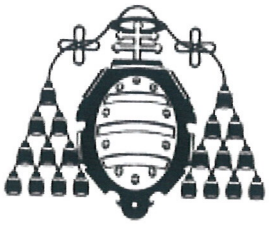
UNIVERSIDAD DE OVIEDO



ESCUELA DE INGENIERÍA INFORMÁTICA

TRABAJO FIN DE MASTER
(Especialidad Investigación)

"Especificaciones de analíticas de aprendizaje aplicadas a entornos integrados de desarrollo"

Vº Bº del Director del Proyecto

DIRECTOR: Juan Ramón Pérez Pérez

AUTOR: Luis Valdés Cuesta

Agradecimientos

Quiero agradecer el apoyo recibido durante la realización de este trabajo fin de master a las siguientes personas.

Gracias a Juan Ramón Pérez Pérez por todo el tiempo y paciencia que me ha dedicado durante el proyecto, sobre todo por la dificultad de trabajar a distancia.

Gracias a mi hermano y a mis padres, especialmente a mi madre, por animarme y apoyarme para acabar este proyecto.

Y gracias a Mireia por hacerme más fácil el tiempo dedicado a este trabajo.

Resumen

Aprender a programar es una tarea complicada. Existen herramientas como los entornos integrados de desarrollo (IDE) que facilitan la programación, pero aún así es fácil cometer errores. Durante las sesiones prácticas o talleres de programación, los alumnos aplican los conceptos teóricos aprendidos, guiados por un profesor. Estos talleres se llevan a cabo con múltiples alumnos en salas de ordenadores compartidos, por lo que el profesor no puede realizar un seguimiento individualizado.

Algunos patrones de comportamiento pueden explicar por qué los estudiantes tienen tantas dificultades para mejorar sus habilidades de programación, pero los profesores no pueden detectar esos patrones en todos los alumnos a la vez. En este proyecto se utilizan técnicas de Learning Analytics (analíticas de aprendizaje) para capturar las acciones de los usuarios en el IDE Eclipse durante los talleres de programación, después se procesan, analizan y se muestran en un dashboard interactivo. Con ello se pretende facilitar a los profesores la tarea de supervisión de grandes cantidades de alumnos, uno a uno y en tiempo real, y la detección de malas estrategias de aprendizaje o patrones de comportamiento.

Palabras Clave

Analíticas de aprendizaje, comportamientos de programación, xAPI, Entornos Integrados de Desarrollo (IDE), aprendizaje de programación, e-learning.

Abstract

Learning software programming is a difficult task, even using tools such as integrated development environments (IDEs), it is easy to introduce programming errors. During programming sessions or workshops, students apply the concepts learned, guided by a teacher. These programming workshops take place with multiple students at the same time, making it impossible for teachers to supervise all the students in the classroom, one by one.

Some behavior patterns may explain why students struggle improving their programming abilities, but teachers are not able to detect those patterns when supervising all the students at the same time. This project utilizes Learning Analytics to capture students' actions inside Eclipse IDE during programming workshops, and then the data is processed, analyzed and displayed in an interactive dashboard. This aims to help teachers easily review the progress of every student, one by one, in real time, and detect bad learning strategies or behavior patterns.

Keywords

Learning analytics, programming behaviors, xAPI, Integrated Development Environments (IDE), programming learning, e-learning.

Índice General

Índice de Figuras

1 Introducción

La programación software forma parte de varios campos de estudio y se enseña en distintas especialidades universitarias, desde ingenierías industriales, informáticas, telecomunicaciones, hasta multimedia. El aprendizaje de la programación presenta ciertas dificultades a los estudiantes, ya que requiere comprender conceptos abstractos y conocer una sintaxis muy específica, por lo que es fácil cometer errores. Incluso cuando los estudiantes conocen la sintaxis y la semántica de un lenguaje de programación, no siempre saben combinar las sentencias correctamente para generar un programa válido [1].

Durante las sesiones prácticas o talleres en clase, los alumnos adquieren nuevos conocimientos y ponen en práctica los que ya tienen a través del desarrollo de ejercicios de programación. Normalmente estas sesiones prácticas se llevan a cabo en salas de ordenadores compartidos con un profesor que, o bien guía la sesión completamente o únicamente resuelve dudas cuando los alumnos lo solicitan de forma individual. En estas sesiones, los estudiantes realizan ejercicios de programación desarrollando programas funcionales y algoritmos utilizando algún tipo de estructuras de datos, pero debido a la inexperiencia, es común que se encuentren con problemas inesperados difíciles de solucionar. En esos casos, los alumnos pueden llegar a bloquearse sin saber por dónde atacar el problema y cómo alcanzar una solución, dificultando su aprendizaje de la programación. Gracias a la ayuda del profesor presente durante las sesiones, los estudiantes pueden comprender el problema y solventarlo.

El proyecto desarrollado pretende ayudar a los profesores a visualizar de forma rápida el progreso de los alumnos en las sesiones prácticas de programación. De esta forma, los profesores pueden detectar visualmente los bloqueos de los alumnos en un proyecto, bien en tiempo real o bien más tarde durante la revisión de la sesión. La herramienta también puede utilizarse como un histórico donde se almacenan las sesiones de programación de los alumnos, pudiendo acudir a ella más tarde para ver el progreso de un alumno a lo largo del tiempo, proyecto a proyecto, y así evaluar no sólo el resultado final, sino cómo ha llegado el alumno hasta él, proporcionando nuevas perspectivas. Con esta información, el profesorado puede identificar problemas de comprensión de conceptos, modificar la forma de dar las clases o adaptar los contenidos para reforzar esos conceptos en los que los alumnos presentan más dificultades.

La herramienta se compone de un *plug-in* para el entorno de desarrollo integrado Eclipse que realiza la captura de datos, un servidor central donde se almacenarán esos datos y un panel de control web donde mostrar información útil para el profesorado mediante la aplicación de analíticas de aprendizaje sobre los datos almacenados en el servidor central. Las analíticas de aprendizaje, o *learning analytics*, son técnicas que consisten en seguir el rastro de los estudiantes en el ámbito educativo para medir, analizar y representar sus acciones y comportamientos con el fin de mejorar el proceso de aprendizaje.

1.1 Motivación

Existen herramientas que facilitan la programación, como los entornos de desarrollo integrados (IDEs). Son aplicaciones profesionales que proporcionan ayudas como asistentes para creación de proyectos, compilación en tiempo real, ventanas con información contextual

mientras se escribe el código fuente, etc. Aún así, es fácil cometer errores de programación que se detectarán en tiempo de compilación o, más tarde, en tiempo de ejecución.

El dominio de las habilidades de programación no es fácil de adquirir [2]. Cuando los alumnos están aprendiendo a programar no tienen un conocimiento completo del lenguaje de programación o la técnica, sino fragmentos. Saben algunas sentencias del lenguaje, algunas estructuras de datos, etc. Los alumnos en ese estado tienen lo que se denomina un conocimiento frágil [3], [4], lo que puede ayudar a explicar las dificultades que los alumnos encuentran durante el aprendizaje.

Jadud [5], Blikstein [6] y otros autores [7], [8] observan un comportamiento común entre los estudiantes de programación que identifican y clasifican en tres categorías o tipos de estudiantes: *stoppers*, *movers* y *tinkerers*.

- *Stoppers* son los alumnos que cuando encuentran un problema difícil no son capaces de ver inmediatamente cómo continuar. Se dan por rendidos o bien buscan ayuda sin antes intentar resolver el problema por su cuenta.

- *Movers* son los que continúan hacia delante intentando resolver los problemas desde distintos enfoques. Aunque a veces acaban introduciendo nuevos errores en su código fuente, suele ser de una forma controlada.

- Y *tinkerers* son los estudiantes excesivamente *movers*, es decir, intentan resolver los problemas, pero sin cambiar la forma de atacarlos, simplemente haciendo pequeños cambios en el código fuente con la esperanza de resolver los errores encontrados. Sin embargo, acaban por introducir nuevos errores en el código sin resolver los previos, hasta el punto de hacer mucho más difícil el volver a un estado de compilación sin errores.

Tanto los alumnos *stoppers* como los *tinkerers* tienen estrategias de aprendizaje que dificultan su progreso en la resolución de problemas de programación. Si los docentes pudiesen detectar estos comportamientos de los alumnos de forma sencilla, podrían identificar problemas en las sesiones prácticas de programación de forma temprana, y adaptar las clases reforzando conceptos o enseñando distintas técnicas de depuración del código fuente o de planificación (identificación de qué serie de acciones deben seguir para resolver un problema), mejorando la enseñanza de la programación para este tipo de alumnos.

Con las sesiones prácticas o talleres de programación, los alumnos pueden recibir ayuda y orientación por parte de los profesores, pero al tratarse de sesiones con múltiples alumnos al mismo tiempo, es difícil realizar un seguimiento detallado de cada alumno en cada sesión. Utilizando una herramienta que registre las acciones del alumno durante las sesiones prácticas, el profesor podría revisar de forma sencilla el progreso de cada alumno de forma individualiza. Con esta información se pueden identificar problemas durante las clases y reforzar conceptos en futuras sesiones para corregir esos problemas y facilitar el aprendizaje de la programación.

1.2 Finalidad del Proyecto

La finalidad del proyecto es ayudar al profesorado a realizar un seguimiento individualizado de los alumnos durante las sesiones de programación, bien en tiempo real o bien más tarde, y así detectar de forma temprana problemas de asimilación y aplicación de conceptos

de programación y comportamientos poco eficientes (*stoppers* y *tinkerers*). Una vez identificados los problemas, los profesores pueden adaptar sus clases teóricas y prácticas para hacerlas más efectivas y facilitar a los alumnos el aprendizaje de la programación.

Otros autores proponen unos sistemas de seguimiento distintos. Algunos están enfocados a obtener un listado de los errores más frecuentes que cometen los alumnos, en otros la captura de datos se realiza en lotes, no en tiempo en real y en otras ocasiones utilizan formatos propietarios para almacenar los datos recolectados, disminuyendo las posibilidades de acceder a esos datos desde otros grupos de investigación. En cambio, el proyecto desarrollado proporciona nuevas perspectivas a los docentes con el fin de detectar el comportamiento de los alumnos durante las sesiones prácticas, en tiempo real y en un formato abierto, para mejorar el aprendizaje de la programación.

1.3 Estructura de la documentación

El resto del documento se organiza de la siguiente manera. En la siguiente sección se plantean los objetivos del proyecto así como el escenario de aplicación. La sección 3 recoge un estudio del estado actual de los sistemas de seguimiento de los alumnos o analíticas de aprendizaje. En la sección 4 se presenta en detalle el sistema desarrollado. La sección 5 explica la metodología que se ha seguido durante el trabajo y en la 6 se muestran los resultados obtenidos con este proyecto. Finalmente, la sección 7 concluye el proyecto e indica cuál será el trabajo futuro.

2 Fijación de objetivos

El proyecto está dirigido al aprendizaje de la programación desde el punto de vista del profesorado. El objetivo principal es facilitar a los docentes el seguimiento del aprendizaje de la programación de sus alumnos. Este objetivo principal se puede dividir en varios objetivos más pequeños:

- Asistir a los profesores en la revisión del proceso de programación de múltiples alumnos de forma fácil, rápida e individualizada. El proceso de programación es un proceso dinámico, por lo que es difícil hacer un seguimiento del mismo. Sin embargo, podemos aplicar learning analytics, o analíticas de aprendizaje, para recopilar las interacciones del estudiante con el entorno, medirlas, analizarlas y mostrarlas en un panel de control, donde los profesores puedan ver de forma rápida e individualizada, proyecto a proyecto y en tiempo real, el proceso de programación de cada alumno.

- Ayudar a los profesores en la detección temprana de alumnos con problemas de aprendizaje de la programación como bloqueos (*stoppers*) o bajo rendimiento (*tinkerers*), para poder tomar acciones correctivas lo antes posible.

Además, el proyecto debe cumplir los siguiente objetivos secundarios:

- El establecimiento, captura, almacenamiento, análisis y representación de variables significativas para el seguimiento del comportamiento del alumno durante las sesiones de programación.

- La integración de la herramienta con el entorno de desarrollo del alumno de forma transparente, evitando interferir en su flujo de trabajo cotidiano.

- El uso de un formato abierto para el almacenamiento de los datos recolectados, permitiendo el acceso a otros grupos y comunidades, y promoviendo el uso de herramientas ya existentes.

