 26 fueron modificados de alguna manera, lo que da lugar a un 70,2 % de ficheros modificados que componen el *software* de la extensión.

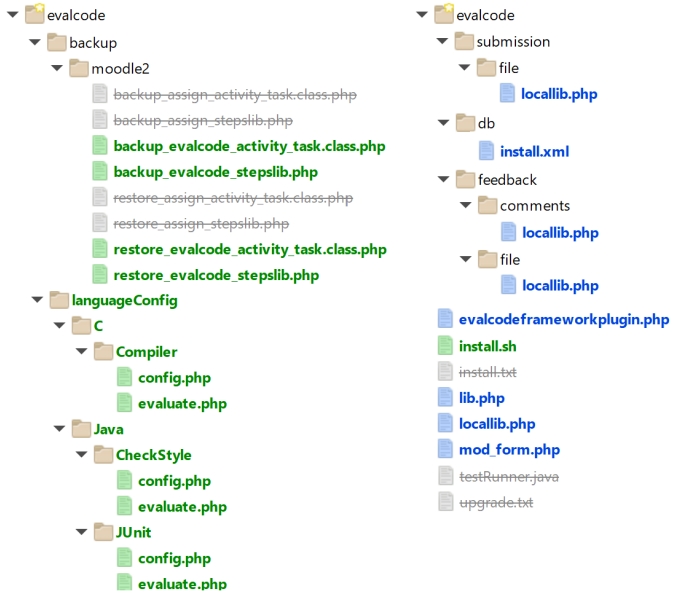


Figura 20: Lista (en dos columnas) con los ficheros que han cambiado.

5 Metodología de uso

En este apartado se describen, de forma más general, las guías de uso desarrolladas para la versión final de EvalCode en el Anexo 1 (9).

5.1. Descripción general

La extensión EvalCode permite evaluar ejercicios de programación utilizando herramientas de evaluación configuradas por un profesor para diferentes lenguajes de programación. La evaluación automática del código entregado por los alumnos se realiza en base a los criterios que se indiquen en el formulario de creación de la actividad.

En los siguientes apartados se procede a describir en más detalle cada una de las fases.

5.2. Creación de actividad EvalCode

Durante la creación de una actividad de tipo EvalCode el profesor debe configurar los aspectos requeridos por cualquier tarea Moodle: fechas de entrega, configuración de envío, grupos, etc. También debe indicar la ponderación en la calificación final del alumno de cada herramienta de análisis. Además, las herramientas pueden necesitar campos adicionales, como el número mínimo de test que se requieren para aprobar en el caso de JUnit. De esta forma, las calificaciones de los alumnos se calcularán proporcionalmente al valor que se indique en cada una. La figura 21 muestra un detalle del cuadro de diálogo correspondiente al comienzo de creación de una actividad EvalCode.

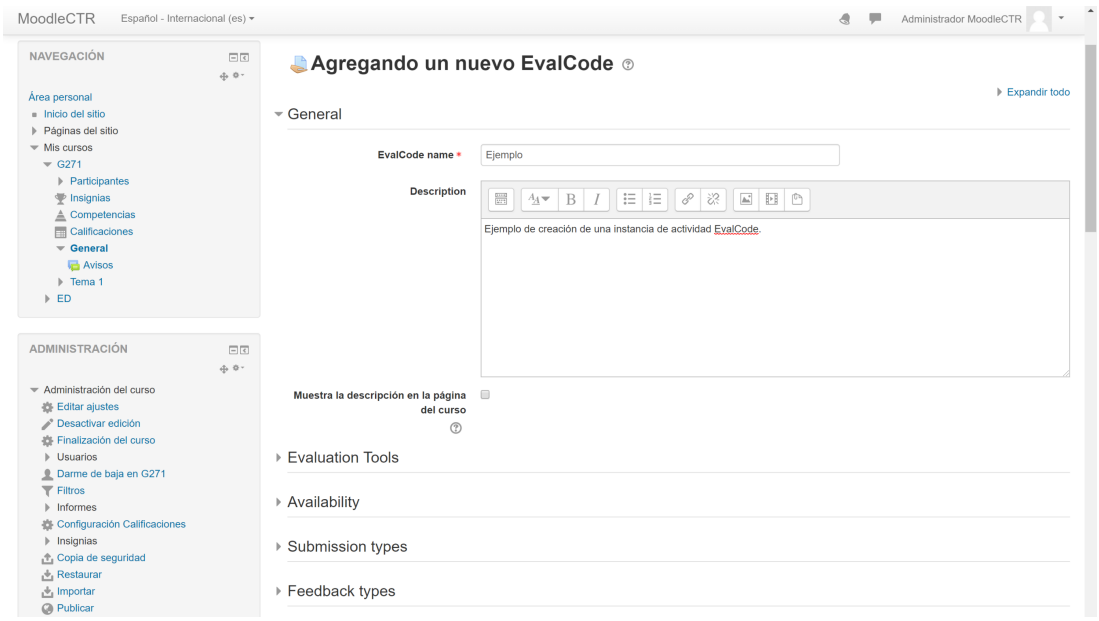


Figura 21: Página de creación de una actividad de EvalCode.

Dentro de la pestaña "Evaluation Tools" es donde se seleccionan las herramientas que se quieren usar para evaluar la entrega, como hemos dicho antes, se debe indicar el porcentaje de la nota final que corresponde a cada herramienta. En el ejemplo de la figura 22 aparecen tres herramientas (las que hay dentro de `/evalcode/toolsConfig/`).

