COLMENA: Modelo para la asistencia en la programación potenciado por la tecnología basado en análisis de datos

Universidad de Oviedo

Carlos Fernández Medina

Director
Dr. Juan Ramón Pérez Pérez

Programa de Doctorado en Informática
Universidad de Oviedo

Oviedo, España

Noviembre de 2023

Universidad de Oviedo

RESUMEN DEL CONTENIDO DE TESIS DOCTORAL

1.- Título de la Tesis	
Español/Otro Idioma: COLMENA: Modelo para la asistencia en la programación potenciado por la tecnología basado en análisis de datos	Inglés: COLMENA: Technology-Enhanced model for programming languages assistance using data analytics

2.- Autor	
Nombre: **Carlos Fernández Medina**	
Programa de Doctorado: Programa de Doctorado en Informática	
Órgano responsable: Centro Internacional de Postgrado	

RESUMEN (en español)

En los últimos años, la mejora continua del rendimiento durante el desarrollo del software ha sido uno de los campos de trabajo de los docentes e investigadores, que siempre han considerado uno de sus objetivos mejorar los métodos de aprendizaje de programación. Para tal efecto, se han llevado a cabo procedimientos y estudios que tratan de buscar patrones o variables que supongan una influencia sobre este aprendizaje, ya sea bien a través de estudios teóricos o prácticos o bien mediante herramientas software.

Unas de las variables que ha aparecido con asiduidad en este tipo de estudios son los errores y warnings generados durante la programación y, más específicamente, aquellos generados en tiempo de compilación. El interés por los errores está constatado por los diferentes artículos que han sido periódicamente publicados en publicaciones científicas y congresos nacionales e internacionales de docencia de la informática, en los que de forma general se busca una relación entre errores de programación y un mal resultado, guiando a los docentes en las áreas y conceptos de programación que se tendrían que reforzar. Estos estudios se enmarcan normalmente fuera del proceso de aprendizaje de los estudiantes, y aportan un gran valor para extraer conclusiones generales de cara a posteriores cursos, pero no ofrecen la posibilidad de proporcionar una realimentación en tiempo real a los estudiantes.

De una forma similar a los estudios teóricos, en la actualidad numerosos grupos o empresas se han centrado en desarrollar y mantener herramientas o complementos para los entornos de Desarrollo Integrados (IDE) que facilitan la asistencia a los desarrolladores durante la programación. A pesar de que algunas de estas herramientas son de calidad, el uso de las mismas no garantiza que los desarrolladores realmente aprendan a programar ni que mejoren la calidad de su código, pudiendo incluso perder dominio sobre el lenguaje de programación que están utilizando.

En la presente tesis se propone un modelo, COLMENA, orientado hacia este campo, el aprendizaje de la programación de software, utilizando como entorno principal de aprendizaje los IDEs. El modelo propuesto ha sido implementado en un contexto real a través de un sistema distribuido de información y consulta, el cual se integra con el IDE en las tareas de recopilar información como errores de compilación y warnings, y a su vez mostrándola en un soporte de aplicación web dando lugar a la realización de un estudio pormenorizado.

El entorno ha sido puesto a prueba en el contexto universitario, con estudiantes en varias asignaturas de programación a lo largo de diferentes cursos académicos, y el estudio del caso real ha permitido la captura y análisis de información enriquecida sobre el comportamiento de los estudiantes, dando a conocer en qué conceptos y temas los estudiantes cometen más errores. Adicionalmente, esta información sobre errores se visualiza para los usuarios clasificada por familias de errores, sesiones de prácticas y estudiantes, permitiendo al profesor realizar un seguimiento completamente personalizado, para un grupo o un estudiante concreto, y además documentar los distintos errores estableciendo una relación de estos con los conceptos de programación y buenas prácticas.

Al utilizar una herramienta llamada COLMENA, cualquier profesor puede brindar a los estudiantes retroalimentación formativa efectiva que sea específica de la tarea, inmediata y correctiva. Los resultados indican que los estudiantes novatos que recibieron retroalimentación del profesor a través de COLMENA redujeron sus errores, lo que demuestra que la retroalimentación generada a partir de errores de compilación es efectiva.

Con esta investigación, por tanto, se consigue un flujo de intercambio de información que puede asistir a profesores y estudiantes para una mejora del proceso enseñanza-aprendizaje, donde los primeros tengan una herramienta que enriquezca el contexto donde se efectúa el proceso de enseñanza-aprendizaje y obtengan así información que les permita ofrecer un feedback formativo correcto y de calidad a los segundos.

RESUMEN (en Inglés)

In recent years, continuous improvement during software development has been one of the areas of work for lecturers and researchers, who have always considered one of their goals to improve programming learning methods. For this purpose, procedures and studies have been carried out that try to find patterns or variables that influence this learning, either through theoretical or practical studies or through software tools.

One of the variables that has appeared with relative regularity in this type of study is the errors and warnings generated during programming and, more specifically, those generated at compile time. The interest in errors is verified by the different papers that have been periodically published in academic journals or international conferences on computer science teaching, in which a relationship between programming errors and a bad result is generally sought, guiding teachers in the areas and programming

concepts that should be reinforced. These studies are normally framed outside the student's learning process and provide great value for drawing general conclusions for subsequent courses, but they do not offer the possibility of providing real-time feedback to students.

In a similar way to theoretical studies, currently, many groups or companies have focused on developing and maintaining tools or plug-ins for Integrated Development Environments (IDEs) that facilitate assistance to developers during programming. Even though some of these tools are of quality, their use does not guarantee that developers will really learn to program or that they will improve the quality of their code, and they may even lose control over the programming language they are using.

This phd. proposes a model, COLMENA, oriented towards this field, the learning of software programming, using IDEs as the main learning environment. The proposed model has been implemented in a real context through a distributed information and consultation system, which is integrated with the IDE in the tasks of collecting information such as compilation errors and warnings, and in turn, showing it in an application support. website leading to a detailed study.