2.1 Escenario de Aplicación

El proyecto desarrollado está enfocado principalmente al entorno educativo, para el personal docente que imparte clases de programación. Durante las clases prácticas o talleres de programación, el profesor plantea un problema a los alumnos que deben resolver aplicando las estructuras de datos, algoritmos y conceptos aprendidos durante las clases teóricas. Estos talleres se llevan a cabo en salas de ordenadores compartidos con unos 20 alumnos por sesión, utilizando un IDE para programar y un sistema que captura los datos de los alumnos durante la sesión y los envía y almacena en un repositorio central. Por otro lado, una herramienta accede a esos datos almacenados en tiempo real, los procesa, analiza y muestra en un panel de control. Con esta herramienta, los profesores pueden revisar el progreso de los alumnos durante los talleres prácticos de programación, viendo los errores que cometen y si se bloquean en algún punto de la práctica, de forma completamente individualizada. De esta manera se pueden fijar

en los problemas de cada uno de los alumnos de forma rápida, sin perder detalle. Sin la herramienta sería imposible supervisar a todos los alumnos de una misma clase de forma individualizada. Además, al tener la información almacenada en un servidor central, pueden acceder a ella más tarde y ver la progresión de un alumno a lo largo del tiempo en sucesivas prácticas.

Como una ampliación, la herramienta también se puede aplicar en el ámbito profesional, durante las pruebas de selección de candidatos para puestos de desarrollo software. Cuando una empresa solicita una prueba técnica de entrada a un aspirante para un puesto de desarrollo de software, pueden utilizar la herramienta para ver paso a paso cómo el candidato ha desarrollado la prueba, en qué puntos se ha bloqueado, si ha cometido errores, etc. También puede ser útil para comparar a un aspirante con otros.

3 Estado Actual de los Conocimientos Científico-Técnicos

En nuestro proyecto necesitamos realizar un seguimiento del proceso de programación de los alumnos, enviando la información recolectada en tiempo real a un servidor de forma transparente, sin requerir la acción del usuario. Nuestra solución debe ser aplicable a entornos educativos, aunque sería valorable su uso en entornos profesionales también, debe poder procesar los eventos en tiempo real y utilizar un formato abierto para el procesamiento y almacenamiento de los datos.

Para capturar, almacenar y procesar la información de los estudiantes aplicamos técnicas de analíticas de aprendizaje o "Learning Analytics". Son técnicas que consisten en la recopilación de los datos específicos sobre la interacción del estudiante con el entorno para su posterior análisis. Adopta muchos de los métodos usados en "big data" como la minería de datos o visualización de datos, pero se enfoca en el contexto del aprendizaje, en las interacciones de los estudiantes con el contenido y con otros estudiantes. Todo ello con el objetivo de crear modelos predictivos e interpretativos sobre el aprendizaje de los alumnos, mejorar las técnicas de enseñanza, proponer contenidos personalizados a la situación de cada estudiante, detectar patrones de éxito o fallo, etc. [9]. En este proyecto nos centramos en capturar las interacciones de los estudiantes con el contenido: cómo modifican el contenido cuando están programando y cómo corrigen los errores de programación que se encuentran.

Algunos autores proponen sistemas de analíticas de aprendizaje diferentes al nuestro o con unos objetivos distintos, por lo general más enfocados en retroalimentar al usuario en lugar del profesor.

Duval [10] se centra en realizar un seguimiento de todas las acciones del usuario a lo largo de todo el sistema, desde el tiempo que pasa en cada aplicación instalada en el equipo, hasta las páginas web que visita. El objetivo es poder comparar los datos recopilados de cada usuario y ver qué relevancia tienen en el aprendizaje, comparando los datos de unos alumnos con otros. Se centra principalmente en cómo debería visualizarse la información, para lo que propone utilizar un panel de control o *dashboard*. También resalta la necesidad de definir qué información será realmente útil mostrar al usuario para filtrar la abundancia de datos.

Esta propuesta se centra más bien en cómo mostrar la información al usuario (en paneles de control o *dashboards*), sin entrar en detalle en cómo recolectar los datos o procesarlos.

Scheffel et al. [11] crearon una máquina virtual especialmente diseñada para los alumnos que están aprendiendo a programar en C. Incluye un sistema operativo y varias herramientas ya preparadas para ser utilizadas, pero está modificadas para recolectar las acciones del usuario. Además de recopilar la fecha y hora de apertura y cierre de las distintas aplicaciones, también capturan los resultados del compilador (mensajes de error y advertencia), los comandos ejecutados en la consola de comandos, las direcciones visitadas en el navegador web, etc. Todos estos eventos se almacenan en un formato propio dentro de una carpeta de usuario en la máquina virtual y, más tarde, cuando el alumno realiza una subida de su trabajo práctico, el sistema incluye los datos recopilados hasta la fecha. En ese momento, el servidor

recibe los eventos recopilados, los transforma siguiendo la especificación CAM (ver 9.2.2) y los almacena en una base de datos relacional.

En la fase de análisis, buscan las secuencias de eventos más frecuentes para así detectar tendencias en el comportamiento de los alumnos. Primero, detectan los errores de programación más comunes, siendo "variable sin declarar" y "dos o más tipos en la declaración de la variable" los dos errores más frecuentes del listado. Y a continuación, detectan qué acciones realizan los alumnos después de obtener uno de esos dos errores. Los resultados de estos análisis son mostrados a los profesores a través de una herramienta gráfica que permite visualizar de forma rápida qué pasos siguen sus alumnos después de encontrarse con un error de programación. Además, también pueden seguir el flujo de acciones de los alumnos a lo largo del sistema.

Esta propuesta cumple de forma parcial el objetivo de facilitar al personal docente la revisión de talleres de programación, pero el análisis de datos no se realiza en tiempo real. Es necesario que sea el propio alumno el que suba los datos recolectados de forma manual. Además, el análisis de los datos está más enfocado a detectar los errores más comunes y las acciones siguientes, no ha detectar problemas o bloqueos de los alumnos.

Otros autores [12], [13] utilizan editores web de código fuente (CodeWrite y Gauntlet IDE, respectivamente) para capturar los errores de programación que generan los estudiantes. Después, aplican estadísticas sobre los datos recopilados para identificar los errores más comunes. En [13] son capaces de predecir 5 de los 10 errores más comunes, pero en [12] van más allá, calculando el tiempo que tardan los alumnos en resolver ciertos errores. Sin embargo, concluyen que no hay una relación directa entre el nivel de los estudiantes con el tiempo necesario para resolver los 3 errores más comunes: "Falta ;", "No coinciden los tipos" y "No se puede resolver el identificador".

Estas propuestas se centran en recopilar los errores de los alumnos y visualizar cuáles son los más comunes. No analizan el comportamiento del alumno para resolver los errores a lo largo de la sesión, sino únicamente tienen en cuenta el tiempo necesario para resolver un tipo de error.

Jadud [5], [14] muestra una herramienta para observar el comportamiento de los alumnos de programación entre una compilación y otra. Utiliza la minería de datos para recopilar el código fuente de los alumnos cada vez que compilan un fichero y después aplica técnicas estadísticas para detectar patrones de comportamiento. Se centra exclusivamente en analizar qué hacen los alumnos inexpertos cuando se encuentran con un error sintáctico en lo que él denomina como ciclos 'editar-compilar'. Ciclos en los que los alumnos editan el código fuente, lo compilan y descubren algún tipo de error de compilación. Entonces, vuelven a editar el código fuente y a compilarlo de nuevo hasta resolver por completo el error, generar un error distinto o encontrarse con el mismo error anterior. Los alumnos utilizan el IDE BlueJ, un entorno pensado para ser usado por programadores inexpertos. Una de las características de BlueJ es que el compilador sólo devuelve un error en cada compilación, aunque existan más en el código fuente. De esta forma, los alumnos no se sienten abrumados por un gran número de errores en cada compilación y no tienen que decidir qué error atacar primero.

Una vez recolectados los datos, Jadud aplica un algoritmo estadístico a las compilaciones para obtener un cociente de error, EQ, de la sesión completa. Este cociente de error varía de 0.0 a 1.0 e indica si el alumno ha tenido excesivas dificultades o no durante la sesión. Sin embargo, más adelante demuestra que el EQ no se puede utilizar como un medidor

del rendimiento del alumno en programación, ya que la relación entre el EQ y las notas finales de la asignatura es muy pobre. Esto lo achaca a un conjunto de datos incompletos. Sin embargo, Jadud también elabora una herramienta para visualizar de forma fácil y rápida el progreso de un alumno en una sesión práctica, mostrando una tabla con el número de compilación, el tipo de error encontrado (si es que existe), el intervalo de tiempo que ha transcurrido entre una compilación y la siguiente, la diferencia de caracteres introducidos y dónde se han realizado las modificaciones. Con esta herramienta de visualización, Jadud afirma [5] que los profesores podrían revisar un gran número de sesiones prácticas de programación de forma frecuente, incluso de cientos de alumnos, y ver si están teniendo problemas graves.

Esta propuesta cumple el objetivo de facilitar al personal docente la revisión de talleres de programación con varios alumnos de una forma rápida, visual y sencilla. Sin embargo, utiliza un IDE para entornos educativos en lugar de un IDE profesional, descartando su uso fuera de cursos iniciales de enseñanza de la programación. Además, para la recolección de datos se utiliza un formato propio no estándar, dificultando la apertura de los datos capturados a otros sistemas de recolección y/o procesamiento de datos.

Cardell-Oliver [7] plantea un sistema distinto. No captura las acciones de los estudiantes durante los talleres de programación, sino que analiza el código fuente enviado al finalizar la práctica. Propone comparar el tamaño del código fuente (número de líneas), la precisión (número de tests unitarios aprobados) y la eficiencia (tiempo de ejecución) con respecto a la solución canónica o esperada del problema, para clasificar a los estudiantes como *stopper* o *tinkerer*. La solución de un *stopper* tendrá un tamaño menor que la solución correcta y fallará la mayoría de tests unitarios (baja precisión). En cambio, un *tinkerer* escribe más código (más líneas) y es más ineficiente, tarda más tiempo en finalizar la ejecución.

Sin embargo, esta propuesta requiere disponer de la solución correcta al problema previamente para poder realizar una comparación del código fuente de los alumnos con respecto a ella. Además, no se puede conocer el tipo de alumno hasta el fin de la práctica, cuando ya tiene el código fuente completo y ha implementado los tests unitarios. Tampoco tiene en cuenta el tiempo, y es que el alumno puede presentar varios comportamientos durante una misma práctica, mostrando patrones *stopper* durante algunos momentos y más tarde *tinkerer*. Este sistema no permite evaluar la evolución del alumno en tiempo real, únicamente al final de la práctica.

Finalmente, Fernández-Medina y Pérez-Pérez crean un modelo de asistencia en la enseñanza de la programación llamado COLMENA [15], [16]. Se trata de un modelo diseñado para capturar y reflejar el progreso de aprendizaje de los alumnos. Se compone de cuatro capas, desde la extracción de datos utilizando técnicas de minería de datos a la visualización de la información ya procesada a modo de informe. La flexibilidad del modelo permite ser utilizado en una amplia variedad de entornos de aprendizaje. En [15] aplican el modelo en la enseñanza de la programación a alumnos de Ingeniería Informática. El objetivo es obtener un mejor conocimiento de los errores de programación más comunes que cometen durante las sesiones prácticas. Para ello, en la capa de extracción de datos obtienen los mensajes de error y advertencia del compilador directamente desde el IDE Eclipse, para después procesar esa información, calculando el total y la frecuencia de los errores encontrados en el código fuente de los alumnos. Finalmente, muestran los resultados en tiempo real en un panel adicional dentro del IDE para que los alumnos tengan mayor conciencia de sus propios errores. Los profesores también pueden acceder a esa información, permitiéndoles entender mejor las necesidades de aprendizaje de sus alumnos y tomar mejores decisiones. En [16] aprovechan la información

recopilada para clasificar a los alumnos dentro del grupo y adaptar los paneles y herramientas de la interfaz de usuario de Eclipse a las necesidades individuales de cada alumno.

Su propuesta utiliza un formato propio para la recolección de los datos, pero al ser flexible se puede extender y utilizar un formato estándar para el almacenamiento de los datos.

Debido a la flexibilidad del modelo de COLMENA, en nuestro proyecto extendemos este sistema. Modificamos el *plug-in* para Eclipse con el objetivo de recolectar otros datos distintos del alumno y almacenarlos en un repositorio remoto siguiendo un formato estándar en lugar de uno propio. También aplicamos analíticas de aprendizaje distintas para identificar comportamientos típicos de alumnos *stopper*, *mover* o *tinkerer*. Para ello, analizaremos el tamaño del código fuente, como propone Cardell-Oliver [7], pero añadimos la variable del tiempo para realizar la evaluación en tiempo real. Finalmente, mostramos la información de forma visual en un panel de control o *dashboard* fuera del IDE, orientado como una herramienta para el equipo docente.

En la sección 4 hay una descripción completa del sistema desarrollado.

4 Descripción del Sistema

El sistema desarrollado se compone de un almacén de datos y cuatro capas: la capa de extracción de datos, la capa de procesamiento de datos, la capa de análisis de datos y la capa de visualización. Para la capa de extracción de datos se ha extendido el *plug-in* de recolección de datos de COLMENA [15], [16], un *plug-in* para el entorno de desarrollo integrado Eclipse que captura información de la actividad del alumno cuando compila el proyecto y la envía a un almacén de datos central siguiendo un formato abierto. Para las capas de procesamiento, análisis y visualización de datos se ha creado un *dashboard* o panel de control web que aplica analíticas de aprendizaje a la información almacenada en el servidor central y muestra los resultados ya procesados. Desde el *dashboard* los profesores pueden visualizar de forma rápida y sencilla información individualizada sobre los alumnos, proyecto a proyecto y actualizada en tiempo real, con el objetivo de facilitarles la tarea de identificar comportamientos típicos de alumnos *stopper*, *mover* o *tinkerer*.