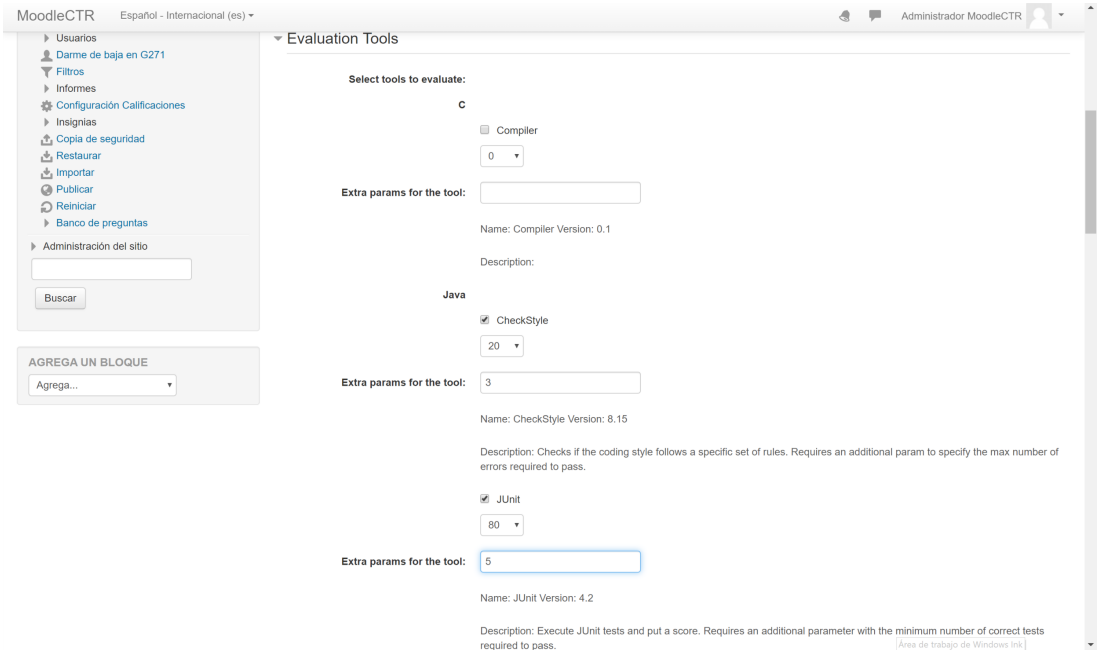


Figura 22: Selección de herramientas de evaluación para una actividad EvalCode.

Por último, el profesor debe incluir dos ficheros comprimidos `.zip` (figura 23). El primero con el código inicial que se quiere proporcionar al alumno para la realización de la tarea (campo "*Provided Files*"), puede ser código parcial o completamente implementado, o un conjunto de enunciados. Deberá incluir las librerías requeridas para la realización de la práctica (en el caso de que las hubiera). También se puede facilitar al alumno la configuración que utilizará la herramienta para evaluar (como un conjunto de test JUnit) para probar su código antes de realizar la entrega.

El segundo fichero (campo "*Evaluation Files*") es también un `.zip` cuyo contenido se descargará junto con la entrega en el directorio de trabajo de la herramienta para las labores de evaluación. Por ejemplo, puede ser un comprimido con las clases test de JUnit con las que se va a probar la práctica.

5.3. Entrega y realimentación

El alumno realiza la actividad partiendo del código inicial o enunciado proporcionado por el profesor. Dependiendo del ejercicio propuesto, la solución podría ser arbitrariamente compleja, pudiendo (en el caso de Java) requerir la implementación de nuevos paquetes, clases y/o interfaces. Este desarrollo puede realizarse utilizando cualquier IDE (p.ej. Eclipse). Si se desea, durante esta fase se puede permitir que el alumno ejecute en su ordenador un conjunto de pruebas para evaluar sus progresos antes de realizar la entrega.

Una vez realizado el ejercicio, el alumno entrega un fichero comprimido con el código desarrollado. El fichero debe incluir todo el contenido del directorio de fuentes (por ejemplo el directorio `src/` de un proyecto Eclipse si se está evaluando con la herramienta JUnit). Al realizar la entrega, EvalCode genera la realimentación para el alumno y calcula su calificación siguiendo los parámetros especificados por el profesor como se ha visto en el apartado anterior.

El resultado de la ejecución de cada herramientas de análisis junto con la calificación obtenida constituye la realimentación proporcionada al alumno. La calificación se registra en la tabla de calificaciones del alumno, como la de cualquier otra actividad calificable de un curso Moodle. El alumno

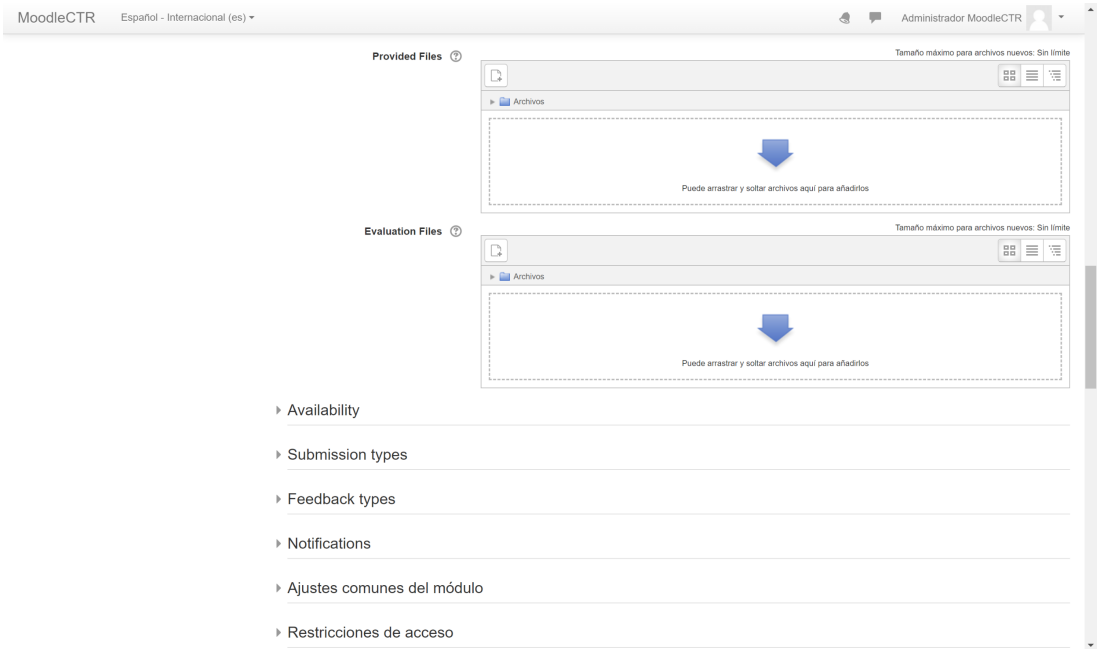


Figura 23: Campos para archivos adicionales de una nueva actividad EvalCode.

puede volver a realizar una entrega tantas veces como desee, salvo que el máximo número de intentos haya sido limitado en los parámetros de configuración de la tarea.

Las figuras 24, 25 y 26 se corresponden con tres intentos de entregas con las herramientas CheckStyle y JUnit seleccionadas con un 20 % y un 80 % de la calificación final respectivamente. La primera es una entrega para la que pasan todos los test de JUnit y sin errores de estilo. En la segunda se ha introducido un fallo de estilo en el código para que lo detecte el CheckStyle. Como puede observarse, aparece la traza del error cometido para que el alumno pueda seguirla hasta su fallo y la nota se calcula acorde a ese fallo, en este caso se trata de un comentario con un *TODO* que no se ha borrado. En la última se ha generado un error durante la compilación del código para demostrar que el proceso de evaluación no se detiene por este fallo; en concreto el error introducido es una clase Java sin el corchete de cierre. Podemos comprobar que en el apartado de la herramienta JUnit se notifica el error de compilación y se pone un 0 en esa parte, pero eso no ha impedido que la otra herramienta se ejecute y asigne su calificación.