The environment has been put to the test in the university context, with students in several programming subjects throughout different academic years, and the study of the real case has allowed the capture and analysis of enriched information on the behavior of the students, allowing them to know in which concepts and topics students make more mistakes. Additionally, this error information is displayed for users classified by families of errors, practice sessions, and students, allowing the teacher to carry out a completely personalized follow-up, for a group or a specific student, and also document the different errors establishing a relationship of these with programming concepts and good practices.

By using a tool called COLMENA, any lecturer may give students effective formative feedback that is task-specific, immediate, and corrective. The results indicate that novice students receiving feedback from the lecturer via COLMENA reduced their errors, demonstrating that feedback generated from compilation errors is effective.

Through this research, therefore, an information exchange flow is achieved that can assist teachers and students to improve the teaching-learning process, where the former have a tool that enriches the context where the teaching-learning process takes place and thus obtains the information that allows them to offer correct and quality formative feedback to the seconds.

SR. PRESIDENTE DE LA COMISIÓN ACADÉMICA DEL PROGRAMA DE DOCTORADO EN INFORMÁTICA

Resumen

En los últimos años, la mejora continua del rendimiento durante el desarrollo del software ha sido uno de los campos de trabajo de los docentes e investigadores, que siempre han considerado uno de sus objetivos mejorar los métodos de aprendizaje de programación. Para tal efecto, se han llevado a cabo procedimientos y estudios que tratan de buscar patrones o variables que supongan una influencia sobre este aprendizaje, ya sea bien a través de estudios teóricos o prácticos o bien mediante herramientas software.

Unas de las variables que ha aparecido con asiduidad en este tipo de estudios son los errores y warnings generados durante la programación y, más específicamente, aquellos generados en tiempo de compilación. El interés por los errores está constatado por los diferentes artículos que han sido periódicamente publicados en publicaciones científicas y congresos nacionales e internacionales de docencia de la informática, en los que de forma general se busca una relación entre errores de programación y un mal resultado, guiando a los docentes en las áreas y conceptos de programación que se tendrían que reforzar. Estos estudios se enmarcan normalmente fuera del proceso de aprendizaje de los estudiantes, y aportan un gran valor para extraer conclusiones generales de cara a posteriores cursos, pero no ofrecen la posibilidad de proporcionar una realimentación en tiempo real a los estudiantes.

De una forma similar a los estudios teóricos, en la actualidad numerosos grupos o empresas se han centrado en desarrollar y mantener herramientas o complementos para los entornos de Desarrollo Integrados (IDE) que facilitan la asistencia a los desarrolladores durante la programación. A pesar de que algunas de estas herramientas son de calidad, el uso de las mismas no garantiza que los desarrolladores realmente aprendan a programar ni que mejoren la calidad de su código, pudiendo incluso perder dominio sobre el lenguaje de programación que están utilizando.

En la presente tesis se propone un modelo, COLMENA, orientado hacia este campo, el aprendizaje de la programación de software, utilizando

como entorno principal de aprendizaje los IDEs. El modelo propuesto ha sido implementado en un contexto real a través de un sistema distribuido de información y consulta, el cual se integra con el IDE en las tareas de recopilar información como errores de compilación y warnings, y a su vez mostrándola en un soporte de aplicación web dando lugar a la realización de un estudio pormenorizado.

El entorno ha sido puesto a prueba en el contexto universitario, con estudiantes en varias asignaturas de programación a lo largo de diferentes cursos académicos, y el estudio del caso real ha permitido la captura y análisis de información enriquecida sobre el comportamiento de los estudiantes, dando a conocer en qué conceptos y temas los estudiantes cometen más errores. Adicionalmente, esta información sobre errores se visualiza para los usuarios clasificada por familias de errores, sesiones de prácticas y estudiantes, permitiendo al profesor realizar un seguimiento completamente personalizado, para un grupo o un estudiante concreto, y además documentar los distintos errores estableciendo una relación de estos con los conceptos de programación y buenas prácticas.

Al utilizar una herramienta llamada COLMENA, cualquier profesor puede brindar a los estudiantes retroalimentación formativa efectiva que sea específica de la tarea, inmediata y correctiva. Los resultados indican que los estudiantes novatos que recibieron retroalimentación del profesor a través de COLMENA redujeron sus errores, lo que demuestra que la retroalimentación generada a partir de errores de compilación es efectiva.

Con esta investigación, por tanto, se consigue un flujo de intercambio de información que puede asistir a profesores y estudiantes para una mejora del proceso enseñanza-aprendizaje, donde los primeros tengan una herramienta que enriquezca el contexto donde se efectúa el proceso de enseñanza-aprendizaje y obtengan así información que les permita ofrecer un feedback formativo correcto y de calidad a los segundos.

Palabras Clave

Enseñanza de la programación, Errores de programación, Entorno de desarrollo integrado (IDE), Feedback formativo, Análisis de Datos, Data Mining, Clustering.

Abstract

In recent years, continuous improvement during software development has been one of the areas of work for lecturers and researchers, who have always considered one of their goals to improve programming learning methods. For this purpose, procedures and studies have been carried out that try to find patterns or variables that influence this learning, either through theoretical or practical studies or through software tools.

One of the variables that has appeared with relative regularity in this type of study is the errors and warnings generated during programming and, more specifically, those generated at compile time. The interest in errors is verified by the different papers that have been periodically published in academic journals or international conferences on computer science teaching, in which a relationship between programming errors and a bad result is generally sought, guiding teachers in the areas and programming concepts that should be reinforced. These studies are normally framed outside the student's learning process and provide great value for drawing general conclusions for subsequent courses, but they do not offer the possibility of providing real-time feedback to students.

In a similar way to theoretical studies, currently, many groups or companies have focused on developing and maintaining tools or plug-ins for Integrated Development Environments (IDEs) that facilitate assistance to developers during programming. Even though some of these tools are of quality, their use does not guarantee that developers will really learn to program or that they will improve the quality of their code, and they may even lose control over the programming language they are using.

This phd. proposes a model, COLMENA, oriented towards this field, the learning of software programming, using IDEs as the main learning environment. The proposed model has been implemented in a real context through a distributed information and consultation system, which is integrated with the IDE in the tasks of collecting information such as compilation errors and warnings, and in turn, showing it in an application support. website leading to a detailed study.