En la Figura 1 se puede ver de forma simplificada la arquitectura del proyecto y en las siguientes secciones se detallan estos componentes en profundidad.

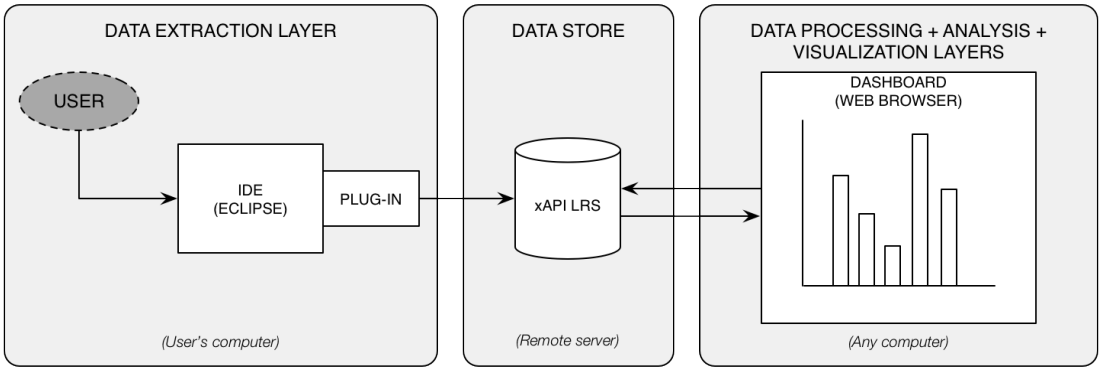


Figura 1 Diagrama del proyecto

4.1 Capa de extracción de datos

Esta capa se encarga de recolectar datos con el objetivo de observar el comportamiento de los alumnos mientras programan. A través del IDE Eclipse, tenemos acceso a la información que devuelve el compilador, y así capturamos los errores que cometen los estudiantes mientras programan, además de recopilar información sobre el contexto actual. Cada vez que el alumno compila el código fuente, capturamos los siguientes datos de la compilación:

- ID de usuario del alumno. Una cadena de caracteres que identifique de forma única al alumno.
- Nombre del proyecto. El nombre completo del proyecto donde se encuentra el archivo compilado.
- Nombre del archivo compilado.

- Código fuente completo del archivo. Para poder hacer comparaciones con respecto a compilaciones previas y obtener la diferencia de caracteres y líneas introducidos entre compilaciones consecutivas de cada fichero.

- Tipo de error. El tipo de error que devuelve el compilador Java de las JDT (Java Development Tools) de Eclipse. Puede ser Campo, Sintaxis, Import, Tipo, Método, Constructor, Interno, Javadoc o ninguno en caso de compilación correcta.

- Timestamp. Un sello de tiempo de la compilación actual. Se utilizará para agrupar los archivos por compilación y también para calcular la diferencia de tiempo entre compilaciones consecutivas.

4.1.1 Plug-in

Para capturar esta información, hemos adaptado el *plug-in* de recolección de datos de COLMENA [15], [16] para Eclipse. El *plug-in* forma parte de un sistema distribuido. Se instala en el Eclipse de todos los usuarios y después de cada compilación, recorre todos los ficheros modificados recolectando mensajes de error del compilación, además de otra información. Después, los datos capturados se almacenan en un archivo local o en una base de datos local o remota utilizando un formato propio. La recolección y envío de datos se hace de forma transparente para el usuario, pero el *plug-in* permite detener la captura de información a través de una opción en el menú.

COLMENA xAPI es la versión modificada de COLMENA que no captura la misma información, sino los datos descritos anteriormente (ID del usuario, proyecto, código fuente…), y los envía a un servidor remoto, accesible desde Internet a través de HTTP, siguiendo el estándar xAPI (ver sección 4.2). Mantiene las mismas opciones del menú (ver Figura 2) para detener la captura de datos, por ejemplo, pero modifica el panel de opciones de las preferencias de Eclipse para permitir al usuario introducir la URL y las credenciales del servidor remoto (ver Figura 3).

Cada vez que el usuario compila el proyecto, el *plug-in* recibe el evento y recorre todos los archivos modificados recogiendo los errores encontrados, si es que existen, además del resto de datos descritos anteriormente. Para cada error, o no error, se crea una sentencia en formato xAPI que se envía directamente al almacén de datos central donde queda almacenada.

El *plug-in* requiere conexión a Internet para enviar la información al servidor. Si no es capaz de acceder al servidor a través de Internet de forma inmediata, la información capturada se pierde. En futuras versiones del *plug-in*, esta limitación se abordará utilizando un servidor local intermedio que almacene los datos antes de enviarlos al servidor remoto, central y accesible desde Internet.

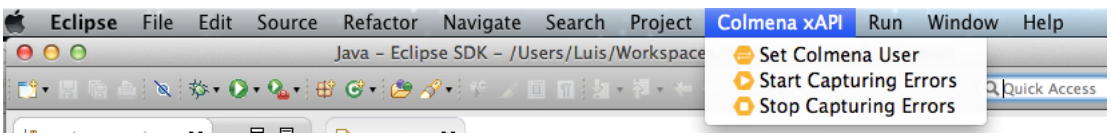


Figura 2 Menú COLMENA xAPI

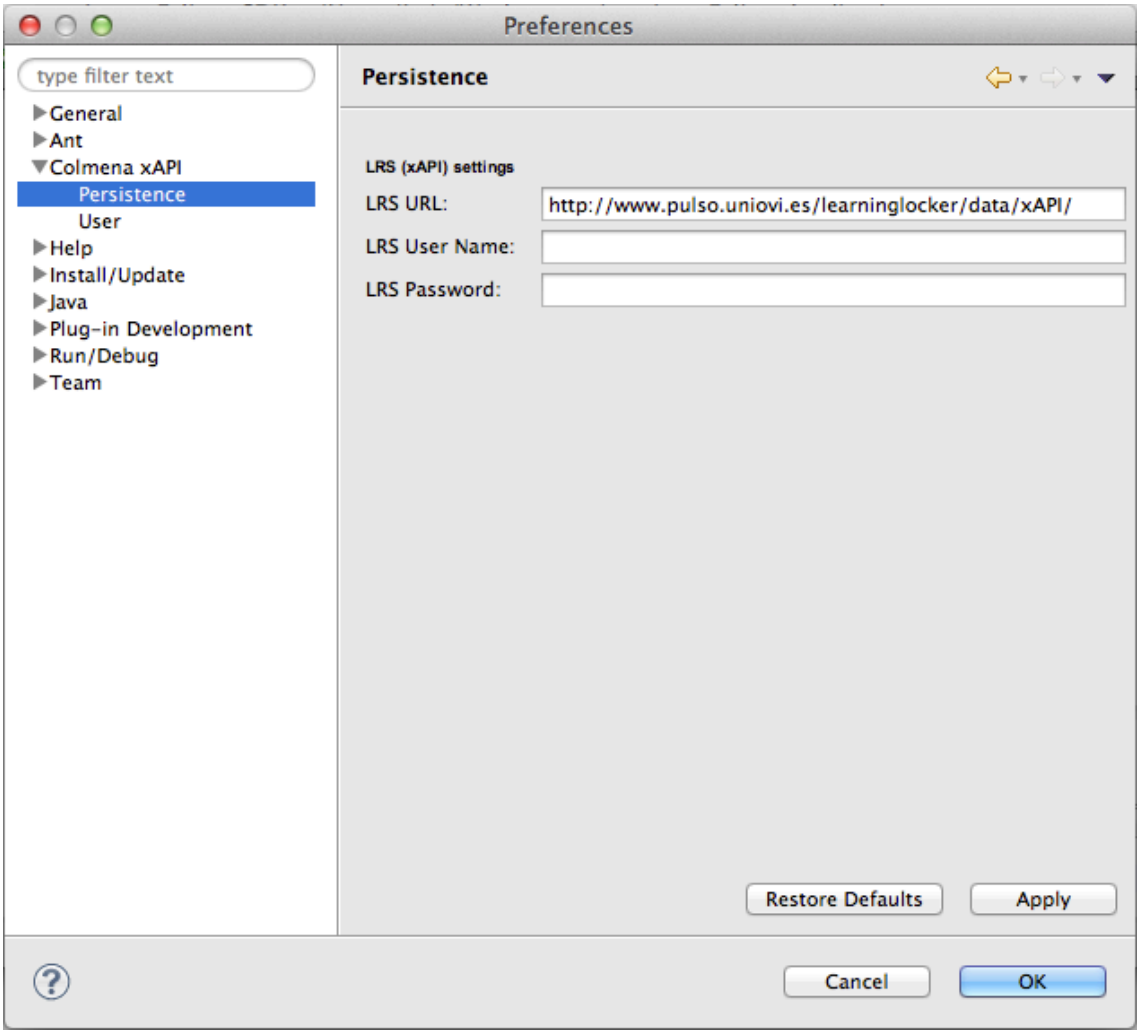


Figura 3 Opciones COLMENA xAPI

4.2 Almacén de datos central

Los datos recogidos por el *plug-in* de Eclipse se representan, envían y almacenan siguiendo el estándar xAPI [17]. Las principales ventajas de xAPI son su extensibilidad, la apertura de los datos y la libertad de plataforma y conexión. El estándar define un formato de datos y unas especificaciones de servidor, dejando abiertas las implementaciones. El formato que se utiliza para almacenar la información, Activity Streams[18], permite guardar casi cualquier actividad en JSON y se puede extender para soportar nuevos modelos de datos sin romper la compatibilidad. Los sistemas de almacenamiento son distribuidos y abiertos, permitiendo compartir los datos con otros sistemas que también implementan el mismo formato. Además, cualquier dispositivo puede enviar información, ya que se basa en peticiones REST, e incluso no es necesaria una conectividad permanente, haciéndolo muy apropiado para entornos móviles. El estándar xAPI es adecuado para este proyecto, porque define cómo representar las acciones, o experiencias, de los estudiantes (por ejemplo, usuario A compila el fichero F) en un formato extensible y mantiene los datos abiertos a otros sistemas.

Otro posible formato para representar las acciones de los estudiantes es el schema CAM [19]. En cambio, CAM trata a los documentos sobre los que trabaja el usuario de una forma muy estática, esperando que apenas cambien con el tiempo. Este modelo no es adecuado para

nuestro proyecto, ya que los documentos cambian en cada evento de compilación. Además, la documentación disponible sobre CAM es escasa, está poco organizada y es confusa. Por ejemplo, la propia estructura del schema CAM parece que aún no es definitiva y sigue en desarrollo.

En el anexo 9.2 se pueden ver las características de cada formato en detalle.

El estándar xAPI define un almacén de datos, un sitio donde guardar las sentencias, llamado Learning Record Store (LRS). Es un servidor accesible desde Internet que permite recibir sentencias xAPI [17] desde cualquier cliente compatible con el estándar y con las credenciales correctas. Un LRS es un repositorio de registros de aprendizaje o experiencias (sentencias xAPI) que puede ser leído por otros LRS, herramientas de informes o LMSs, y que puede ser independiente o estar embebido dentro de un LMS. Se trata de un sistema distribuido, permitiendo enviar la información de un LRS a otro, sea central o no, por lo que podemos compartir los datos almacenados con otros grupos de investigación de forma sencilla.

Existen varias implementaciones de un LRS. En nuestro proyecto utilizamos Learning Locker[1], una implementación de código libre con algunas de las siguientes características:

- Posibilidad de configurar múltiples LRS en una única instancia. Por ejemplo, un LRS para cada asignatura.
- Dispone de una API adicional, más potente que la estándar de xAPI, para hacer consultas y generar reportes más completos.

También existen otras implementaciones de un LRS como Wax LRS[2] o GrassBlade[3] pero son propietarias, utilizan una comercialización Software-as-a-Service o licencias.