La calificación asignada automáticamente al alumno puede ser posteriormente modificada por el profesor. Así, este puede realizar una revisión del código para evaluar aspectos difícilmente detectables por las herramientas automáticas, tales como la estructura del código o su eficiencia. Además, nuestra herramienta incluye una opción que permite recalificar las entregas de algunos o todos los alumnos utilizando un nuevo conjunto de pruebas.

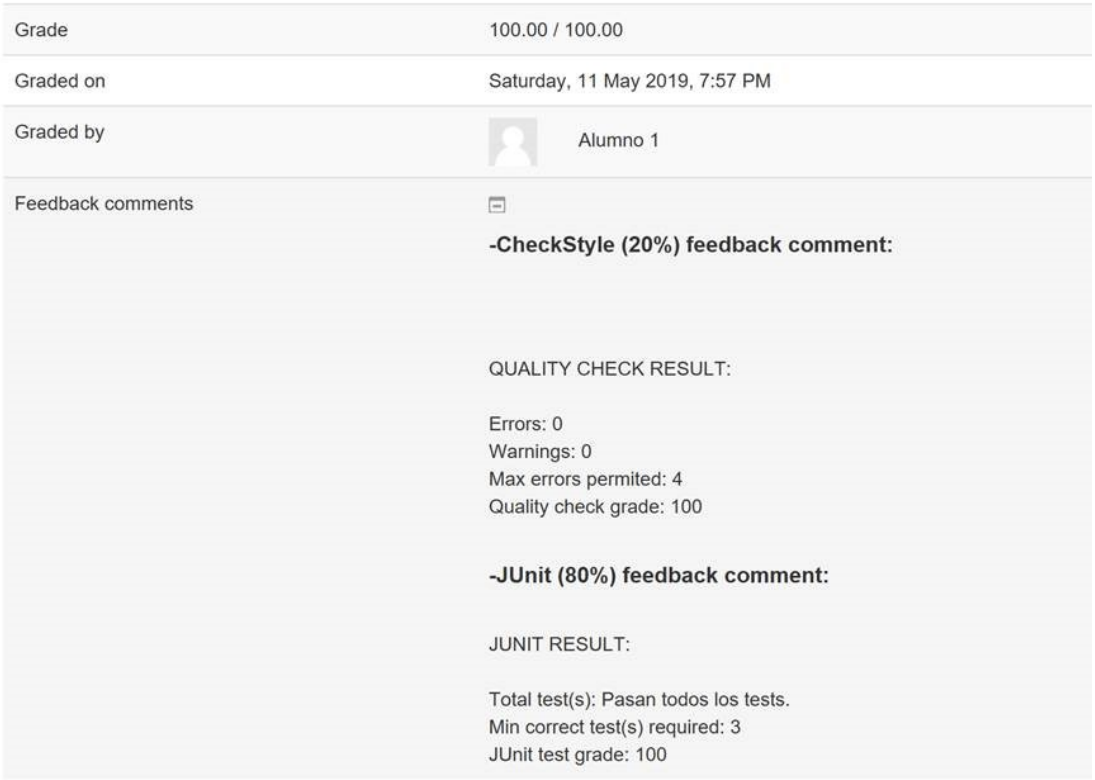


Figura 24: Ejemplo de *feedback* con la entrega perfecta y dos herramientas.

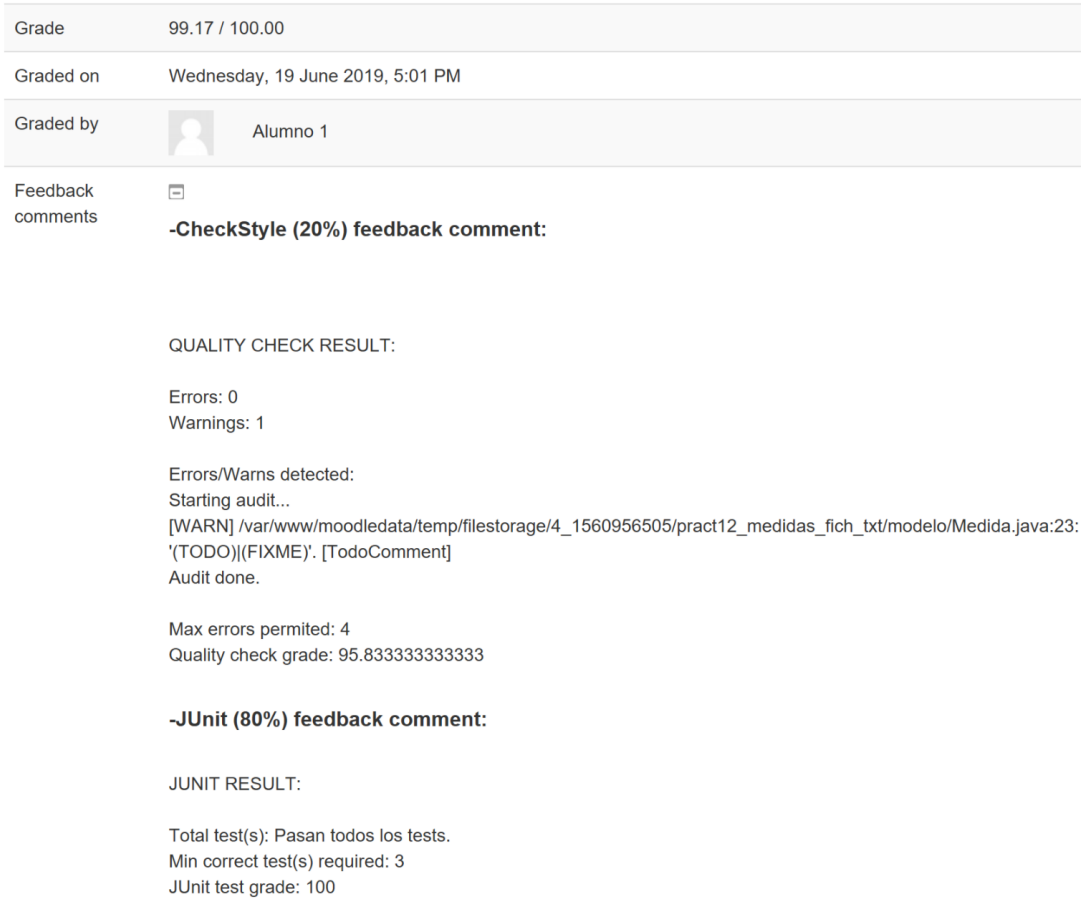


Figura 25: Ejemplo de *feedback* con un error de CheckStyle.

Figura 26: Ejemplo de *feedback* con un fallo de compilación.

6 Pruebas experimentales

A fin de poner EvalCode en explotación y de comprobar la viabilidad y recepción entre el alumnado y administración, se han realizado diversas pruebas. Las pruebas unitarias iniciales se hicieron en una máquina de desarrollo, pero para ponerlo a disposición de los alumnos se hizo necesario contar con un servidor de explotación temporal que sea accesible desde los laboratorios docentes, tal como ocurre con el servidor Moodle institucional. Hay que tener en cuenta que EvalCode nunca antes había sido probado en un entorno real y por eso las pruebas de la extensión no se realizaron sobre el Moodle de la Universidad de Cantabria ya que, al ser una versión aún en desarrollo, está sujeta a constantes cambios y podría ocasionar fallos desconocidos. Por este motivo se optó por desplegar un servidor de pruebas y explotación para este propósito.