The environment has been put to the test in the university context, with students in several programming subjects throughout different academic years, and the study of the real case has allowed the capture and analysis of enriched information on the behavior of the students, allowing them to know in which concepts and topics students make more mistakes. Additionally, this error information is displayed for users classified by families of errors, practice sessions, and students, allowing the teacher to carry out a completely personalized follow-up, for a group or a specific student, and also document the different errors establishing a relationship of these with programming concepts and good practices.

By using a tool called COLMENA, any lecturer may give students effective formative feedback that is task-specific, immediate, and corrective. The results indicate that novice students receiving feedback from the lecturer via COLMENA reduced their errors, demonstrating that feedback generated from compilation errors is effective.

Through this research, therefore, an information exchange flow is achieved that can assist teachers and students to improve the teaching-learning process, where the former have a tool that enriches the context where the teaching-learning process takes place and thus obtains the information that allows them to offer correct and quality formative feedback to the seconds.

Keywords

Computer Science learning, Programming Errors, Integrated Developed Environment (IDE), Formative Feedback, Data Analysis, Data Mining, Clustering.

Agradecimientos

A mi Padre, por enseñarme a jugar a las palas en la Almadraba.

A mi Madre, por enseñarme que si te esfuerzas, hay recompensa.

A Reyes, por enseñarme que los colores y las formas no tienen límites.

A Desirée, por enseñarme aquel día 20 de marzo de 2005, que todos los días empieza la Primavera.

Índice de contenidos

Índice de figuras

Índice de tablas

1. Introducción

1.1 Antecedentes y contexto

1.1.1 Aprendizaje de la programación

Java[1] es uno de los lenguajes de programación más utilizado en el mundo, con un 12,5% de representación sobre el total, según el índice internacional TIOBE[2]. Debido a su popularidad, Java es también uno de los lenguajes de programación más impartidos en las escuelas de ingeniería para sus titulaciones de informática. En 2014 un estudio demostró que Java estaba entre los tres lenguajes más utilizados en las titulaciones de informática más prestigiosas en Estados Unidos [83]. La facilidad de aprendizaje del mismo y su condición de lenguaje de propósito general válido para prácticamente cualquier solución software, encaja perfectamente con la variedad de asignaturas de las enseñanzas universitarias y la enorme tipología de proyectos software que se pueden abordar con su manejo.

Lo más frecuente es que para desarrollar código fuente se utilicen los Entornos de Desarrollo Integrados (o IDE por las siglas de su versión en inglés, Integrated Development Environments), programas cuya finalidad es ayudar al usuario en las tareas de diseño y desarrollo de software. Uno de los objetivos principales de un IDE es aislar las tareas relacionadas con el uso del lenguaje (configuración, preparación de dependencias, compilación, uso de librerías, etc) de forma que visualmente el desarrollador pueda ocuparse de lo que está relacionado específicamente con la programación. Además, suelen ofrecer un entorno visual enriquecido para desarrollar código en editores amigables, facilitando al máximo la complejidad innata de este proceso. En la depuración, ofrecen mecanismos para visualizar e inspeccionar el código fuente, dando un extra de facilidad al desarrollador.

[1] https://www.java.com/es/

[2] https://www.tiobe.com/tiobe-index/

En la actualidad existen varios IDEs líderes en el mercado, tanto en versiones comerciales como académicas. En el espectro comercial, Visual Studio[3], de Microsoft, es uno de los más completos, que además ofrece también una versión gratuita y más enfocada al desarrollo web, conocida como Visual Studio Code[4]. Otro fuerte competidor en el terreno comercial es la familia de JetBrains, con IDEs específicos según el lenguaje de programación (PHP, Python, .NET environment o Java) y con versiones educativas (Edu), con licencias menos costosas o incluso gratuitas para promocionar el uso de las herramientas y acercarlas al público estudiante. En la versión de Java, el software que JetBrains desarrolla se conoce como IntelliJ IDEA[5]. Dentro del espectro de IDEs sin propósito comercial, sin duda Eclipse[6] es una referencia fundamental. Focalizado principalmente en Java, destaca por su extensa documentación y libros de referencia para el desarrollo de extensiones sobre la propia plataforma [16]. Con tintes más académicos y con el propósito de facilitar el el aprendizaje de Java, compite frente a Eclipse el entorno BlueJ[7]. Con una interfaz más simple que Eclipse, también posee documentación exhaustiva y libros para el aprendizaje y dominio del IDE [4].

Los Entornos de Aprendizaje Virtuales (en inglés, Virtual Learning Environments, o VLEs) dan otro tipo soporte a los estudiantes para el aprendizaje, a través de tecnologías multimedia y de comunicación [59]. Los VLEs Facilitan el acceso a variedad de materiales personalizados, ofreciendo la posibilidad de definir perfiles de usuario a los que asociar características concretas como preferencias de aprendizaje, antecedentes o su progreso en aprendizaje para ofrecer así materiales personalizados, lo que se denomina "e-learning adaptativo" [58]. Son ideales para el aprendizaje a distancia y hacen posible que, a través de Campus Virtuales, un estudiante pueda acceder a una formación de forma no presencial. Estos entornos, sin embargo, presentan pocos artefactos que faciliten la mejora

[3] https://visualstudio.microsoft.com/es/

[4] https://code.visualstudio.com/

[5] https://www.jetbrains.com/es-es/idea/

[6] https://www.eclipse.org/

[7] https://www.bluej.org/

del aprendizaje en programación, y tampoco dan soporte a un feedback en tiempo real.

En el contexto académico de este trabajo, el enfoque del aprendizaje de los lenguajes de programación se basa en la realización de ejercicios teóricos sobre "papel" para adquirir los conceptos básicos y, como complemento, sesiones prácticas de laboratorio utilizando un IDE. Además, el curso académico puede verse reflejado en un entorno de aprendizaje virtual (VLE) desde donde el estudiante puede hacer un seguimiento de los materiales y las tareas que constituyen una asignatura, confeccionados y puestos a su disposición por parte del personal docente.