A continuación se muestra un ejemplo de sentencia xAPI enviada por el *plug-in* y almacenada en el LRS. Representa un evento "usuario ha compilado fichero" siguiendo el modelo Actor-Verbo-Objeto de xAPI donde el nombre del usuario ("actor") es UO189054 y el "objeto" es un fichero "Main.java". La información que no encaja dentro del modelo del estándar xAPI se almacena utilizando los campos "extensiones". Por ejemplo, el código fuente completo de un archivo y el nombre del proyecto al que pertenece se almacenan dentro de extensiones de "objeto". Además, el resultado de la acción ("result") indica si la compilación ha sido exitosa ("result" igual a "true") o no. Si se hubiese encontrado algún error de compilación, se almacenaría en el campo "extensions" de "result".


```
{
    "version": "1.0.0",
    "id": "2fd0c2c2-d000-4682-97d8-407ca2c3af0b",
    "timestamp": "2015-05-25T17:20:19Z",
    "verb": {
        "id": "http://adlnet.gov/expapi/verbs/compiled",
        "display": {
            "en-US": "compiled"
        }
    },
    "actor": {
        "objectType": "Agent",
        "name": "UO189054",
```


[1] http://learninglocker.net/

[2] https://www.waxlrs.com/

[3] http://www.nextsoftwaresolutions.com/grassblade-lrs-experience-api/

```
                "mbox": "mailto:UO189054@uniovi.es"
            },
            "object": {
                "objectType": "Activity",
                "id":
"http://www.pulso.uniovi.es/learninglocker/data/xAPI/statements/UO189054-c17e603c-c553-
478a-b0ea-d7bbf2bc6880-1432574419734",
                "definition": {
                    "extensions": {
                        "http://www.pulso.uniovi.es/object/project": "fibonacci",
                        "http://www.pulso.uniovi.es/object/sourceCode": "\npublic class
Main {\n\n\t\n\tpublic static void main(String[] args) {\n\t\tint x=1;\n\t\tint
y=1;\n\t\tSystem.out.print(x + \", \" + y + \", \");\t\n\t\t\n\t\tfor(int n=1;n<19;
n++){\n\t\t\tint z=x+y;\n\t\t\tx=y;\n\t\t\ty=z;\n\t\t\tint total = 0;\n\t\t\ttotal=total
+ z;\n\t\t\t\n\t\t\tSystem.out.print(z + \", \");\n\t\t}\n\t\t\n\t\tint average = total
+2;\n\t\tSystem.out.print(\"The average is\" + average);\n\t\t\n\t}\n\n}\n"
                    },
                    "name": {
                        "en": "Main.java"
                    }
                }
            },
            "result": {
                "success": false,
                "extensions": {
                    "http://www.pulso.uniovi.es/result/errorType": "Field"
                }
            },
            "authority": {
                "objectType": "Agent",
                "name": "",
                "mbox": "mailto:admin@colmena.uniovi.es"
            },
            "stored": "2015-05-25T17:20:06.320500+00:00"
        }
```


4.3 Capa de procesamiento de datos

En esta capa se procesan los datos almacenados en el LRS para facilitar posteriormente la labor de análisis. Para el proyecto se ha desarrollado una herramienta web que se encarga de implementar esta capa y las de análisis y visualización en JavaScript, recuperando los datos a partir del LRS, procesándolos y finalmente generando una visualización.

El procesamiento de los datos se realiza siguiendo el diagrama de la Figura 4. Primero agrupamos los datos de acuerdo al estudiante y al proyecto al que pertenecen, juntando todos los eventos que se han producido en un proyecto concreto de un alumno. Después agrupamos los eventos por fecha de compilación y, ya que en una misma compilación se pueden encontrar varios errores en un mismo fichero, agrupamos por fichero después. Al final tenemos una captura del estado de cada fichero en cada compilación, incluyendo el código fuente y la lista de errores.

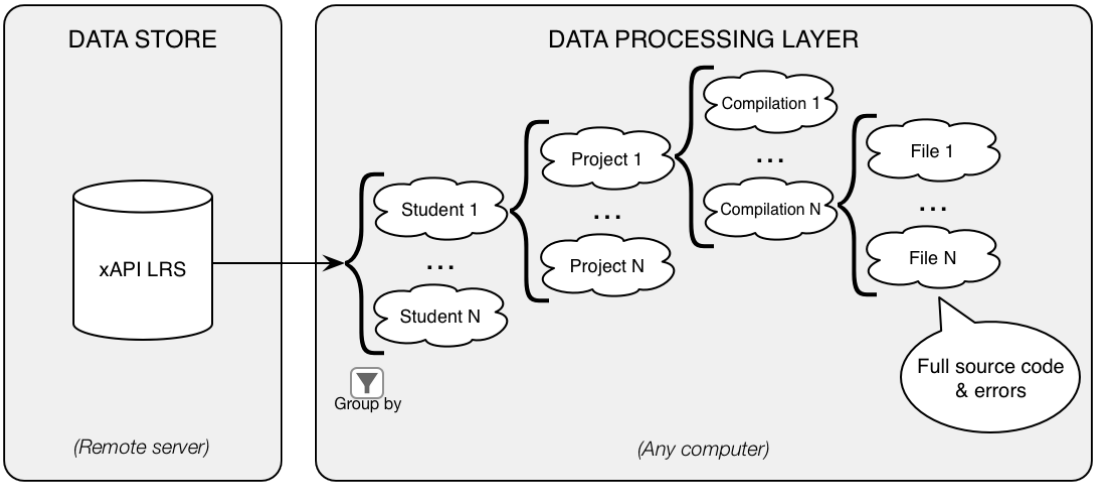


Figura 4 Procesamiento de datos

4.4 Capa de análisis de datos

En esta capa se analizan los datos ya procesados para ayudar al personal docente a detectar alumnos con patrones de comportamiento de tipo *stopper* o *tinkerer* [5]–[8]. A diferencia de los alumnos *mover*, estos estudiantes tienen estrategias de aprendizaje que dificultan su progreso en la resolución de problemas de programación. Utilizando el tamaño del código fuente, una de las métricas propuestas por Cardell-Oliver [7], y añadiendo la variable del tiempo transcurrido, podemos analizar el ritmo de escritura de los alumnos para observar patrones *stopper* o *tinkerer*.

Ambos patrones tienen en común un ritmo de escritura característico para cada grupo. Los alumnos *stopper* introducirán menos caracteres por minuto que los *movers*, ya que cuando encuentran un problema difícil no son capaces de ver cómo continuar. Se dan por rendidos o buscan ayuda, por lo que introducen pocos caracteres o tardan más tiempo en compilar. Por otro lado, los alumnos *tinkerer* son excesivamente *movers*, realizando más cambios en el código fuente para intentar resolver los errores encontrados. Para este patrón, el número de caracteres introducidos por minuto, o ritmo de escritura, es alto.

Para obtener el ritmo de escritura aplicamos la Ecuación 1. Calculamos la diferencia de caracteres introducidos en cada fichero en una compilación con respecto a la compilación anterior para obtener el incremento de caracteres. Después, dividimos el valor absoluto del incremento de caracteres entre el tiempo transcurrido entre una compilación y otra para obtener, finalmente, el ritmo de escritura del alumno entre dos compilaciones consecutivas.

$$Typing\ rate_{(compilation\ n)} = \frac{\sum_{files} \left| chars(fileA_{(n)}) - chars(fileA_{(n-1)}) \right|}{timestamp_{(n)} - timestamp_{(n-1)}}$$

Ecuación 1 Ritmo de escritura por compilación

Para detectar patrones *stopper* o *tinkerer* en un alumno necesitamos comparar su ritmo de escritura con respecto al ritmo medio de un *mover*. Calculamos ese ritmo medio de *mover* a partir de los datos de los demás alumnos, incluyendo *stoppers*, *movers* y *tinkerers*, y haciendo

una media artimética. Sin embargo, si no tenemos muchos datos y la mayoría de los alumnos de una clase son *stopper* o *tinkerer*, el ritmo medio de *mover* que hemos calculado no será preciso. Por lo tanto, dejamos que sean los profesores los que reflexionen sobre los datos, aplicando su experiencia en la materia y utilizando el ritmo de escritura como orientación. Para ello, mostraremos los datos de una forma visual en un panel de control o *dashboard* haciendo que sean fácilmente analizables.

4.5 Capa de visualización de datos

En la capa de visualización se exportan los datos ya procesados en un *dashboard* o panel de control que facilita la revisión y exploración de los mismos de una forma más rápida. El *dashboard* se divide en dos páginas web:

- Un índice o listado de todas las sesiones de los alumnos, agrupadas en proyectos individuales.
- Una página con información detallada de la sesión o proyecto de un alumno.

El listado de las sesiones de los alumnos sirve como página de entrada o de referencia del *dashboard*, mostrando todas las sesiones almacenadas hasta la fecha en el LRS y agrupándolas por proyecto y alumno. Como se ve en la Figura 15, en la parte superior se encuentran los campos donde podemos introducir nuestras credenciales para obtener acceso a la información del LRS y, justo debajo, se muestra una tabla paginada y ordenable con el listado de las sesiones de los alumnos.

Pulsando sobre un proyecto se accede a la página con información detallada del mismo (ver Figura 16). A través de esta página el profesor puede acceder en tiempo real a los eventos de compilación del alumno para revisar rápidamente sesiones de programación de forma individualizada.

El elemento principal de este *dashboard* es la gráfica interactiva que encabeza la tabla de compilaciones realizadas por el alumno. Esta gráfica es interactiva, permite mostrar y ocultar series de valores. En ella se muestran las siguientes series:

- El ritmo de escritura del estudiante en cada compilación.
- El número de errores en cada compilación.
- El ritmo medio del estudiante durante la sesión o proyecto actual.
- El ritmo medio global de todos los alumnos a lo largo de todos los proyectos.

Además, en el subtítulo de la gráfica se incluye un resumen del proyecto con el ritmo de escritura medio, el número de compilaciones total y el tiempo total que ha transcurrido desde la primera compilación hasta la última.

En la Figura 5 se ve la gráfica de un alumno durante un proyecto. El ritmo de la primera compilación es muy alto porque aún no hay una compilación previa para calcular el tiempo transcurrido, por lo que utilizamos un minuto exacto para calcular el ritmo de escritura de la primera compilación. A simple vista vemos que el alumno tiene un ritmo de escritura bajo, llegando incluso a compilaciones con un ritmo de 0 caracteres por minuto. Además, su ritmo

medio está por debajo del ritmo medio global, por lo que podemos pensar que este alumno es de tipo *stopper*. Sin embargo, fijándonos en los errores y el ritmo durante las primeras 10 compilaciones, parece que esos momentos tiene un ritmo más cercano a la media global, pero también introduce bastantes errores, llegando a obtener 5 compilaciones consecutivas con hasta 3 errores. En ese tramo desde la compilación 5 hasta la 10, el estudiante muestra patrones de comportamiento de tipo *tinkerer*.

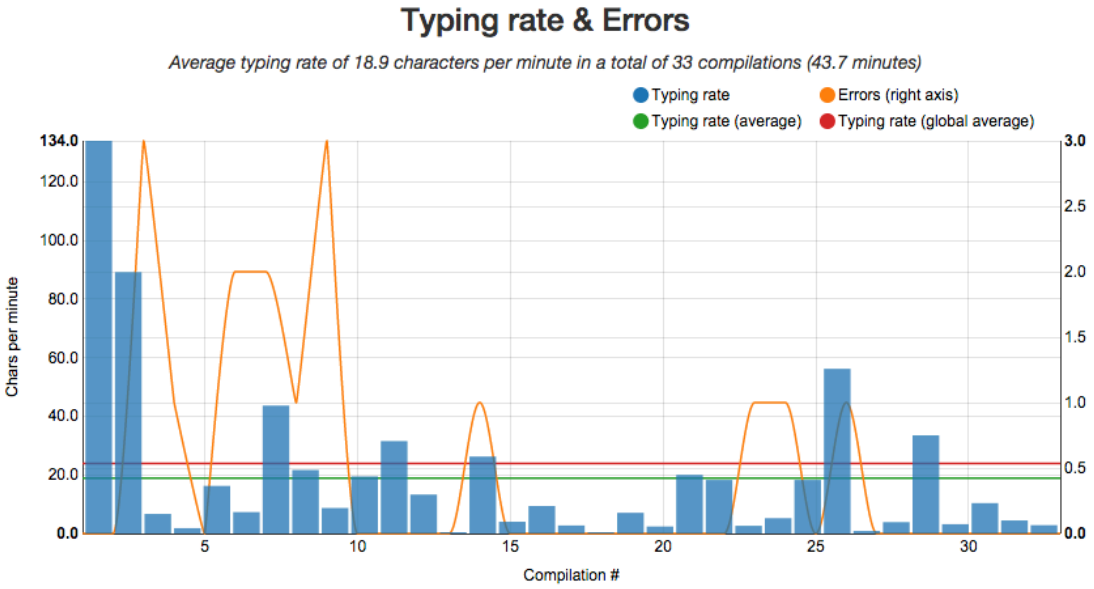


Figura 5 Gráfica de un proyecto

Si nos fijamos en el código fuente de alguna de esas compilaciones, vemos que efectivamente el alumno no comprende aún la sintaxis del lenguaje e intenta corregir los errores mediante pequeños cambios en el código fuente, pero acaba introduciendo nuevos errores (*tinkering*).