6.1. Despliegue

Tanto para desarrollar EvalCode como para realizar las pruebas con alumnos es necesario disponer de un servidor Moodle. Por lo que durante la primera fase de desarrollo se dispusieron dos Máquinas Virtuales (MV) que actuarían simulando ser servidores de universidad en los que se aloja la plataforma Moodle. La propuesta es que sean lo más parecidos a un sistema real, con varios alumnos y profesores.

Servidor local de desarrollo

Este primer servidor se obtuvo del trabajo inicial de EvalCode, en esta máquina se desarrolló inicialmente el proyecto y se utilizará ahora para corrección de errores, añadir nuevas funcionalidades y pruebas generales. El sistema consta de una distribución Debian GNU/Linux 7.11 (wheezy) de 32 bits, además de la versión 3.1.2 de Moodle y Apache 2.2.22.

Esta MV tiene configurado el reenvío de puertos 22 y 80 al 10022 y 1080 respectivamente, así como la dirección IP 10.0.2.15 (de la MV) a la 127.0.1.1 (dirección local del PC anfitrión). Gracias a esto, podemos ejecutar a través de VirtualBox la máquina desde cualquier PC y acceder con facilidad al Moodle con cualquier navegador en la url `localhost:1080/` y conectarnos con SSH en `127.0.1.1:10022`. Para trabajar con más comodidad, se optó por usar la extensión de Visual Studio Code, *SSH File System*, que conecta un *workspace* del IDE (*Integrated Development Environment*) con el servidor; de este modo los cambios quedan guardados automáticamente y se puede usar un editor más completo y versátil que las herramientas que se pueden encontrar para trabajar directamente en la terminal de Linux.

Servidor de pruebas y explotación

Este servidor es una réplica del servidor utilizado por la UC para el Moodle institucional. Únicamente fue proporcionada la base del sistema, sin usuarios registrados ni cursos.

El propósito principal de este segundo servidor es el de mantenerse como alojamiento Moodle del Departamento de Ingeniería Informática de la Facultad de Ciencias de la Universidad de Cantabria para su uso en asignaturas relacionadas con la programación. Esto permite un mayor control sobre el *plug-in* EvalCode y sus herramientas de evaluación, a cambio de menores prestaciones (al tratarse de un solo equipo) y limitación en el acceso, ya que solo es accesible a través de la red cableada o wi-fi de la universidad para evitar problemas de seguridad.

Se trata de una Máquina Virtual alojada en un PC con Linux Mint, que ejecuta una distribución Ubuntu 18.04.1 LTS con Apache 2.4.34 (Ubuntu). Para acceder a ella vía SSH es necesario usar la IP cableada estática asignada, `193.144.198.46:22`, y al Moodle se puede entrar mediante la URL: `https://moodle.ctr.unican.es/moodle`. En este caso se ha proporcionado instalada la versión de Moodle 3.3.7. Para abrir los puertos en el *firewall* de la universidad para este nuevo servidor de explotación temporal fue necesario también contar con el apoyo de los vicerrectorados correspondientes, así como el del Servicio de Informática de la Universidad de Cantabria.

6.2. Pruebas de rendimiento

El hecho de que se utilizase una copia del entorno de explotación final facilitó la comunicación con el servicio de gestión informática de la universidad de cara a ponerlo en funcionamiento, así como resultar familiar para los alumnos por ser idéntico al que están acostumbrados a utilizar. Se introdujeron los alumnos en el curso de Moodle de prueba con sus nombres, correos electrónicos y DNI a modo de contraseña (ya que esta no podía ser facilitada desde el Moodle institucional).

Las primeras pruebas en el servidor de desarrollo sirvieron para depurar fallos en la codificación de los *scripts* como los mencionados en el apartado 4.2.1. Durante el proceso de mantenimiento de EvalCode se plantearon pruebas con prácticas reales para probar las modificaciones que se estaban realizando y detectar otros posibles fallos potenciales que se hayan pasado por alto. La asignatura empleada para estas pruebas ha sido "Estructuras de Datos" del segundo curso de Ingeniería Informática, que realiza prácticas de programación semanales y para las que los alumnos están ya bastante familiarizados con las herramientas de programación que utilizan.

En las pruebas participaron un total de 32 alumnos y consistieron en pedirles que desarrollen dos prácticas de programación en Java y que, además de subir sus prácticas al servidor Moodle que han venido empleando de forma habitual, lo hicieran, al finalizar la práctica en el aula, en el servidor de pruebas controlado. Por un lado se pretendía obtener información relevante sobre la opinión de EvalCode por parte del alumnado, y por otro obtener datos útiles sobre los tiempos que se tardaba en realizar una evaluación, las calificaciones que se calculaban y los posibles fallos que pudieran surgir.

6.2.1. Análisis y mejoras

Las prácticas seleccionadas para las pruebas tenían una complejidad de resolución similar. La figura 27 muestra las partes del proceso de entrega cuyos tiempos que se han medido instrumentando convenientemente el código de la herramienta. El tiempo de almacenado en la base de datos (a), el de extracción al sistema de ficheros y descompresión (b) y finalmente el de Ejecución de la evaluación propiamente dicha (c). Los tiempos de envío por la red en ambos sentidos así como el de compilación de la unidades a probar no han sido monitorizados.

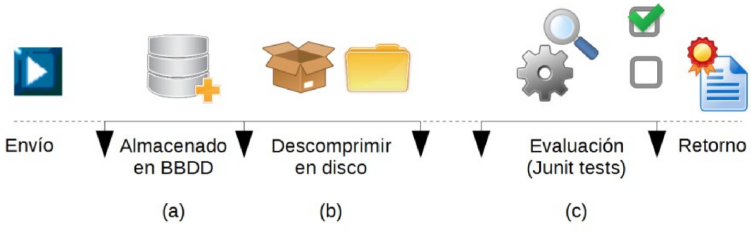
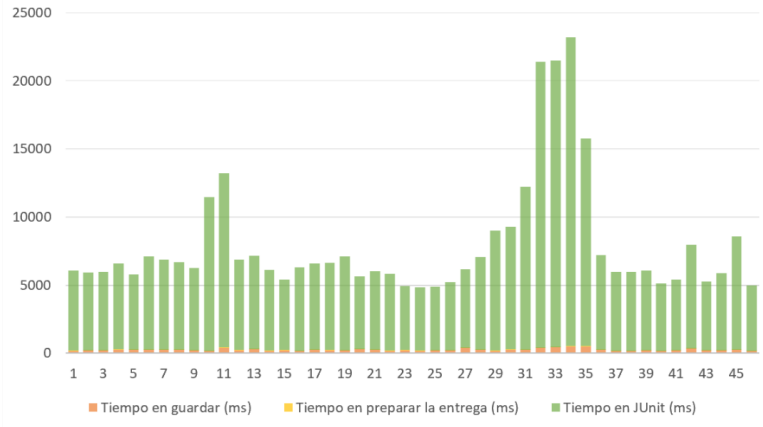