1.2 Motivación

1.2.1 Dificultades del aprendizaje de la programación

El estudio de las dificultades que viven los estudiantes a la hora de aprender programación, así como las causas de sus errores es un tema que se ha tratado desde hace décadas, apareciendo ya durante los años 80 estudios empíricos realizados sobre la programación basados en lenguaje Pascal [92]. Más adelante, en la década de los 90, Ben-Ari, en su enfoque constructivista para la enseñanza de Informática, insistiría en este concepto de dificultad de la materia de programación [10]. Concretamente, a través de varios estudios empíricos, los cuales demostrarían dichas dificultades, debido a problemas a la hora de asimilar conceptos tales como el aprendizaje de las variables, los parámetros o los conceptos generales de la programación orientada a objetos [42, 64, 82]. Gomes et al en 2007 extendieron la tipología de causas que potencian estas dificultades, siendo éstas bien los métodos de enseñanza (enseñanzas no personalizadas, estrategias de enseñanzas que no incluyen a todos los estilos de aprendizaje, la enseñanza de materia dinámica a través de materiales estáticos o la no focalización en la solución práctica de problemas), los métodos de estudio (metodologías inadecuadas o poco esfuerzo), las habilidades de los estudiantes (poca habilidad para resolver problemas, poca experiencia en matemáticas, lógica o programación) y su psicología (poca motivación en un periodo difícil de su vida) o bien la propia

naturaleza de la programación (altos niveles de abstracción, sintaxis complejas) [37]. Es por esto que, uno de los objetivos principales de investigadores y docentes es desarrollar mecanismos para la predicción de estudiantes en situación riesgo, a través del análisis del rendimiento de los mismos o el conocimiento de las causas de sus carencias durante la etapa académica [94, 106].

Un caso destacable aparece en las carreras de Ingeniería Informática y Ciencias de la Computación, en las que el aprendizaje de los conocimientos de programación se convierte en una tarea difícil. Según los estudios de McCracken et al, en torno al 35% de los estudiantes suspenden durante el primer curso, y concretamente en Estados Unidos o Reino Unido, el 30% de los estudiantes no comprenden los conceptos base de programación [73]. Otro estudio conducido por Bennedsen et al sobre 63 instituciones diferentes en enseñanzas de programación encontró que en torno al 34% de los estudiantes no pasaba del primer curso en enseñanzas de Ingeniería Informática [11]. En una revisión reciente de la enseñanza y aprendizaje de la programación en cursos de introducción, se demuestra que el tema cada vez es más recurrido a la hora de realizar estudios e investigaciones [63].

Desde un punto de vista docente, es extremadamente importante conocer los puntos en los que los estudiantes están teniendo dificultades, de cara a abordar unas estrategias u otras a la hora de enseñar [15]. El seguimiento individualizado de los estudiantes se vuelve una tarea compleja ya que, además de supervisarlos, los profesores tienen que realizar otras tareas como evaluación o preparación del material de las clases. La revisión del código individualizado, por tanto, se convierte en una labor prácticamente imposible de compaginar con el resto de actividades docentes. En búsqueda de mecanismos e instrumentos que permitan mejorar durante ese proceso de desarrollo, existen estudios en los que las iniciativas de colaboración a la hora de desarrollar código, como las técnicas de pair programming dentro de la metodología de Extreme Programming (XP) que suponen una evidencia en la mejora de rendimiento [108]. No obstante estas iniciativas no siempre son fáciles de llevar a cabo en los contextos académicos.

Cambiando la perspectiva hacia los estudiantes, el aprendizaje de un lenguaje de programación puede ser entendido como una tarea ardua, y existen varios estudios que constatan la complejidad de los conceptos y las habilidades que los estudiantes tienen que adquirir en asignaturas de aprendizaje de la programación [54, 84]. Al igual que para los docentes, revisar el código también es una tarea fundamental para el estudiante, y facilitar una revisión autónoma repercute positivamente en el aprendizaje [109] (Figura 1).

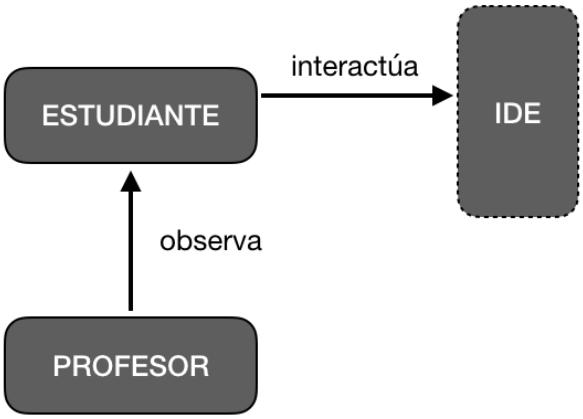


Figura 1. Escenario tradicional donde el estudiante desarrolla el código y el profesor monitoriza su estado directamente a partir de las acciones del propio estudiante.

Poniendo en común la situación actual de los docentes y los estudiantes, parece evidente que lograr disponer de un sistema de análisis y retroalimentación (feedback) puede aportar mucha ayuda a ambos usuarios. Es por esto que diseñar una herramienta capaz de almacenar el trabajo que va realizando cada estudiante a lo largo del proceso de programación de cualquier proyecto, y que también permite agrupar, analizar y facilitar la colaboración y generación de feedback a través un análisis visual de dicho trabajo, puede suponer una ayuda para tomar decisiones apropiadas en tiempo real durante el proceso de enseñanza-aprendizaje [38] (Figura 2).

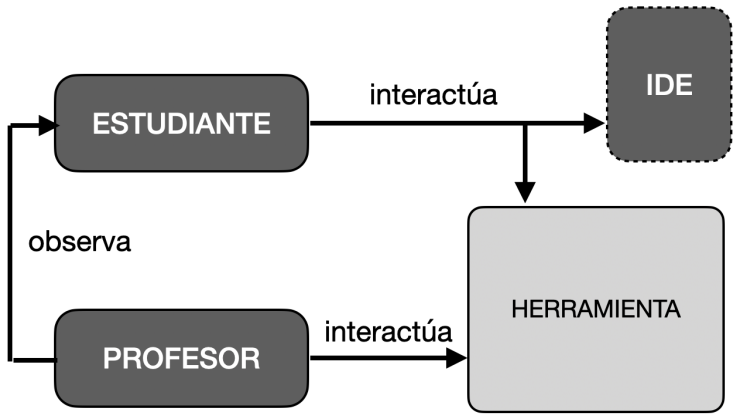


Figura 2. Escenario enriquecido donde el estudiante desarrolla el código y el profesor monitoriza su estado directamente, pero ambos tienen acceso a una herramienta que les permite ampliar sus conocimientos.