Para ver el código fuente, utilizamos la tabla situada debajo de la gráfica (ver Figura 6). Es una tabla paginable y ordenable que muestra más información que la gráfica sobre las compilaciones realizadas en el proyecto actual. Cada fila de la tabla incluye:

- El número de compilación. Se utiliza para agrupar eventos de compilación con el mismo timestamp.
- El número de segundos transcurridos desde la compilación anterior. La cantidad de tiempo transcurrido entre compilaciones consecutivas.
- El ritmo de escritura de la compilación. El número de caracteres por minuto introducidos por el estudiante en la compilación actual.
- El número y tipo de errores encontrados en cada fichero compilado. Agrupa todos los errores encontrados en un fichero en la compilación actual.
- El número de caracteres y líneas introducidos o borrados por los alumnos en esa compilación. Se utiliza para ver de forma rápida si el estudiante tiene problemas continuamente en el mismo trozo de código. Por ejemplo, viendo si siempre

trabaja sobre la misma línea del fichero, es decir, ni inserta ni borra líneas durante múltiples compilaciones consecutivas con errores.

- El nombre del fichero.
- Un botón para abrir un panel que revela las diferencias en el código fuente del mismo fichero con respecto a su compilación previa. En la parte izquierda se muestra el código anterior y en la derecha el código de la compilación actual. Se utiliza para ver de forma rápida qué cambios en el código fuente ha hecho el alumno y dónde.

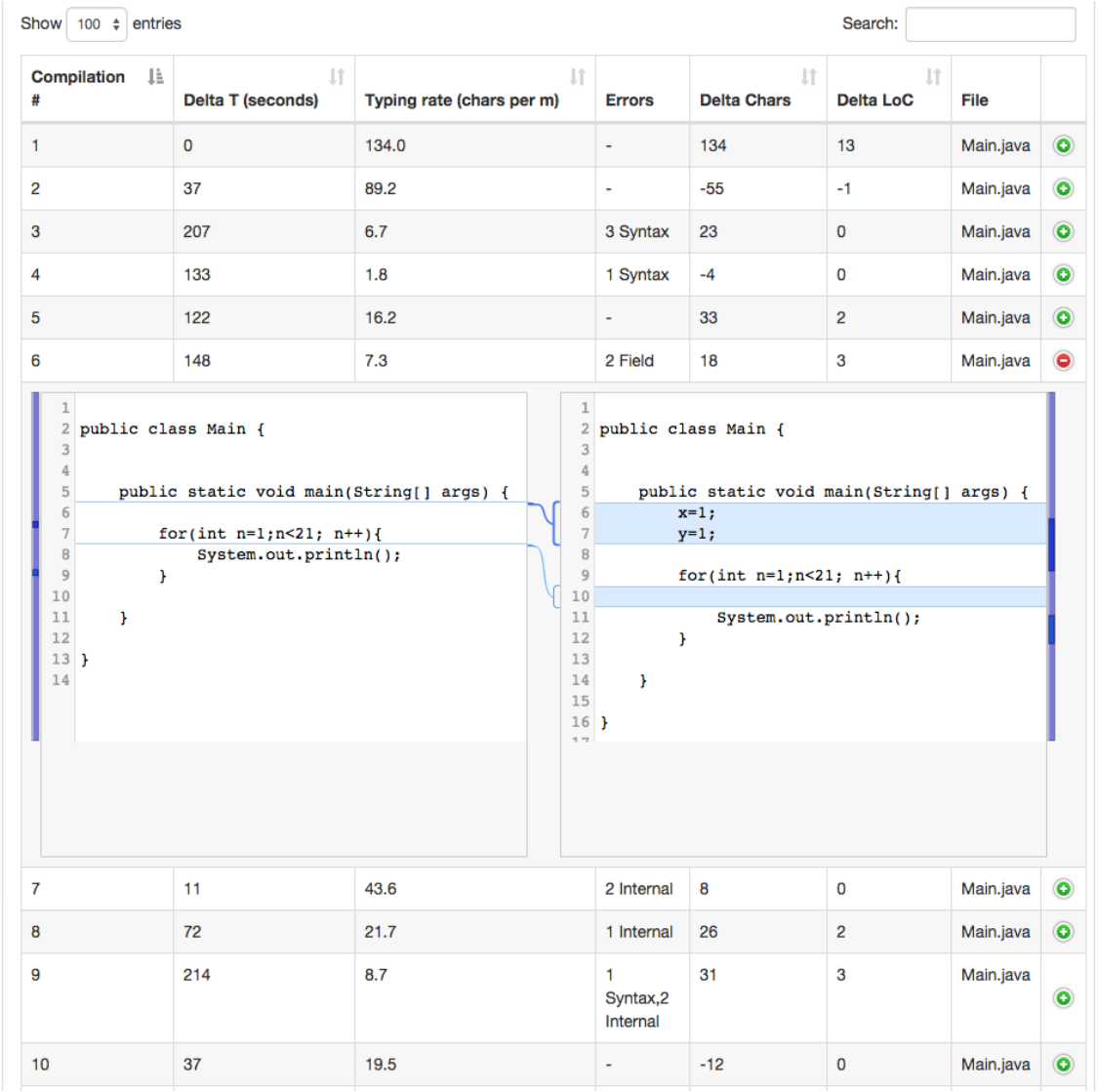


Figura 6 Tabla de compilaciones

Con todos los componentes de este *dashboard* (gráfica interactiva, tabla con errores y comparador de código fuente), los revisores pueden detectar comportamientos *stopper* o *tinkerer*. Un ritmo de escritura alto y muchos errores en compilaciones consecutivas se puede considerar un comportamiento *tinkerer*. Y viceversa, se pueden detectar comportamientos *stopper* cuando el alumno realiza pocas compilaciones, además de tener un ritmo de escritura bajo y tener errores en esas compilaciones con ritmo bajo.

5 Metodología de Trabajo

Nuestra metodología de trabajo se basa en el trabajo previo de Fernández-Medina y Pérez-Pérez [15], [16], en el que definen COLMENA, un modelo de asistencia en la enseñanza de la programación diseñado para capturar y reflejar el progreso de aprendizaje de los alumnos. En nuestro proyecto primero adaptamos COLMENA para recolectar otros datos a través del *plug-in* de Eclipse y enviarlos en formato xAPI a un LRS. Después, hemos construido un *dashboard* en JavaScript que se encarga de procesar, analizar y visualizar esos datos recopilados con el objetivo de facilitar al personal docente la revisión de sesiones de programación de forma remota, individualizada y rápida. Finalmente, se realizaron varias pruebas con algunos usuarios para capturar sus resultados.

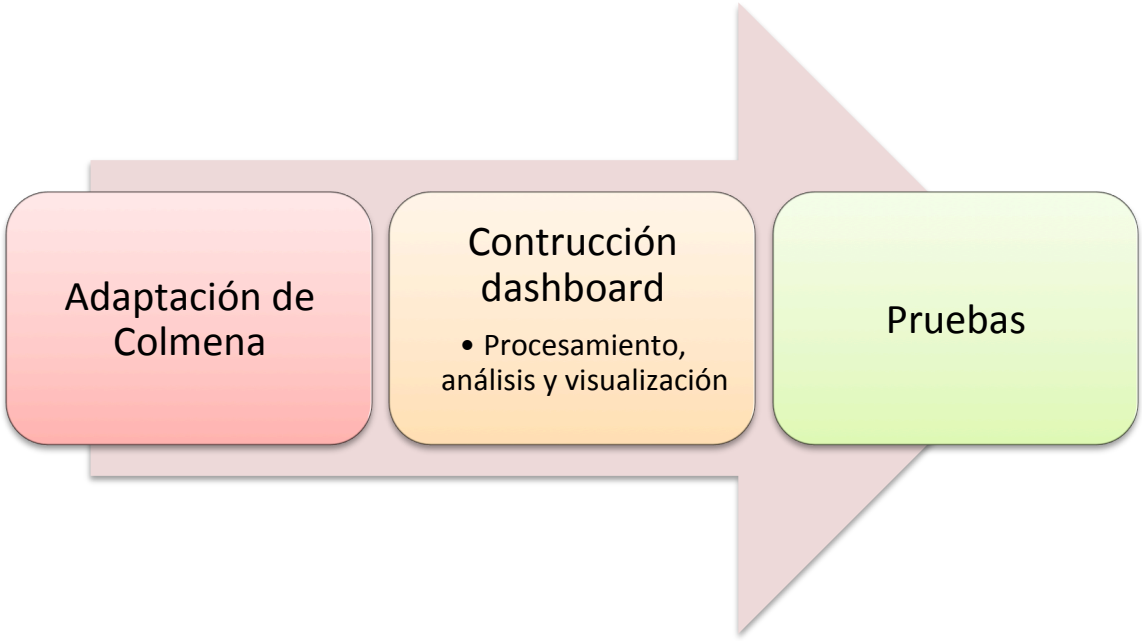


Figura 7 Diagrama metodología de trabajo

Durante el proceso de pruebas, algunos usuarios con muy pocos conocimientos de programación y de Java han realizado un ejercicio sencillo de programación y se ha registrado, mediante COLMENA xAPI, los eventos de compilación así como una captura del estado actual del proyecto completo. Después se han procesado, analizado y visualizado los resultados. A continuación se indica el enunciado del ejercicio propuesto:

> Exercise Fibonacci (Loop): Write a program called Fibonacci to display the first 20 Fibonacci numbers F(n), where F(n)=F(n–1)+F(n–2) and F(1)=F(2)=1. Also compute their average. The output shall look like:
>
> "1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765
>
> The average is 885.5"

6 Resultados Obtenidos

Tras recolectar los datos de los alumnos en cada evento de compilación, se ha llevado a cabo un procesamiento, análisis y, finalmente, visualización de los mismos a través del *dashboard* construido para este proyecto. A continuación se muestra la gráfica de un alumno que presenta los dos tipos de patrones de comportamiento ineficientes, *stopper* y *tinkerer*, en diferentes tramos de la prueba.

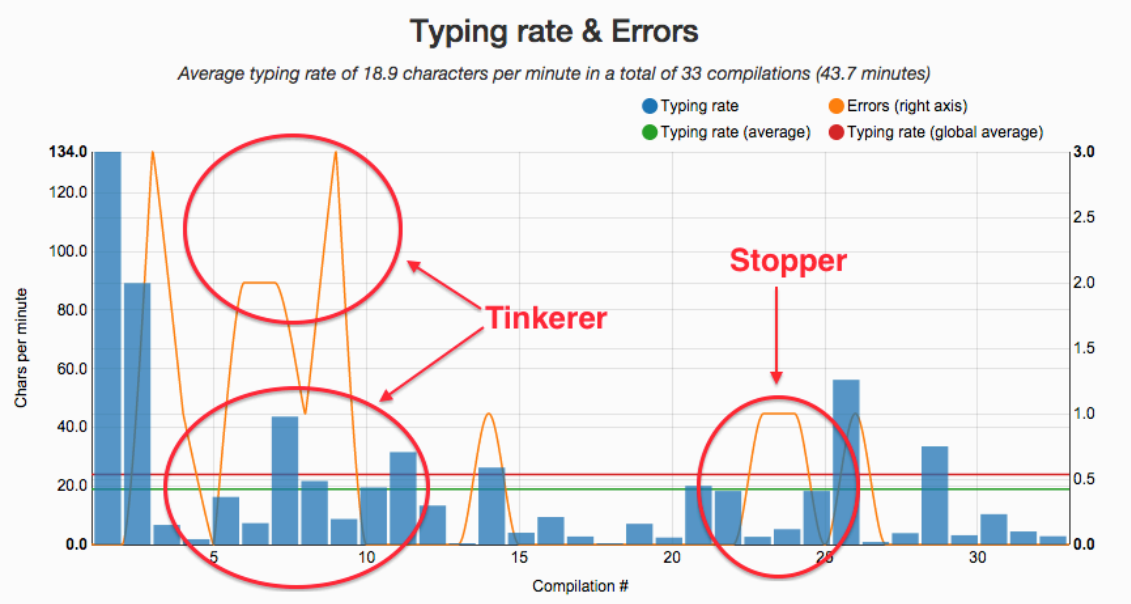


Figura 8 Gráfica *tinkerer* y *stopper*

Vemos que este alumno presenta un comportamiento *tinkerer* entre las compilaciones número 5 y 10, introduciendo código fuente de forma más rápida y generando errores de programación. Además, también presenta un patrón de comportamiento de tipo *stopper* entre las compilaciones 23, 24 y 25, donde tiene un ritmo de escritura menor al de su media, introduciendo errores también y pasando bastante tiempo entre compilaciones consecutivas. De hecho, observando la tabla debajo de la gráfica como se ve en la Figura 9, se puede apreciar que han pasado 274 segundos, más de 4 minutos y medio, desde la compilación 22 hasta la compilación 23. Es el mayor espacio de tiempo que se ha producido entre dos compilaciones consecutivas en todo el proyecto.