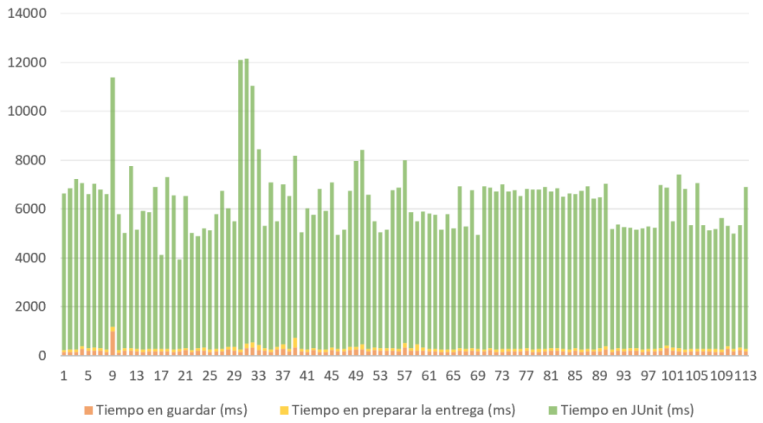
Figura 27: Fases del proceso de evaluación de una entrega.

Los resultados obtenidos para la primera práctica se muestran en la figura 28a. Como se puede apreciar el grueso del tiempo empleado se dedica a la evaluación de los *tests* y se agolpa cuando las entregas se hacen de forma simultánea. La nota media obtenida por los alumnos que entregaron vía EvalCode esta práctica fue de 5.41. En la segunda práctica se recibieron muchas más entregas pues

se pidió a los alumnos que las hicieran como forma de autoevaluación hasta conseguir los mejores resultados posibles. En este caso la nota media de los alumnos fue 6.3. La figura 28b muestra la distribución de tiempos de evaluación para las entregas recibidas en la segunda práctica



(a) Primera práctica



(b) Segunda práctica

Figura 28: Tiempos observados en las entregas.

La tabla de la figura 29 muestra los valores promedio calculados para los tiempos observados en las entregas de las dos practicas recibidas que se muestran gráficamente en la figura 28. Estos valores son razonables de cara a su uso en explotación en el servidor institucional. El tiempo empleado para el procesado de ficheros en la segunda práctica resulta notablemente mayor debido a que en esa práctica se proporcionaba una librería adicional.

	Guardado (ms)	Preparar ficheros (ms)	Junit (ms)
Primera práctica	238,01	17,44	7688,06
Segunda práctica	200,74	116,85	6066,75

Figura 29: Medias en los tiempos observados en las entregas de las dos prácticas.

6.2.2. Resultados y observaciones

Con toda la información recogida de estas pruebas sobre EvalCode se pudieron arreglar comportamientos erróneos de la extensión, así como mejorar otros que los alumnos señalaron. Las pruebas han

resultado satisfactorias en todos los aspectos contemplados: los alumnos se han adaptado fácilmente a su uso; el tiempo requerido por el servidor para la evaluación de las entregas ha sido aceptable y el uso de la realimentación (permitido en la segunda de las prácticas) se ha visto reflejado en una mejora de la nota obtenida. Esta mejora sobre las calificaciones ha sido posible debido principalmente a que esos comentarios de *feedback* daban la información necesaria al alumno para identificar sus errores y subsanarlos para volver a realizar la entrega hasta conseguir la mejor puntuación.

6.3. Pruebas de integración de herramientas y lenguajes

La versión final de EvalCode no ha sido aún probada en un entorno real con alumnos. Sin embargo, se han realizado multitud de pruebas sobre la máquina de desarrollo para garantizar que pueda quedar lista para la explotación en el siguiente curso académico y pueda formar parte en los procesos de evaluación de las asignaturas del Grado en Ingeniería Informática de la Universidad de Cantabria.

Además, para probar la nueva estructura modular de EvalCode, se añadió y probó con éxito una herramienta de comprobación de estilo para el lenguaje C, *Artistic Style* [30]. Para ello, se ha añadido el fichero de configuración `config.php` correspondiente como se muestra en la figura 30, indicando el nombre de la herramienta, la versión, el lenguaje para el que realiza la evaluación y una breve descripción.

evalcode ▸ toolsConfig ▸ C ▸ Astyle ▸ config.php

```php
<?php
/**
 * This file contains the configuration for a specific tool for EvalCode.
 */

//### AStyle ###
$languageTool = new \stdClass();
//Name for the tool, must be the same as the folder and without blank spaces
$languageTool->name = 'Astyle';
//Specify the current version of the tool
$languageTool->version = '3.1';
//Code language that the tool evaluates, must be the same as the language folder name and only one
$languageTool->language = 'C';
//Description for the tool
$languageTool->description = 'Artistic Style is a source code indenter, formatter, and beautifier for
                              the C, C++, C++/CLI, Objective-C and C# programming languages.';

return $languageTool;
?>
```

Figura 30: Contenido del fichero de configuración de la herramienta Astyle.

Como se ha visto en el apartado 4.2.2, hay que crear otro fichero `evaluate.php` con la lógica de la evaluación, que en este caso consiste en hacer llamadas a comandos de la *Bash* y guardar su salida como se muestra en la figura 31. Una explicación más detallada sobre el proceso de creación de nuevas herramientas de evaluación puede verse en el apartado 9.1.2 del Anexo 1.

Se configuró una nueva tarea EvalCode con el 100 % del peso de la nota sobre la herramienta Astyle y se hizo una entrega con un código simple para probar el funcionamiento. La calificación y comentario de realimentación obtenido se muestran en la figura 32. Este *feedback* resulta de gran ayuda ya que muestra los errores de estilo en la entrega (en rojo) y las correcciones que se le deben hacer al código (en verde).