1.2.2 Uso de IDEs en un contexto académico

Para los estudiantes, utilizar una herramienta como un IDE, la cual ofrece funcionalidades de edición, compilación y depuración, es ideal para focalizar su aprendizaje en el uso del lenguaje y las labores que le corresponden en cada asignatura, sin perder tiempo ni esfuerzos en labores menos interesantes en esta etapa de su formación, tales y como pueden ser la configuración del entorno u otras tareas que de otra forma implicaría el uso de varias herramientas o trabajo manual.

A pesar de todas las ventajas que los IDEs les puedan proveer a los estudiantes, también presentan algunas limitaciones en cuanto los errores generados por los estudiantes. La volatilidad de la información sobre los errores y warnings en tiempo de compilación es una de ellas, ya que los IDEs no almacenan esta información en ningún sistema. De este modo, es posible que los estudiantes, tras varias compilaciones, ya no recuerden las causas de un error generado previamente, repitiéndolo de nuevo y no habiendo aprendido a solucionarlo. Esta repetición sistemática de los mismos errores es una demostración de que el estudiante está teniendo problemas a la hora de programar [5, 6].

La capacidad de los IDEs de producir código autogenerado también supone un riesgo, ya que puede hacer que los estudiantes tengan una dependencia demasiado alta del entorno, sin aprender realmente los conceptos base del lenguaje [44]. Otro punto negativo son los mensajes de error devueltos por el compilador, crípticos y difíciles de entender, impidiendo que los estudiantes sean conscientes de los conceptos que hay detrás de los errores que cometen [7, 72, 103]. Además, los errores en ocasiones vienen acompañados de otros errores producidos en cadena, los cuales entorpecen nuevamente conocer realmente qué error se ha producido [74]. Estos problemas mencionados anteriormente suponen dificultades añadidas al proceso natural de aprendizaje de la programación a los estudiantes.

1.2.3 El feedback formativo en la programación

El feedback formativo ha sido reconocido históricamente como una herramienta facilitadora durante el proceso de aprendizaje, teniendo como uno de sus objetivos principales contribuir a que los estudiantes mejoren su conocimiento y su rendimiento [52, 90, 100]. En entornos de aprendizaje basados en el uso del ordenador, la combinación de feedback implícito (sin una comunicación directa entre el emisor y el receptor) y explícito (interacción específica de retroalimentación emisor-receptor) puede proporcionar asistencia en tiempo real para las tareas de desarrollo de código [104]. De esta manera, el feedback puede servir para resolver de forma inmediata la diferencia entre el estado actual del proceso de aprendizaje del estudiante y los resultados de aprendizaje que se desean. Adicionalmente, estos entornos de aprendizaje con ordenador pueden ayudar a los profesores permitiéndoles ofrecer un feedback más individualizado a los estudiantes que el feedback convencional que pueden obtener ellos solos a través de un IDE. A pesar de ser un elemento clave en el aprendizaje, el feedback no siempre consigue ser efectivo, considerándose diversos parámetros para valorar esta efectividad, como la naturaleza, dimensión temporal y los portadores del mismo [43]. Tomando como referencia estos parámetros, el feedback más efectivo se define según algunos autores como aquel basado en la tarea específica, y que es dado de forma inmediata, positiva, correctiva y sin ánimo de juicios [89, 97].

En cuanto a su clasificación, el feedback puede ser analizado desde dos puntos de vista; desde el docente o bien desde el estudiante:

- En el primer caso, se trata de un formato de tutorización en el que el profesor interactúa directamente con el estudiante, y donde el feedback cumple con los parámetros mencionados anteriormente [12]. Sin embargo, en el contexto académico actual, no siempre es posible conseguir este formato de tutorización, ya que existen grupos de estudiantes en los que el profesor dispone de un tiempo limitado, y muchas veces insuficiente, para realizar este seguimiento individualizado.
- Por otra parte, desde la perspectiva de los estudiantes existen estudios que afirman que los estudiantes más noveles dedican mucho tiempo a corregir los errores de compilación que les devuelve el IDE [8], que para ellos hace una función de proxy de los profesores. Los estudiantes, por tanto, tardarán en corregir los errores y aprender el lenguaje de programación en función del feedback que les dé el IDE durante todo su proceso de aprendizaje. Existen también investigaciones que indican que el feedback sobre su evaluación o el feedback formativo durante su aprendizaje es necesario, llegando a demandar feedback escrito que cumpla los parámetros anteriores, y resaltando la importancia de que esté presente un refuerzo positivo [31, 91].

No poder aplicar en tiempo real este formato de tutorización es uno de los motivos por el que en los últimos años se haya profundizado en el desarrollo de herramientas o sistemas que generen feedback de una forma más automática [56, 65]. En este tipo de sistemas, es habitual hacer uso de técnicas de Learning Analytics para la recolección, análisis y generación de informes sobre los aprendices y los contextos en los cuales se produce el aprendizaje, para así generar un feedback acorde a los parámetros mencionados sin depender de una tutorización individualizada exclusivamente [53, 68].

Algunas investigaciones ya han trabajado en la generación automática de feedback basándose en la medición de parámetros durante el proceso de desarrollo [51], siendo el proporcionar feedback con ejemplos de código

para corregir los errores una de las líneas de trabajo empíricamente más contrastadas. En este área, los estudios realizados han demostrado que los estudiantes perciben estos ejemplos como una ayuda [75, 96, 103]. Sin embargo, el feedback generado desde el IDE y basado en este tipo de ejemplos hacia el estudiante, es pedagógicamente cuestionable, ya que no hay una garantía de que el estudiante pueda relacionar el ejemplo genérico a un error específico, o que por otra parte el estudiante tenga el suficiente conocimiento para encontrar sentido al ejemplo proporcionado [8]. Además, a nivel práctico, Pather et.al encontraron que mostrar ejemplos de código a los estudiantes novatos para corregir los errores les confundía, ya que entonces dedicaban mucho tiempo en buscar ese ejemplo en sus ficheros de código, generando una distracción [79].