21	21	20.0	-	7	0	Main.java	
22	23	18.3	-	7	0	Main.java	
23	274	2.6	1 Internal	12	1	Main.java	
24	92	5.2	1 Internal	8	0	Main.java	
25	46	18.3	-	14	1	Main.java	
26	79	56.2	1 Field	74	2	Main.java	
27	209	0.9	-	3	1	Main.java	

Figura 9 Tabla *stopper*

7 Conclusiones y Trabajo Futuro

En este proyecto hemos desarrollado una herramienta para ayudar al profesorado a realizar un seguimiento individualizado de los alumnos durante las sesiones o talleres de programación. Para ello, hemos extendido COLMENA para recolectar nueva información de los estudiantes a través de un *plug-in* para el IDE Eclipse. De forma transparente para el usuario, el *plug-in* captura el estado actual del proyecto en cada compilación y envía esa información a un LRS, o almacén de datos central, siguiendo el formato estándar xAPI.

También hemos construido un *dashboard* que muestra información procesada y detallada a partir de los datos del LRS, y pone a disposición herramientas para visualizar esos datos de forma fácil y cómoda a través de una gráfica interactiva, una tabla con las compilaciones realizadas y un comparador de ficheros. Con estas herramientas se facilita al profesor la revisión de sesiones de programación y la detección temprana de alumnos con problemas de aprendizaje: *stoppers* o *tinkerers*.

Como resultado de la investigación realizada en este proyecto se ha creado el artículo "Learning analytics specifications applied to Integrated Development Environments" y se ha enviado a la revista del "Computer Applications in Engineering Education" JCR:

- Web: http://onlinelibrary.wiley.com/journal/10.1002/(ISSN)1099-0542
- Editorial: Wiley-Blackwell
- ISSN: 1061-3773
- Online ISSN: 1099-0542
- Impact Factor 2014: 0.296 (Thomson Reuters Journal Citation Report)
- 5-Year Impac Factor: 0.933 (Thomson Reuters Journal Citation Report)
- Estado del artículo: recibido.

La copia de autor del artículo aparece en los anexos del proyecto.

Como trabajo futuro nos proponemos extender el LRS, manteniendo la compatibilidad con xAPI, para realizar las tareas de procesamiento y análisis de los datos en el servidor en lugar del cliente JavaScript. También nos proponemos realizar analíticas más potentes que se apliquen no sólo por alumno y proyecto de forma individual como ahora, sino a todo el conjunto de los datos almacenados en el servidor, aplicando técnicas de Big Data.

Además, el *plug-in* actual de COLMENA requiere una conexión a Internet para enviar la información al servidor. Se podría mejorar almacenando los datos en un servidor local cuando no hay acceso a Internet, y más tarde subir toda la información local cuando la conexión a Internet vuelve a estar disponible.

Por otro lado, el *dashboard* también se puede mejorar implementando características adicionales, dando más información sobre el contexto de los estudiantes aplicando nuevas analíticas a los errores recolectados. Igualmente, se puede mejorar la usabilidad y la interfaz de usuario del *dashboard*.

Por último, podríamos analizar la viabilidad de utilizar los datos del LRS y el *dashboard* para otros usos como, por ejemplo, detectar código copiado y pegado durante

exámenes prácticos, observando cómo un alumno ha introducido muchos caracteres en muy poco tiempo entre una compilación y la siguiente (picos muy altos en el ritmo de escritura).

8 Bibliografía

[1] L. E. Winslow, "Programming Pedagogy—A Psychological Overview," *SIGCSE Bull.*, vol. 28, no. 3, pp. 17–22, Sep. 1996.

[2] E. Lahtinen, K. Ala-Mutka, and H.-M. Järvinen, "A Study of the Difficulties of Novice Programmers," in *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, New York, NY, USA, 2005, pp. 14–18.

[3] R. Lister, E. S. Adams, S. Fitzgerald, W. Fone, J. Hamer, M. Lindholm, R. McCartney, J. E. Moström, K. Sanders, O. Seppälä, B. Simon, and L. Thomas, "A Multi-national Study of Reading and Tracing Skills in Novice Programmers," in *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education*, New York, NY, USA, 2004, pp. 119–150.

[4] A. Robins, J. Rountree, and N. Rountree, "Learning and Teaching Programming: A Review and Discussion," *Computer Science Education*, vol. 13, no. 2, pp. 137–172, Jun. 2003.

[5] M. C. Jadud, "Methods and Tools for Exploring Novice Compilation Behaviour," in *Proceedings of the Second International Workshop on Computing Education Research*, New York, NY, USA, 2006, pp. 73–84.

[6] P. Blikstein, M. Worsley, C. Piech, M. Sahami, S. Cooper, and D. Koller, "Programming Pluralism: Using Learning Analytics to Detect Patterns in the Learning of Computer Programming," *Journal of the Learning Sciences*, vol. 23, no. 4, pp. 561–599, Oct. 2014.

[7] R. Cardell-Oliver, "How Can Software Metrics Help Novice Programmers?," in *Proceedings of the Thirteenth Australasian Computing Education Conference - Volume 114*, Darlinghurst, Australia, Australia, 2011, pp. 55–62.

[8] A. Begel and B. Simon, "Novice Software Developers, All over Again," in *Proceedings of the Fourth International Workshop on Computing Education Research*, New York, NY, USA, 2008, pp. 3–14.

[9] S. Dawson and G. Siemens, "Analytics to literacies: The development of a learning analytics framework for multiliteracies assessment," *The International Review of Research in Open and Distance Learning*, vol. 15, no. 4, Aug. 2014.

[10] E. Duval, "Attention Please!: Learning Analytics for Visualization and Recommendation," in *Proceedings of the 1st International Conference on Learning Analytics and Knowledge*, New York, NY, USA, 2011, pp. 9–17.

[11] M. Scheffel, K. Niemann, A. Pardo, D. Leony, M. Friedrich, K. Schmidt, M. Wolpers, and C. D. Kloos, "Usage Pattern Recognition in Student Activities," in *Towards Ubiquitous Learning*, C. D. Kloos, D. Gillet, R. M. C. García, F. Wild, and M. Wolpers, Eds. Springer Berlin Heidelberg, 2011, pp. 341–355.

[12] P. Denny, A. Luxton-Reilly, and E. Tempero, "All Syntax Errors Are Not Equal," in *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in*

Computer Science Education, New York, NY, USA, 2012, pp. 75–80.

[13] J. Jackson, M. Cobb, and C. Carver, "Identifying Top Java Errors for Novice Programmers," in *Frontiers in Education, 2005. FIE '05. Proceedings 35th Annual Conference*, 2005, p. T4C–T4C.

[14] M. C. Jadud, "A First Look at Novice Compilation Behaviour Using BlueJ," *Computer Science Education*, vol. 15, no. 1, pp. 25–40, 2005.

[15] C. Fernandez-Medina, J. R. Pérez-Pérez, V. M. Álvarez-García, and M. del P. Paule-Ruiz, "Assistance in Computer Programming Learning Using Educational Data Mining and Learning Analytics," in *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education*, New York, NY, USA, 2013, pp. 237–242.

[16] C. Fernández-Medina, J. R. Pérez-Pérez, M. P. Paule-Ruiz, and V. M. Álvarez García, "COLMENA: Collaborative knowledge and user classification environment based on programming experience," in *Proceedings of the VIII Multidisciplinary Symposium on Design and Evaluation of Digital Content for Education (Ciudad Real, Spain*, 2011.

[17] C. Glahn, "Using the ADL Experience API for Mobile Learning, Sensing, Informing, Encouraging, Orchestrating," in *2013 Seventh International Conference on Next Generation Mobile Apps, Services and Technologies (NGMAST)*, 2013, pp. 268–273.

[18] I. Guy, I. Ronen, and A. Raviv, "Personalized Activity Streams: Sifting Through the 'River of News,'" in *Proceedings of the Fifth ACM Conference on Recommender Systems*, New York, NY, USA, 2011, pp. 181–188.

[19] M. Wolpers, J. Najjar, K. Verbert, and E. Duval, "Tracking Actual Usage: the Attention Metadata Approach.," *Journal of Educational Technology & Society*, vol. 10, no. 3, 2007.

[20] W. Wang, J.-F. Weng, J.-M. Su, and S.-S. Tseng, "Learning portfolio analysis and mining in SCORM compliant environment," in *Frontiers in Education, 2004. FIE 2004. 34th Annual*, 2004, p. T2C–17.

[21] K. Niemann, M. Scheffel, and M. Wolpers, "An Overview of Usage Data Formats for Recommendations in TEL," in *Proceedings of the 2nd workshop on recommender systems for technology enhanced learning (RecSysTEL-2012). http://ceur-ws. org*, 2012, vol. 896.

9 Anexos

9.1 Planificación y Presupuesto

9.1.1 Planificación

Para el calendario del proyecto se toma como inicio el 16 de enero de 2014 y como fecha de entrega el 18 de mayo de 2015. La jornada de trabajo es de 2,5h al día, de lunes a viernes, a realizar por una sola persona, aunque se diferencian dos perfiles de cara al presupuesto: investigador y desarrollador software.

La estimación total de horas de trabajo es de 586h, compuestas por 440h de investigación y 146h de desarrollo. Sin embargo, la duración del proyecto se ha alargado debido a múltiples periodos de inactividad como vacaciones, cambios de domicilio, situaciones laborales y viajes por motivo de trabajo de hasta un mes.

En la Figura 10 se puede ver el diagrama de Gantt del proyecto y la lista de tareas.

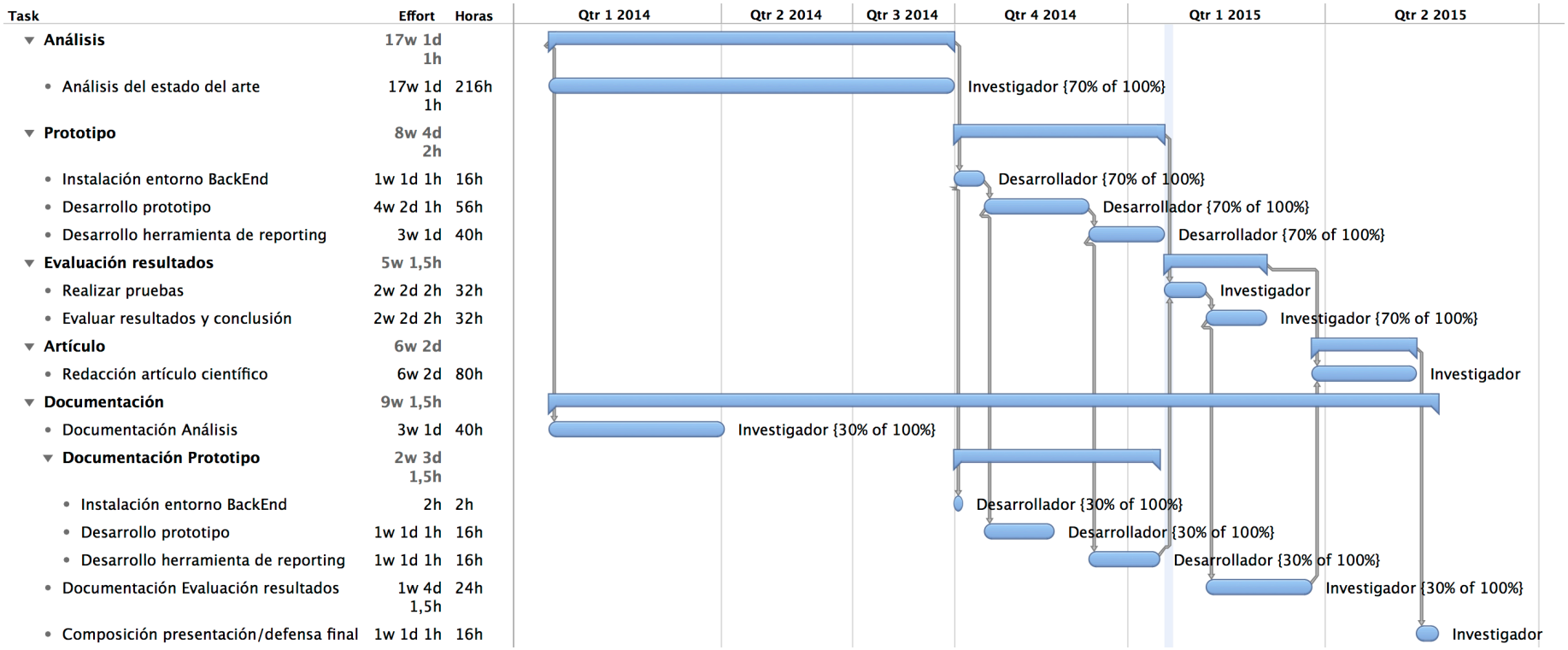


Figura 10 Planificación

9.1.2 Presupuesto

El presupuesto estimado del trabajo es de 30.694,37€ por una labor de investigación y desarrollo, con un coste de 35€/h y 30€/h por el trabajo del investigador y el desarrollador software, respectivamente. También se incluye el coste de amortización de las licencias y equipos necesarios durante el desarrollo del trabajo, establecido en un 25% anual, que debido a una duración del proyecto de 17 meses, se estima un 35% de coste de amortización para este periodo.

En la Tabla 1 se puede ver el presupuesto detallado.