evalcode ▸ toolsConfig ▸ C ▸ Astyle ▸ evaluate.php

```php
<?php
/**
 * This file contains the main logic for the evaluation process
 */
//### Astyle ###

/**
 * This is the function called when a student clicks the 'Submit' button
 * @param $path Is the temporal path in wich your function will operate, it contains
 *      the student submission and your evaluation files.
 * @param $returndata Is a generic class to specify the grade and the feedback comment.
 *      It's the only thing that this function should return.
 * @param $additionalParams This is a string containing all the additional information
 *      you've provided during the creation of your EvalCode activity.
 * @return $returndata Data class containing the grade and the feedback comment.
 * @throws NothingHere This is the example error to show how you sould use it. All
 *      the errors this function generates are treated and shown to the student in the
 *      sumbission comment box.
 * NOTE: This is an Anonymous function, that's the reason why it has to be inside $evaluatefunc.
 */
$evaluatefunc = function ($path,$returndata,$additionalParams){
    shell_exec('find * -name "*.c" > sources_list.txt');
    $contents = file_get_contents('sources_list.txt');
    $salida = shell_exec('style50 -o score @sources_list.txt 2>&1');
    if(strpos($salida,'error') || strpos($salida,'errors')){
        $returndata->grade = 0;
        $returndata->feedbackcomment = "Error: <br>".str_replace("\n", "<br>", $salida);
        return $returndata;
    }

    error_log("Sell exec: ".$salida."\n", 3, "/var/www/moodledata/temp/filestorage/evalcode.log");
    shell_exec('style50 -o split @sources_list.txt > feedback.log');
    $feedback = shell_exec('ansi2html < feedback.log');
    $grade = intval($salida);

    $comment = "";
    $comment .= "ASTYLE RESULT: <br>";
    $comment .= $feedback;
    $comment .= "<br>\tAstyle grade: " . $grade;

    $returndata->grade = $grade;
    $returndata->feedbackcomment = $comment;
    return $returndata;
};
?>
```

Figura 31: Contenido del fichero de evaluación de la herramienta Astyle.

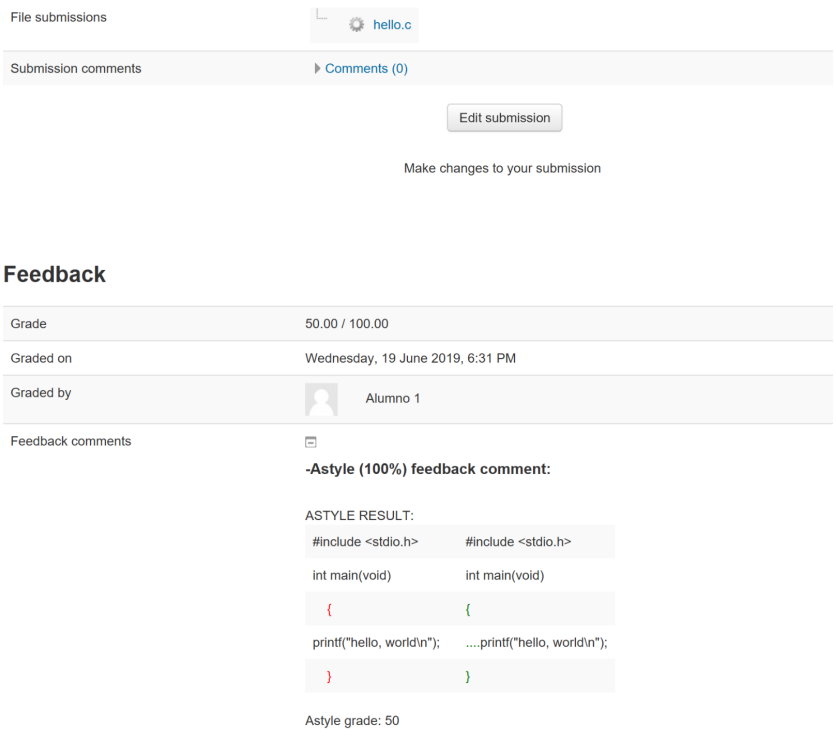


Figura 32: Ejemplo de calificación y *feedback* para una entrega que utiliza Astyle.

6.3.1. Aspectos a mejorar

Existen varios aspectos a ampliar o mejorar en la herramienta, entre ellos los más importantes podrían ser la integración de nuevas herramientas de análisis y de control de plagio y el soporte para otros lenguajes de programación. En particular es importante añadir la posibilidad de invocar herramientas de análisis de calidad y auditoría, como Sonar por ejemplo, de modo que en cursos más avanzados los alumnos reciban retroalimentación más exhaustiva de la deuda técnica y otras figuras de mérito relacionadas con la calidad global de su trabajo.

7 Conclusiones y planes para el futuro

En este proyecto se ha descrito la herramienta EvalCode, una extensión de Moodle para la evaluación automática de código. EvalCode permite analizar el código entregado por el alumno desde diferentes puntos de vista utilizando distintas herramientas de evaluación

El resultado del análisis realizado por cada herramienta junto con la calificación obtenida constituyen la realimentación proporcionada al alumno. La versión actual de la herramienta permite evaluar ejercicios de programación en cualquier lenguaje y se ha probado para Java y C. La estructura interna de EvalCode está pensada para facilitar la integración de nuevas herramientas. A diferencia de otras herramientas existentes, EvalCode permite analizar código arbitrariamente complejo (constituido por un conjunto de paquetes con clases y/o interfaces), el cuál puede haber sido desarrollado utilizando cualquier IDE (p.ej., Eclipse). Otra ventaja de EvalCode respecto a otras herramientas similares es la simplificación de la tarea del profesor, que fundamentalmente se centra en el diseño del *script* de evaluación. Las pruebas de la herramienta han resultado satisfactorias en todos los aspectos contemplados: los alumnos se han adaptado fácilmente a su uso; el tiempo requerido por el servidor para la evaluación de las entregas ha sido aceptable y el uso de la realimentación se ha visto reflejado en una mejora de la nota obtenida. Existen varios aspectos a ampliar o mejorar en EvalCode, entre ellos los más importantes podrían ser la integración de nuevas herramientas de análisis y de control de plagio.

Por otro lado, uno de los propósitos principales de esta memoria es que sirva como guía inicial para todos aquellos desarrolladores que quieran adentrarse en el mundo de desarrollo de módulos para la plataforma Moodle. Para el desarrollo de esta memoria se ha buscado información de un gran número de webs y foros relacionados con la plataforma con el fin de recopilar la información básica y esencial que debe conocer cualquier persona que esté interesada en desarrollar un módulo. En este punto es importante destacar que, aunque Moodle tiene una documentación bastante extensa, en la mayoría de las ocasiones esa documentación es demasiado técnica o poco intuitiva a la hora de empezar con el desarrollo, por lo que gracias a foros o a ejemplos de otros desarrolladores se han podido solventar las diferentes dudas que han surgido durante el desarrollo.

8 Bibliografía

[1] Ricardo Alexandre Peixoto Queirós y José Paulo Leal. Petcha: a programming exercises teaching assistant. ACM ITiCSE, pages 192–197, 2012.

[2] Rafael M. Luque-Baena Francisco Luna Luis Arévalo, Francisco J. Rodríguez. Experiencia con una herramienta de pruebas de caja negra para el aprendizaje de asignaturas de programación en evaluación continua. JENUI, pages 61–68, 2017.

[3] Rubén Sebrango Briz. Marco para la evaluación automática de código basado en moodle. SAGE Open, pages 1–64, 2017.