Otro enfoque de feedback formativo en los IDEs, es aquel que está basado en dar sugerencias o pistas (hints) que ofrecen soluciones para conocer cómo los errores pueden ser solucionados. Este feedback ha demostrado ser efectivo cuando estas pistas se usan en situaciones donde hay muchas posibilidades de que la sugerencia propuesta corrige el error específico [78], de lo contrario se puede producir el efecto de "llevar al usuario por el camino equivocado" [66]. Motivado en parte por las desventajas que tiene este tipo de feedback, algunas investigaciones optan por mezclar ejemplos con sugerencias [96]. Este enfoque tiene la misma carencia que el anterior, es decir, el feedback puede ser inefectivo porque no está contextualizado en una tarea (task-specific) [89, 97]. Si la solución propuesta no corrige el error específico, el feedback por tanto no va a ser correctivo, ni tampoco positivo.

Investigaciones recientes han incluido mejoras en relación al feedback proporcionado por los IDEs, permitiendo así que éste sea formativo e inmediato simultáneamente [25, 57]. A pesar de este progreso los autores de citadas investigaciones han confirmado que este feedback es mínimo, asistiendo al programador en corregir errores pero no informando sobre las razones por las cuáles el error se produce, creando una situación donde se hace más difícil conceptualizar el error en sí y entenderlo [8]. Como consecuencia de esto, el feedback no es completamente formativo ya que no cumple que sea task-specific, correctivo y positivo al mismo tiempo [89,

97]. De hecho, este feedback será determinado por la calidad y cantidad de información relevante que el IDE provea y a partir de la misma, el entorno generará el feedback con ejemplos y pistas, de las cuales ya se conocen las potenciales desventajas.

Por tanto, el feedback automático todavía está lejos de estar lo suficientemente maduro. Mientras tanto, el profesor sigue siendo un agente fundamental en el proceso de enseñanza-aprendizaje de la programación. No obstante, se ha de enfrentar a la situación de enseñar a corregir los errores en una clase con muchos estudiantes donde dispone a su vez de poco tiempo para hacerlo. En la mayoría de los casos, su feedback está fundamentado en la experiencia de sus años de docencia, sin poder recuperar información de la situación específica de su grupo de estudiantes. Hay que recordar también que los profesores noveles no tienen experiencia y el feedback que pueden proporcionar se basa en su experiencia como estudiantes o profesionales, no como profesores.

1.3 Contribuciones

Ante la realidad descrita en el capítulo anterior, en la que se percibe una carencia tanto hacia los estudiantes como hacia los profesores en cuanto a información acerca del proceso de aprendizaje de la programación, es fundamental proporcionar herramientas que den soporte a los profesores para que éstos a su vez puedan dar feedback formativo a sus estudiantes. Con este propósito se ha desarrollado COLMENA, una modelo y una herramienta que ofrecen un feedback basado en dos conceptos:

1. Analíticas de los errores que cometen los estudiantes: a través de dashboards, el profesor puede saber cuáles son los errores más comunes de cada sesión, que estudiante lo ha cometido y cuantas veces han aparecido.

2. Ejemplos de errores en el código con su solución y sugerencias de cómo resolverlos, además de la causa que produce dicho error.

Frente a un feedback automático, el presente trabajo apuesta por potenciar el feedback formativo del docente con herramientas de soporte

como COLMENA. A través de las analíticas, el profesor es consciente de los errores más comunes en la sesión y que usuario lo ha cometido, por tanto, puede explicar específicamente el error y su causa, permitiendo un feedback task-specific e inmediato. Los ejemplos de los errores en el código con su solución unido a las sugerencias de cómo resolverlo, potencian el carácter correctivo y positivo del feedback. El diseño, desarrollo e implementación de todo COLMENA en un entorno real de aprendizaje, nos sirve por tanto para plantear un estudio piloto con asignaturas de introducción a la programación.

1.3.1 Preguntas de Investigación

Las preguntas de investigación (Research Questions o RQs en inglés) nos permiten delimitar la línea de investigación que este trabajo de tesis pretende abarcar. Se separan en dos grandes bloques, debido a las dos grandes iniciativas realizadas y que se definen en los posteriores capítulos principales: El análisis de los errores y el análisis del feedback formativo respectivamente.

Preguntas de investigación relacionada con el análisis de los errores de los estudiantes en contextos formativos:

1. ¿Se comportan los estudiantes de acuerdo a una distribución normal en función a los errores y warnings?

2. ¿Hay tipos de errores representativos? ¿Cuales son durante un curso académico?

Para estas dos preguntas de investigación, la hipótesis de partida es que los errores de programación en tiempo de compilación puedan definir la distribución de unos alumnos y que los errores sean representativos para un curso académico. Para corroborar esta hipótesis, la metodología que se lleva a cabo es el estudio estático de los errores en unas sesiones de prácticas y el análisis de los mismos durante un curso académico de un grado de Ingeniería Informática, de forma transparente para los estudiantes.

En relación al feedback, es necesario saber el efecto que provoca el feedback formativo del docente con COLMENA sobre los estudiantes, pero además se quiere comprobar ese efecto también sobre los errores. Es por ello que se plantean las siguientes cuestiones:

3. Por mensaje de error, ¿es el feedback efectivo?

4. Por estudiante, ¿es el feedback efectivo?

La hipótesis para estas dos preguntas de investigación es que una herramienta de soporte va a potenciar el feedback formativo dado por el profesor, permitiendo que sea más efectivo. Para corroborar esta hipótesis, la metodología de investigación que se aplica es un caso de estudio sobre estudiantes novatos matriculados en un grado de Ingeniería Informática. Específicamente, se comparan dos años académicos. Durante el primer año, el mismo profesor ofrece feedback formativo basado en su experiencia y el segundo año se apoya en COLMENA para dar el feedback formativo. Los resultados demuestran que las herramientas como COLMENA potencian el feedback formativo del profesor sobre los estudiantes y los errores.