Item	Subitem	Concepto	Cantidad	Coste Unitario	Coste Total	Coste Item
1		**Investigación + Desarrollo Software**				19.780,00€
	1	Investigador	440	35,00€	15.400,00€	
	2	Desarrollador	146	30,00€	4.380,00€	
2		**Coste de amortización de licencias y equipos**				513,80€
	1	Puesto de trabajo	0,35	999,00€	349,65€	
	2	Office 2011	0,35	269,00€	94,15€	
	3	OmniPlan 2	0,35	199,99€	70,00€	
					TOTAL BRUTO	***20.293,80€***

	Total Bruto	20.293,80€
25%	*Beneficio*	5.073,45€
	Total	25.367,25€
21%	*IVA*	5.327,12€
	TOTAL PRESUPUESTO	**30.694,37€**

Tabla 1 Presupuesto

9.2 Formatos para Captura de Datos

Existen varios formatos de almacenamiento y procesado de analíticas de aprendizaje. Guardar la información en un formato propio, creado ad hoc, puede parecer una buena idea inicialmente, pero requiere crear un diseño inicial, construir herramientas, validar que sea eficiente en el manejo de un volumen elevado de datos, que se pueda extender en el futuro, etc. Por ello, el uso de un formato estándar en el sector, con un diseño sólido, herramientas ya creadas y probadas, que sea extensible para cubrir las necesidades específicas de nuestro proyecto, y que permita abrir los datos a otros grupos y comunidades, presenta más ventajas que un formato propio.

Para guardar los datos recogidos en nuestro proyecto se puede elegir entre varios formatos de almacenamiento y procesado de analíticas de aprendizaje.

9.2.1 Experience API (xAPI)

Experience API [17], también llamado TinCan API o xAPI, surge como una evolución de SCORM para cubrir nuevos casos de uso y romper viejas restricciones. SCORM (Sharable Content Object Reference Model) es un conjunto de estándares y especificaciones que permite crear objetos de aprendizaje e importarlos a sistemas de gestión de aprendizaje, Learning Management Systems (LMS) como Moodle o Blackboard, por ejemplo. SCORM es el estándar de e-learning más utilizado para mantener la consistencia del formato de los cursos entre distintos sistemas de e-learning [20], y antes de su aparición, se utilizaban formatos propietarios para crear objetos de aprendizaje, por lo que no se podía, o era muy costoso, intercambiar esos contenidos entre diferentes sistemas de gestión de aprendizaje. Es un estándar puramente técnico, no entra en temas pedagógicos, y trata de satisfacer los siguientes requerimientos:

- Acceder a los contenidos remotamente, a través de Internet.
- Personalizar la formación.
- Resistir a la evolución de las tecnologías.
- Poder utilizarse en distintas plataformas.
- Ser reusable.

ADL (Advanced Distributed Learning)[1], la organización a cargo del estándar SCORM, decide presentar un nuevo estándar en colaboración con otras organizaciones para resolver algunas limitaciones de SCORM y añadir nuevas capacidades. Se identifican nuevos escenarios de aprendizaje, como los simuladores, los juegos formativos, actividades en el mundo real, aprendizaje en el móvil, aprendizaje en compañía, de forma colaborativa, offline, etc. que ahora sí se pueden registrar con la nueva especificación, Experience API. Ya no depende únicamente de la web, se pueden medir los avances de los usuarios en cualquier momento, lugar y actividad, ya sea desde una web, una herramienta en el escritorio o una aplicación en el móvil.

[1] http://www.adlnet.gov/

Figura 11 Experience API

Experience API permite capturar datos acerca de las experiencias que tiene una persona. Por ejemplo, si el usuario ha leído un capítulo determinado de un libro, si ha visto un vídeo hasta el final, accedido a una página web, modificado un documento, etc. Estas experiencias se pueden representar como sentencias, o statements, utilizando el formato Activity Streams (ver apartado 9.2.1.1), para ser después enviadas y almacenadas en un Learning Record Store (LRS).

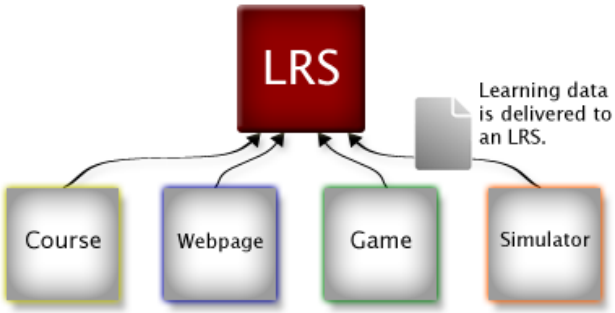


Figura 12 Learning Record Store

Un LRS es un repositorio de registros de aprendizaje o experiencias que puede ser leído por otros LRS, herramientas de informes o LMSs, y que puede ser independiente o estar embebido dentro de un LMS. Recuerda al sistema de control de versiones Git, ya que un LRS no se trata de un repositorio centralizado, sino distribuido, con la posibilidad de transferir la información de un LRS a otro, sea central o no. Esto permite posibilidades de configuración más amplias. Por ejemplo, podemos hospedar un LRS dentro de una aplicación móvil almacenando las experiencias que el usuario realiza en el teléfono sin necesidad de una conexión constante con un LRS central. Esta configuración nos aporta una arquitectura más

sólida y robusta preparada para situaciones en las que la cobertura del dispositivo sea mala o nula, pudiendo reportar las sentencias al LRS local del teléfono, y más tarde, bajo condiciones de cobertura óptimas, transferir todos los datos al LRS central.

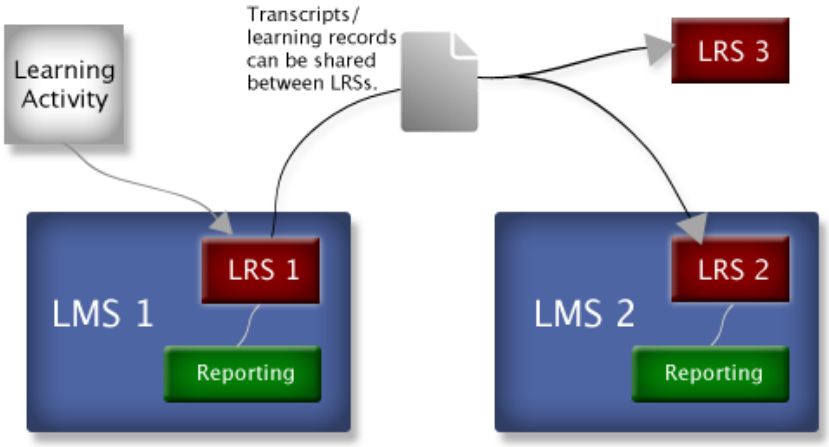


Figura 13 LRS distribuidos

9.2.1.1 Activity Streams

Las "experiencias" que realizan los usuarios se modelan siguiendo el formato abierto Activity Streams [18], que define una estructura "Actor – Verbo – Objeto" en formato JSON. Estos Activity Streams se enviarán y almacenarán en un LRS que permitirá que otros sistemas accedan a su información para poder realizar labores de análisis, estadísticas, reportes, etc. Este formato surge para ofrecer una semántica más rica que la de otros formatos de redifusión como RSS o Atom, especialmente orientado a los flujos de noticias (streams) de las redes sociales. Facebook[1] y MySpace[2] son algunas de las webs sociales que implementan este formato, pero también se ha convertido en una parte esencial en aplicaciones empresariales como Yammer de Microsoft o Chatter de Salesforce.

A partir de este formato surge la especificación[3] para definir las experiencias, o statements, sin apenas cambios con respecto a Activity Streams. Un ejemplo sencillo de cómo expresar la experiencia "Julia ha subido una imagen" en un Activity Stream es:

```
{
        "actor": "urn:example:person:julia",
        "verb": "post",
        "object": "http://example.org/foo.jpg"
}
```

El formato es extensible y se pueden añadir nuevas propiedades al archivo JSON sin romper la compatibilidad con el estándar. Esta flexibilidad se consigue mediante el campo 'extensions' en la definición de una actividad, un contexto o un resultado, permitiendo incluir información muy específica que sólo sea valiosa para una aplicación o un grupo sin romper la compatibilidad con el resto de aplicaciones. Por ello, un LRS debe tener cuidado al leer la

[1] https://www.facebook.com/

[2] https://myspace.com/

[3] https://github.com/adlnet/xAPI-Spec/blob/master/xAPI.md

información que recibe en el campo 'extensions', ya que puede contener propiedades arbitrarias que sólo sean útiles para otras aplicaciones o LRS. Siguiendo el ejemplo anterior, podemos añadir una nueva propiedad al contexto indicando desde qué plataforma móvil se ha realizado dicha "experiencia":

```
{
        "actor": "urn:example:person:julia",
        "verb": "post",
        "object": "http://example.org/foo.jpg",
        "context": {
                "extensions": {
                        "http://example.org/appPlatform": "iOS"
                }
        }
}
```

Aprovechando el campo "extensions", podremos añadir información específica de nuestro contexto, como el nombre del archivo en el que el estudiante ha introducido un error sintáctico, por ejemplo.

9.2.1.2 Implementaciones

Existen varias implementaciones de un LRS. ADL ofrece una implementación de referencia[1] para que sirva como guía para las demás organizaciones a la hora de construir su propio LRS. Cumple con la especificación 1.0.1 de xAPI y se puede acceder a una versión demo desplegada en sus servidores sobre la que ejecutar pruebas o ejemplos. Además, ofrecen múltiples librerías y herramientas para simplificar el uso de un LRS:

- jxapi[2]: una librería Java que permite conectarse a un LRS para enviar experiencias y consultar las ya existentes.
- xAPIWrapper[3]: librería JavaScript cuyo objetivo es simplificar la comunicación completa con un LRS, enviando y consultando experiencias.
- xAPI-Dashboard[4]: librería JavaScript que permite simplificar el procesado, extracción y representación de experiencias almacenadas en un LRS. Puede ser de gran utilidad de cara a la generación de reportes.
- Statement Validator[5]: página web para verificar si el formato de una experiencia cumple con la especificación o no.

[1] https://github.com/adlnet/ADL_LRS

[2] https://github.com/adlnet/jxapi

[3] https://github.com/adlnet/xAPIWrapper

[4] https://github.com/adlnet/xAPI-Dashboard

[5] https://lrs.adlnet.gov/xAPI/statementvalidator

Además de estas librerías, ADL ofrece más herramientas y ejemplos en su apartado para desarrolladores.

Por otro lado nos encontramos con Learning Locker[1], una implementación de código libre de un LRS. Sus principales características es que en una única instancia se pueden configurar varios LRS y además dispone de una API adicional, más potente que la estándar de xAPI, para hacer consultas y generar reportes más completos. A diferencia de ADL, Learning Locker no ofrece librerías cliente o herramientas, centran todos sus esfuerzos en el desarrollo del LRS.

También existen otras implementaciones propietarias como Wax LRS[2] (Software as a Service) o GrassBlade[3] (licenciado).

9.2.2 CAM (Contextualized Attention Metadata)

Contextualized Attention Metadata (CAM) [19] permite seguir la interacción del usuario en cualquier entorno, un navegador web, una aplicación de ofimática u otra herramienta. El schema CAM define una estructura transversal para modelar esas interacciones y registrar no sólo cuál es el foco de atención del usuario, sino todo su comportamiento en el entorno. Por ejemplo, “el usuario A edita un documento de texto X en la aplicación Y”, “el usuario B escuchar la canción Z en la aplicación T”, “el usuario A escribe un mensaje M en el foro F dentro del hilo H”, “el usuario A envía un email E al usuario B en el cliente de correo Y a las 23:50”, etc. Para ello, cada una de las aplicaciones utilizadas por el usuario deberá registrar las interacciones del usuario y almacenarlas siguiendo el schema CAM, o bien en un formato propio que después se pueda transformar al schema CAM[19]. Un análisis posterior de los datos de CAM recogidos permitiría ver qué acciones ha realizado el usuario, en qué aplicaciones, sobre qué documentos, en qué momento… y descubrir estadísticas de uso o popularidad, entre otros casos.

Originalmente el schema CAM surge como una extensión de AttentionXML[19] añadiendo el elemento “evento” para permitir capturar información sobre todos los eventos en los que se han visto involucrados los documentos. En las siguientes iteraciones del schema se ha ido refinando y dando cada vez más protagonismo al elemento “evento”, hasta llegar a la versión actual (2.0), en la que es la pieza central del schema CAM[21]. En CAM 2.0, la información capturada en cada evento es bastante simple y no incluye apenas datos de las entidades que participan en cada evento (usuarios, documentos, páginas web, etc.). Sin embargo, estos metadatos se enlazan desde cada evento. De esta forma no se repite información y las entidades se pueden representar de una forma diferente y más apropiada. De hecho, no hay un schema definido para describir las entidades, se puede utilizar uno propio u otro ya existente[21]. Por ejemplo, para no incluir toda la información del perfil del usuario en cada evento o acción que realiza, esta información se crea una sola vez, en el formato que se desee, y se incluye un enlace a ella en cada uno de los eventos en los que este usuario está involucrado.