[4] Danny S. Guamán y Julio C. Caiza Mónica Guerrero. Revisión de herramientas de apoyo en el proceso de enseñanza-aprendizaje de programación. Revista Politécnica vol. 35, pages 82–90, 2015.

[5] Sugandha Gupta y Anamika Gupta. Assessment tools for programming languages: A review. ICITKM, pages 65–70, 2018.

[6] Rafael M. Luque-Baena Francisco Luna Luis Arévalo, Francisco J. Rodríguez. Experiencia con una herramienta de pruebas de caja negra para el aprendizaje de asignaturas de programación en evaluación continua. JENUI, pages 61–68, 2017.

[7] Richard Lobb y Jenny Harlow. Coderunner: a tool for assessing computer programming skills. ACM Inroads, pages 47–51, 2016.

[8] Dominique Thiébaut. Automatic evaluation of computer programs using moodle's virtual programming lab (vpl) plug-in. J. Comput. Sci. Coll. , Vol 30, No. 6, pages 145–151, 2015.

[9] SonarSource. Sonarqube. https://www.sonarqube.org/. Accedido el 2019-6-3.

[10] The PHP Group. What is php. https://www.php.net/manual/en/intro-whatis.php. Accessed on 2019-5-3.

[11] Carlos Eduardo Plasencia Prado. ¿qué es y por qué aprender sql? https://devcode.la/blog/que-es-sql/. Accedido el 2019-5-12.

[12] Oracle. Java. https://www.java.com/es/about/whatis_java.jsp. Accedido el 2019-5-14.

[13] Mozilla. Introducción a xml. https://developer.mozilla.org/es/docs/Web/XML/Introducción_a_XML. Accedido el 2019-5-14.

[14] Javier Cristóbal. Markdown. https://markdown.es/. Accedido el 2019-5-27.

[15] Moodle Org. Acerca de moodle. https://docs.moodle.org/all/es/Acerca_de_Moodle. Accedido el 2019-5-14.

[16] Microsoft. Visual studio code. https://code.visualstudio.com/docs. Accedido el 2019-5-14.

[17] FileZilla Org. Filezilla, the free ftp solution. https://filezilla-project.org/. Accedido el 2019-5-14.

[18] Putty project. https://www.putty.org/. Accedido el 2019-5-14.

[19] Oracle. Virtualbox. https://www.virtualbox.org/. Accedido el 2019-5-14.

[20] Kai Willadsen. Meld. https://meldmerge.org/. Accedido el 2019-6-3.

[21] Inc. GitHub. Github. https://github.com/. Accedido el 2019-6-3.

[22] Oliver Burn. Checkstyle. http://checkstyle.sourceforge.net/. Accedido el 2019-6-5.

[23] JUnit Team. Junit. https://junit.org/junit5/. Accedido el 2019-6-5.

[24] Free Software Foundation. Gnu gpl. https://www.gnu.org/licenses/licenses.es.html. Accedido el 2019-6-6.

[25] PEAR Manual. Validation filters. https://pear.php.net/manual/en/package.html.html-quickform.intro-validation.php. Accedido el 2019-5-12.

[26] Inc. Eclipse Foundation. Eclipse. https://www.eclipse.org/. Accedido el 2019-6-5.

[27] Mario Piattini et al. Mantenimiento del software. Ra-Ma, pages 3–8, 2000.

[28] Tought Works. Refactoring. https://refactoring.com/catalog/. Accedido el 2019-5-22.

[29] The PHP Group. Funciones anónimas. https://www.php.net/manual/es/functions.anonymous.php. Accedido el 2019-5-6.

[30] Jim Pattee. Artistic style 3.1. http://astyle.sourceforge.net/. Accedido el 2019-5-23.

9 ANEXO 1 - Guías de uso

9.1. Administrador

La figura del administrador en este caso se entiende por aquella persona (o grupo) que se encarga del mantenimiento de los servicios web de una universidad o institución de enseñanza que utilice el servicio Moodle. Las tareas que recaen sobre el Administrador para hacer operativo EvalCode son las descritas a continuación.

9.1.1. Instalar y configurar EvalCode

Antes de instalar EvalCode es necesario comprobar si el sistema sobre el que se va a instalar cumple una serie de requisitos.

Requisitos

- El *plug-in* ha sido probado en versiones de Moodle hasta la 3.3.7. Se desconoce si versiones posteriores podrían afectar al funcionamiento de la extensión. La última versión con mantenimiento a largo plazo (2021) de Moodle es la 3.5.2.
- Sistema con base Linux (probado para Weezy y Ubuntu, ver 6.1) y con acceso *root*.
- Apache2.

Instalación/Actualización

Con Moodle instalado sobre la máquina o servidor, es necesario localizar el directorio que contiene todas las extensiones de la plataforma, por defecto suele ser `/var/www/html/moodle/mod/` aunque puede variar dependiendo de la distribución de Linux utilizada. Una vez aquí, hay que copiar todos los archivos de EvalCode en un nuevo directorio `mod/evalcode/` que tenga todas sus letras en minúscula, sin espacios y sin caracteres inválidos (ñ o ´), este nuevo directorio debe pertenecer a *root root* (usuario y grupo, respectivamente).

Tras esto, accedemos a Moodle con credenciales de administrador y nos debería aparecer una página que indica la detección de una nueva extensión y que es necesario configurarla y actualizar la BD. Aceptamos y configuramos la herramienta como convenga. Las opciones están marcadas por defecto para un funcionamiento estándar. Cuando Moodle avise con un mensaje de “éxito”la extensión está lista para usarse.

9.1.2. Añadir herramientas de evaluación

La creación de nuevas herramientas normalmente recae sobre el profesor, ya que será él quién programe las distintas funcionalidades que necesite, pero, aparece en este apartado porque requiere acceso a los archivos principales de EvalCode.

Crear una nueva *language tool*

Las herramientas de evaluación se encuentran en `../moodle/mod/evalcode/languageConfig/`, son las que un profesor selecciona para evaluar y calificar la entrega de un alumno. Para empezar es necesario incluir dentro de la ruta un directorio con el nombre del lenguaje que va a evaluar y, dentro, otro directorio con el nombre se quiere que tenga la herramienta, sin espacios, como puede verse en 33.

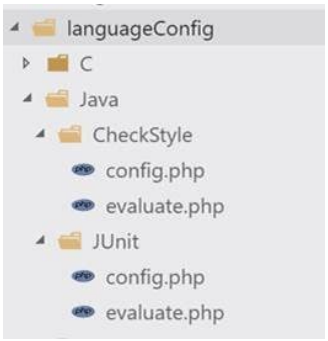


Figura 33: Localización y ejemplo de herramientas de evaluación.