1.4 Estructura del documento

Este documento se estructura de la siguiente forma:

- Introducción: Capítulo de acercamiento del contexto actual, la motivación del trabajo y la contribución que se pretende hacer a la comunidad científica a través de este proyecto.

- Trabajo relacionado: Capítulo donde se describe el estado del arte en relación a sistemas e iniciativas de investigación para el análisis y estudio de los errores que cometen los estudiantes (generalmente noveles) en asignaturas y ejercicios de programación; así como estudios en relación al feedback que se ofrece a los estudiantes de acuerdo a proporcionar información adicional en cuanto al proceso de enseñanza-aprendizaje.

- COLMENA: Capítulo dedicado a la descripción del modelo teórico y la implementación del mismo a través de distintas aplicaciones software.

- Análisis de los errores: Capítulo relacionado con los casos de estudio aplicados sobre los errores recopilados a través de diferentes partes de COLMENA, y los estudios resultantes de analizar su distribución en base a distintos parámetros.

- Evaluación del Feedback: Capítulo en el que se describe un caso de estudio concreto en búsqueda de la efectividad del feedback formativo que COLMENA proporciona a los profesores para que estos contribuyan activamente en las sesiones prácticas sobre los estudiantes.

- Conclusiones: Una síntesis del trabajo realizado.

- Trabajo Futuro: Una recopilación de sugerencias de potenciales líneas de continuidad de la investigación.

- Publicaciones: Serie de trabajos de diseminación científica en relación al propósito de este trabajo.

2. Trabajo Relacionado

2.1 Análisis de los errores de los estudiantes de programación

En el contexto académico de este trabajo, se sabe que las prácticas de laboratorio están guiadas por un profesor y que consisten en la resolución de problemas usando el lenguaje de programación para ir adquiriendo niveles cada vez mayores en el conocimiento del mismo. Las sesiones inicialmente tienen unos objetivos sencillos que promueven que el estudiante adquiera los conceptos básicos del manejo del lenguaje de programación y poco a poco van integrando conceptos más complejos, lo que hace que aumente su complejidad.

Analizando el comportamiento de los estudiantes con el IDE, éstos repiten un ciclo sistemáticamente a la hora de programar que consiste en escribir, compilar y probar el código generado. Este ciclo fue identificado y descrito originariamente por Jadud en diversos estudios con estudiantes en la Universidad de Kent [49] (Figura 3).

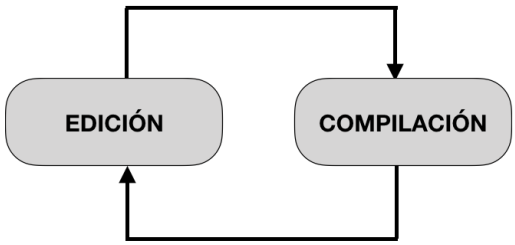


Figura 3. El ciclo de edición-compilación definido por Jadud

En cada paso de este ciclo, el estudiante va refinando el código para corregir los errores e ir añadiendo nuevas funcionalidades. Sin embargo, a lo largo de este ciclo puede darse el caso de que un estudiante no conozca los fundamentos de los errores que está cometiendo. Esta situación le hace seguir una mecánica de trabajo donde de forma reiterada se intenta resolver los proyectos a base de ensayo y error y, con el tiempo, el

estudiante puede encontrarse en una situación donde no es capaz de continuar el ritmo del grupo [73], desembocando en un abandono de la asignatura. Es por esto que resulta una tarea clave ser capaces de analizar lo que sucede durante estos ciclos para reducir la complejidad en el proceso de aprendizaje.

Para poder realizar estos análisis, existen sistemas que ofrecen analíticas de los errores en diferentes formatos (i.e dashboards), que pueden ser clasificadas de forma genérica en dos grandes familias: iniciativas comerciales y académicas. Dentro de las comerciales, se consideran dos de ellas las más representativas y relacionadas para con la presente investigación: Rollbar[8] y Newrelic[9]. Ambas herramientas ofrecen una clasificación de errores, seguimiento de los mismos, capacidad para analizar número de ocurrencias o incluso generar conversaciones para llevar a cabo soluciones. El análisis de las mismas nos permite afirmar que son herramientas muy potentes y de gran ayuda al proceso de desarrollo del software, pero no son muy adecuadas para estudiantes novatos por su complejidad a la hora de mostrar y explicar tanto el error como la solución del mismo.

En el entorno académico es donde se encuentran herramientas más destinadas a estudiantes novatos. En el estudio mencionado anteriormente, donde se describe el ciclo de edición-compilación, Jadud también realiza un análisis exhaustivo de los errores que los usuarios cometen en tiempo de compilación utilizando BlueJ. Concretamente, se centra en evaluar la cantidad y distribución de casi 2000 errores generados por 64 estudiantes novatos y la tipología de los mismos [47]. Este estudio así como el de Luke, suponen ya una diferencia respecto a las alternativas comerciales, al estar contextualizados en una universidad específica y ser plug-ins sobre un IDE de desarrollo, en lugar de una herramienta de terceros independiente [62].

El mismo Jadud, más adelante, en otros estudios, analiza el comportamiento del estudiante a la hora de compilar y generar errores,

[8] https://rollbar.com

[9] https://newrelic.com/

definiendo “eventos de compilación” dentro del ya mencionado anteriormente ciclo de “edición-compilación” [48, 49]. Para llevar a cabo estos estudios, a través de una extensión de BlueJ, se recuperan snapshots de código fuente con y sin errores y genera un coeficiente denominado Error Quotient (EQ) que mide la destreza de los estudiantes en el tratamiento de los errores de sintaxis, con el objetivo de hacer estudios más profundos sobre grupos de estudiantes. y que ha servido como punto de partida para otras investigaciones.

Un ejemplo es el trabajo de 2009 por parte de Tabanao et al [45], donde los autores analizaron la relación entre el EQ y el rendimiento, llegando a la conclusión de que un índice alto se correspondía con peores notas en los exámenes, pero afirmando también que EQ explica muy levemente el rendimiento. El mismo grupo de autores en 2011 realizaron otro estudio sobre una muestra de 124 estudiantes, con cerca de 25,000 eventos de compilación registrados, con el objetivo de clasificar a los estudiantes según se encuentren en una situación de más o menos riesgo. a partir de los errores que han cometido a lo largo de diferentes ejercicios de programación, y así predecir su calificación [94].