[1] http://learninglocker.net/

[2] https://www.waxlrs.com/

[3] http://www.nextsoftwaresolutions.com/grassblade-lrs-experience-api/

En la Figura 14 se puede ver un diagrama de los elementos del schema en la versión 2.0.

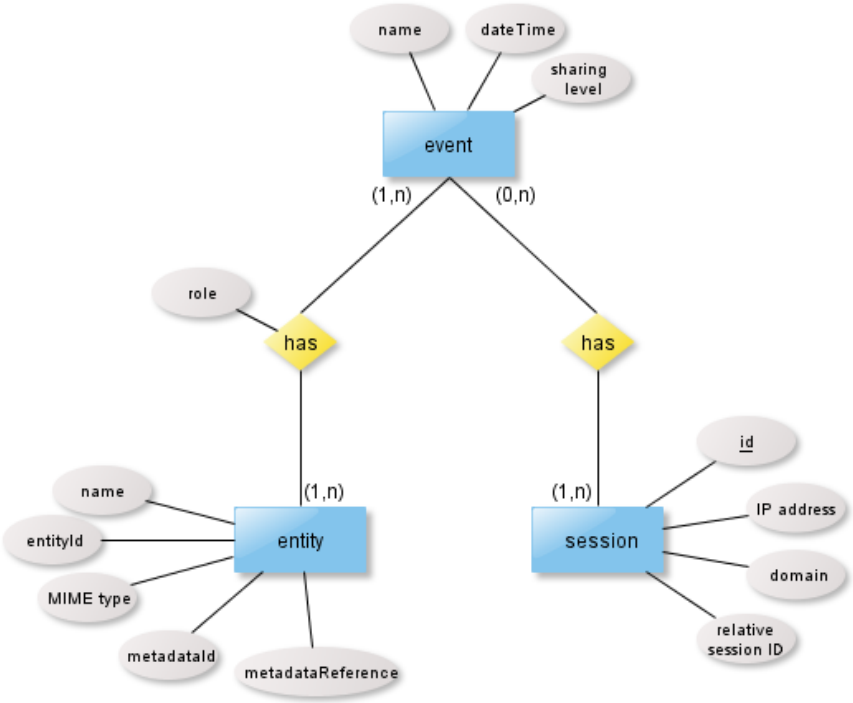


Figura 14 CAM schema 2.0

El elemento principal de una instancia CAM es el "evento", que tendrá un nombre que describa la acción, la fecha en la que se realizó, y el nivel de privacidad del evento. Sólo se almacena esta información básica de un evento para evitar repetir información recurrente, como por ejemplo los datos del usuario. Por ello, los usuarios o los documentos que están involucrados en un evento no se almacenan dentro del elemento "evento", sino en "entidades" enlazadas. Dependiendo del tipo de evento, puede haber varias "entidades" con distintos roles. Por ejemplo, en el caso de una instancia CAM que describa "el usuario A edita un documento de texto X en la aplicación Y", tendremos tres "entidades", una que describa al usuario con el rol de "sujeto" o "usuario", otra para el documento con el rol "documento", y otra más para identificar a la aplicación con la que se ha realizado la edición, con el rol "aplicación". Además, cada evento puede tener lugar en una "sesión", que puede ser el tiempo desde que se inicial el ordenador hasta que se apaga, o directamente una sesión web.

La versión actual del schema CAM no tiene un formato predefinido. Las instancias CAM se pueden representar y almacenar en XML, RDF, JSON o en una base de datos relacional[21], pero desde la web oficial[1] se ofrecen utilidades para empezar a utilizar este formato, como por ejemplo:

- CAM API: una librería Java para crear instancias CAM y serializarlas o deserializarlas en formato JSON.

[1] https://sites.google.com/site/camschema/home

- CAM JavaScript: una librería JavaScript para crear instancias CAM de forma manual.
- CAM Webservice: una aplicación web que proporciona una interfaz para acceder a la base de datos donde almacenar instancias CAM. Permite insertar y consultar los datos a través de peticiones HTTP POST. No genera estadísticas o cualquier otro tipo de informe, sino que tendremos que crear nosotros mismos el reporte recuperando los datos y procesándolos.

9.2.3 Conclusión formatos para captura de datos

El hecho de que CAM trate a los documentos sobre los que trabaja el usuario de una forma tan estática, esperando que apenas cambien con el tiempo, hace que no sea adecuado para nuestro proyecto, ya que necesitamos capturar el estado de todos los documentos en cada evento. Además, la documentación disponible sobre CAM es escasa, está poco organizada y es confusa. Por ejemplo, la propia estructura del schema CAM parece que aún no es definitiva y sigue en desarrollo.

Otro punto que tampoco se adapta a nuestros requisitos es que no existe una estructura por defecto para describir las entidades involucradas en un evento, sino que queda en manos del usuario, pudiendo elegir un formato propio o incluso otro formato distinto. Esto reafirma aún más el protagonismo del elemento "evento" dentro del schema, olvidándose de definir una estructura común para las entidades y dificultando la interoperabilidad entre fuentes de información que ya tengan datos almacenados en este formato.

Las principales ventajas de xAPI son su extensibilidad, la apertura de los datos y la libertad de plataforma y conexión. El formato que se utiliza para almacenar la información permite guardar casi cualquier actividad, se puede extender para soportar nuevos modelos de datos sin romper la compatibilidad y al ser abierto se pueden compartir los datos con otros sistemas que también implementan el mismo formato. Además, cualquier dispositivo puede enviar información, ya que se basa en peticiones REST, e incluso no es necesaria una conectividad permanente, haciéndolo muy apropiado para entornos móviles.

En cuanto al soporte de estos formatos, xAPI goza de más popularidad, contando con el apoyo de gobierno de los Estados Unidos, así como la existencia de varias implementaciones comerciales además de la implementación de referencia. En comparación, sólo hemos sido capaces de encontrar una implementación de CAM, la de referencia.

Por tanto, aprovecharemos el estándar xAPI para almacenar y representar los datos recogidos en nuestro proyecto, utilizando Learning Locker como implementación de un LRS, ya que nos ofrece más posibilidades para crear reportes y analíticas.

9.3 Dashboard visualización de datos

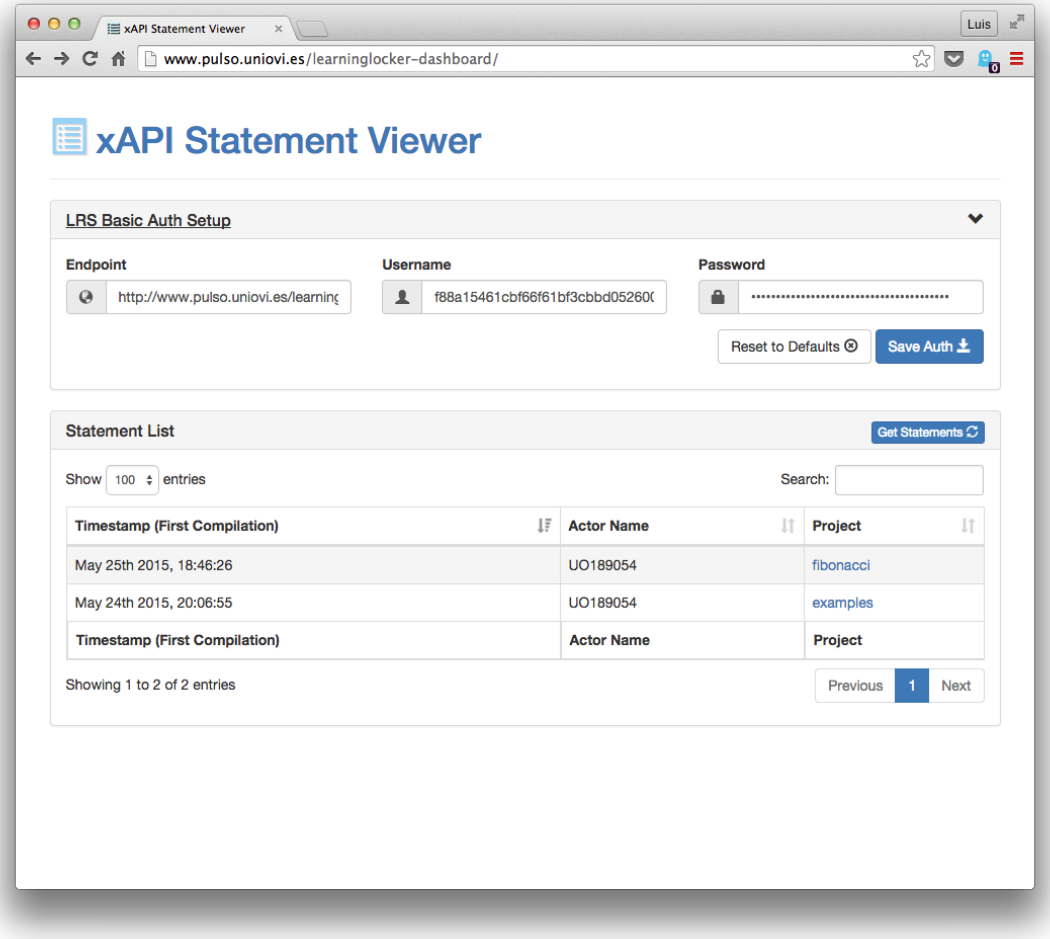


Figura 15 Listado de proyectos

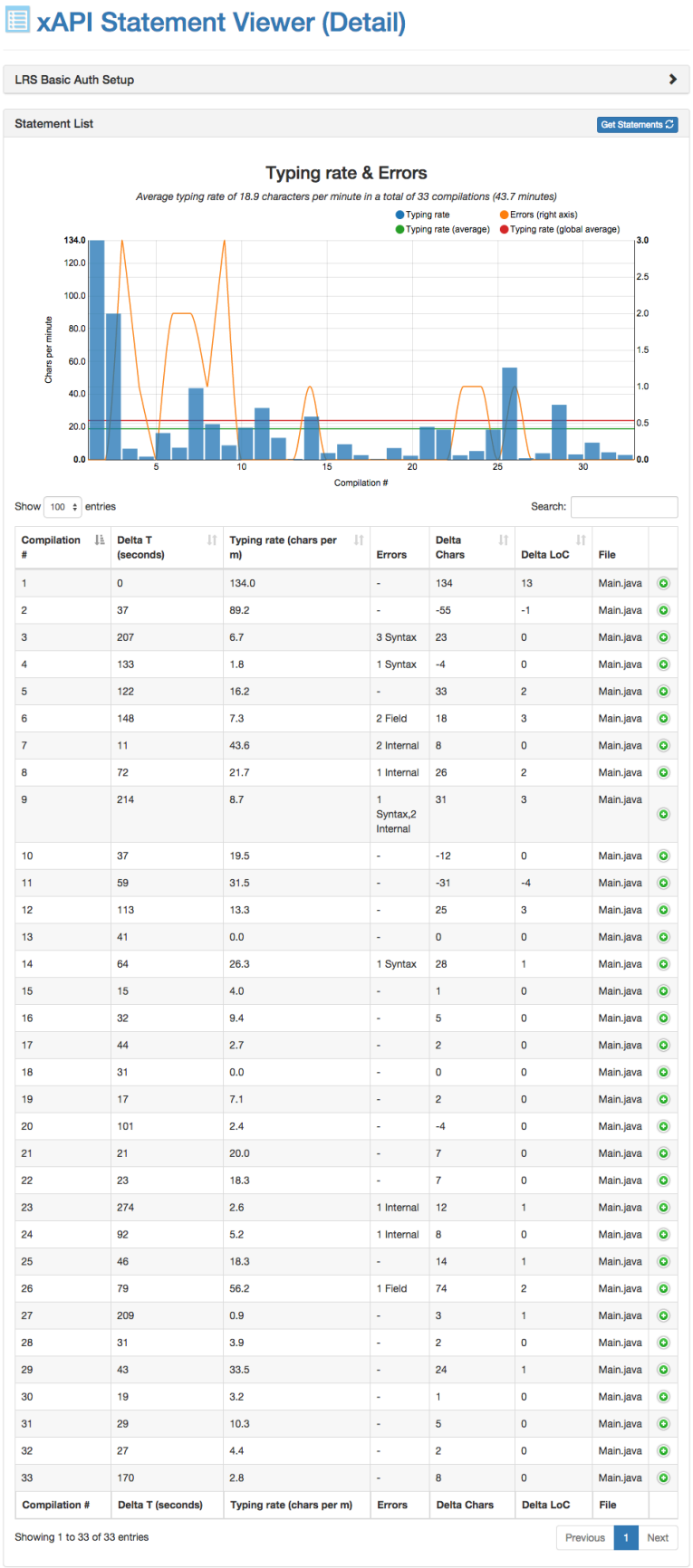


Figura 16 Dashboard de un proyecto