Esta nueva herramienta necesita dos scripts PHP, uno de configuración [fig. 34] (config.php) que contiene información básica de la herramienta como el nombre y la versión; y otro con la lógica de la evaluación [fig. 35] (evaluate.php), esto es, el código necesario para comprobar una entrega (compilar, ejecutar…) y poner una nota y el feedback correspondiente. En el primero se indica: el nombre de la herramienta, que debe ser igual al que aparece en el nombre del directorio; la versión de desarrollo o la que corresponde a otro material, como por ejemplo compiladores; el lenguaje que evalúa, de nuevo, igual que el que aparece en el directorio; y, por último, una breve descripción de la *tool* en la que debe aparecer también si necesita que el profesor añada algún parámetro adicional a la entrega. Estas especificaciones se muestran al profesor en la lista de posibilidades durante el proceso de creación de una nueva actividad EvalCode. En el segundo se programará el comportamiento de la herramienta

```php
<?php
/**
 * This file contains the configuration for a specific tool for EvalCode.
 */

//### NEW TOOL ###
$languageTool = new \stdClass();
//Name for the tool, must be the same as the folder and without blank spaces
$languageTool->name = 'Compiler';
//Specify the current version of the tool
$languageTool->version = '0.1';
//Code language that the tool evaluates, must be the same as the language folder name and only one
$languageTool->language = 'C';
//Description for the tool
$languageTool->description = '';

return $languageTool;
?>
```

Figura 34: Script de configuración (*config.php*).

como tal, puede constar simplemente de una llamada PHP a un método que ejecute un *script* de la *shell*, siempre y cuando retorne la nota con la que se quiere calificar al alumno entre 0 y 100, y el comentario de *feedback* que se proporcionará al alumno. Este comentario se retorna en formato HTML, se ha implementado de esta manera que resulte fácil añadir estilo a esta salida. En la figura 35 se puede apreciar que la función está declarada en la asignación a una variable `evaluatefunc`, esto se conoce como función anónima, también conocidas como cierres (*closures*), permiten la creación de funciones que no tienen un nombre especificado.

```php
<?php
/**
 * This file contains the main logic for the evaluation process
 */

/**
 * This is the function called when a student clicks the 'Submit' button
 * @param $path Is the temporal path in wich your function will operate, it contains
 *      the student submission and your evaluation files.
 * @param $returndata Is a generic class to specify the grade and the feedback comment.
 *      It's the only thing that this function should return.
 * @param $additionalParams This is a string containing all the additional information
 *      you've provided during the creation of your EvalCode activity.
 *
 * @return $returndata Data class containing the grade and the feedback comment.
 *
 * @throws NothingHere This is the example error to show how you sould use it. All
 *      the errors this function generates are treated and shown to the student in the
 *      sumbission comment box.
 * NOTE: This is an Anonymous function, that's the reason why it has to be inside $evaluatefunc.
 */
$evaluatefunc = function ($path, $returndata, $additionalParams){
    throw new Exception('NothingHere');

    $returndata->grade = 0;
    $returndata->feedbackcomment = '';
    return $returndata;
};
?>
```

Figura 35: Script con la lógica de evaluación (*evaluate.php*).

9.2. Profesor

9.2.1. Nueva entrega

Durante la creación de una actividad de tipo EvalCode el profesor debe configurar los aspectos requeridos por cualquier tarea Moodle: fechas de entrega, configuración de envío, grupos, etc. Después, el profesor deberá seleccionar aquellas herramientas de análisis que desea realicen la evaluación, así como indicar la ponderación en la calificación final del alumno de cada una de ellas (la suma de todas ellas debe ser un 100 %). Alguna herramienta puede necesitar uno o más parámetros adicionales de configuración. Por ejemplo, la herramienta JUnit precisa indicar el número de fallos que correspondería a una calificación de 5, de forma que las calificaciones de los alumnos se calcularán proporcionalmente a dicho valor.

Por último, el profesor debe incluir dos ficheros comprimidos:

- Proporcionado: conjunto de paquetes con enunciados y código, parcial o completamente implementado, que constituye el material proporcionado al alumno. Deberá incluir las librerías requeridas para la realización de la práctica (en el caso de que las hubiera). Entre ellas se podría incluir un *script* de evaluación (igual al que usará la herramienta de análisis o menos completo) para facilitar al alumno la prueba de su código antes de realizar la entrega.

- Fichero comprimido con el código final: incluye aquellas clases e interfaces del código inicial que el alumno no debería haber modificado. También incluye la clase probadora con la que se realizará la evaluación (que puede ser igual o diferente de la proporcionada en el código inicial). Las reglas utilizadas por CheckStyle para la evaluación del estilo del código son configurables por el profesor. En el estado actual de nuestra extensión EvalCode, dichas reglas se definen a nivel de servidor Moodle, lo que significa que son comunes para todas las tareas y cursos en dicho servidor.

9.2.2. Revisión de entregas y calificaciones

La calificación asignada automáticamente al alumno puede ser posteriormente modificada por el profesor. Así, el profesor puede realizar una revisión del código para evaluar aspectos difícilmente detectables por las herramientas automáticas, tales como la estructura del código o su eficiencia.

Además, nuestra herramienta incluye una opción que permite recalificar las entregas de algunos o todos los alumnos utilizando un nuevo conjunto de pruebas.

9.3. Alumno

El alumno realiza la actividad partiendo del código inicial proporcionado por el profesor. Dependiendo del ejercicio propuesto, la solución podría ser arbitrariamente compleja, pudiendo requerir la implementación de nuevos paquetes, clases y/o interfaces. Este desarrollo puede realizarse utilizando cualquier IDE (p.ej. Eclipse). Si se desea, durante esta fase se puede permitir que el alumno ejecute en su ordenador un conjunto de pruebas para evaluar sus progresos antes de realizar la entrega. En ese caso, dicho conjunto de pruebas habría sido incluido en el código inicial proporcionado por el profesor (en la forma de una clase probadora de JUnit).

9.3.1. Entrega y realimentación

Una vez realizado el ejercicio, el alumno entrega un fichero comprimido con el código desarrollado. El fichero debe incluir todo el contenido del directorio de fuentes (p.ej. el directorio src/ de un proyecto Eclipse). Al realizar la entrega, nuestra herramienta genera la realimentación para el alumno y calcula su calificación.

La herramienta descomprime en primer lugar el fichero entregado por el alumno y, a continuación, el código final proporcionado por el profesor. De esta forma las clases que no deberían haber sido modificadas y, en particular, la clase probadora, son sobre escritas con el código proporcionada por el profesor, descartando los posibles cambios realizados por el alumno. A continuación se ejecutan las herramientas de análisis (JUnit y CheckStyle). El resultado de la ejecución de las herramientas de análisis junto con la calificación obtenida constituye la realimentación proporcionada al alumno. La calificación se registra en la tabla de calificaciones del alumno, como la de cualquier otra actividad calificable de un curso Moodle. El alumno puede iterar entre las fases de realización y entrega tantas veces como desee salvo que el máximo número de entregas haya sido limitado en los parámetros de configuración de la tarea.