Siguiendo esta línea, Rodrigo et al emplearon el EQ de Jadud para analizar una muestra de 143 estudiantes y relacionar sus estados afectivos respecto al IDE. En el estudio se comprobó que los estudiantes que adquirieron estados de frustración o aburrimiento, no obtienen tan buenos resultados en los exámenes [85]. Paralelamente, Watson et al también realizaron su propio indicador (score) para predecir el rendimiento, denominado WatWin Score, el cual incorpora el factor del tiempo a la hora de realizar las predicciones. Los investigadores analizaron el algoritmo sobre 42 estudiantes y más de 45.000 compilaciones, obteniendo unas predicciones más precisas con relación al EQ de Jadud [106].

Recientemente, ha sido publicado un enfoque más detallado sobre el análisis de errores llevado a cabo por BlueJ [50]. En él, Becker y otros autores comparan el análisis y la producción de errores hecho por BlueJ 3 con su posterior versión, BlueJ 4. En el citado trabajo, los autores han destacado la importancia de la compilación automática de BlueJ 4 frente a la manual de su versión anterior. Además, añaden que esta compilación

automática posibilita mostrar los errores a los estudiantes de uno en uno basados en la compilación realizada por el entorno. Este enfoque tiene la ventaja de evitar la sobrecarga cognitiva de los estudiantes al percibir muchos errores. Sin embargo, la compilación automática tiene a su vez el problema de que el error puede aparecer antes de que el usuario haya terminado de escribir la sentencia, lo que es igualmente parcialmente confuso.

Existen también otras herramientas que no se basan tanto en BlueJ, como DevEventTracker, un sistema que hace de interfaz con los servicios Web-cat para hacer una captura de los datos de desarrollo del estudiante, sin que éste tenga que dedicar ningún esfuerzo adicional [28, 29]. De esta manera, los eventos de desarrollo y compilación se guardan dentro del Eclipse a través de un plug-in que, tras conectar inicialmente con Web-cat para la recuperación del proyecto y la preparación preliminar del mismo, envía de nuevo los datos de los eventos generados por el usuario al servidor, donde se realizan una serie evaluados de forma automática [51]. El sistema además proporciona un dashboard en forma de conjunto de páginas web para instructor y muestra datos útiles en formato de gráficos, tablas o resúmenes al estudiante. Comparando estos enfoques con el anterior de Jadud, se puede comprobar que el primero está enfocado más en los eventos de compilación mientras que DevEventTracker rastrea datos de otros eventos. Además, DevEventTracker monitoriza a los estudiantes en el desarrollo de los proyectos en Eclipse y Jadud se centra en las sesiones que realizan los estudiantes con BlueJ.

Becker et al también hace un análisis minucioso de los errores generados, clasificándolos según el número de apariciones y buscando la efectividad del feedback aportado por su propia herramienta, Decaf, a la hora de ofrecer mensajes de compilación enriquecidos (ECMs) [5]. En su estudio considera 48,800 errores generados, agrupados en 74 mensajes de compilación del error (CEMs) diferentes. Más adelante en otro trabajo, muestra que estos ECMs cambiaron el comportamiento de los estudiantes, ya que después de ver un error enriquecido, los estudiantes eran menos propensos a generar ese mismo error en el futuro. Así, tras realizar comparaciones con varios casos de estudios diferentes, concluye que los

ECMs reducen significativamente el número de errores generados, especialmente en los más frecuentes para la versión del compilador utilizada en el estudio [7].

Pettit et al. también trabajaron en enriquecer esos mensajes de error del compilador, a través de una herramienta automática, en este caso denominada Athene, y utilizada para el desarrollo en el lenguaje C++ [77]. Sin embargo, no encontraron resultados concluyentes en relación a que estos mensajes enriquecidos fueran más útiles que los convencionales generados por el compilador.

El equipo de Denny también creó una herramienta propia, CodeWrite, que también proporciona ECMs, incluyendo ejemplos concretos que ilustran el tipo de error que ha acontecido y posibles formas de solucionarlo. En sus trabajos, se centra en utilizar CodeWrite para llevar a cabo un análisis del código buscando los errores más frecuentes para los estudiantes principiantes en términos de sintaxis [22]. Analizando un grupo de 470 estudiantes, el experimento proporcionó una lista de errores frecuentes donde se manifestaba que pocos errores sucedían muchas veces. Sin embargo, no encontró un efecto significativo al tratar de evaluar la efectividad de los mensajes de error enriquecidos [23]. Posteriormente, Ettles et al realizó un experimento similar sobre una población de 809 estudiantes, clasificando los errores lógicos en lugar de los sintácticos, y obteniendo como resultado patrones similares de reparto de errores [30]. Estudios recientes de Denny et al han mostrado la dificultad que los estudiantes tienen para entender y conceptualizar ECMs, considerando los errores un problema grave, debido a su difícil legibilidad [25].

Estos trabajos espontáneos anteriormente citados sobre diversos grupos aislados, han dado lugar a la existencia de sistemas enfocados a la recogida de datos a gran escala y de fuentes heterogéneas. Por ejemplo, BlackBox [13, 14] es un gran sistema con una colección de datos de todo el mundo en relación al IDE BlueJ y estudiantes noveles. Los datos existentes se recopilaron durante un marco temporal de 5 años (2013-2018), son anónimos y disponibles para investigaciones abiertas, lo que ha beneficiado mucho a la comunidad científica. En 2018 tenía registrados más de 306 millones de eventos de compilación,

proporcionados por aproximadamente el 40% de los usuarios de BlueJ [3]. Los autores, en los 19 trabajos publicados, proponen ejemplos de análisis de los datos, destacando en la misma línea que Jadud, el top ten de los errores de compilación más frecuentes como herramienta de feedback.

Como es evidente, todos los estudios anteriores no solo presentan una gran cantidad de datos, si no que reafirman la necesidad de considerar los errores de compilación como una entrada relevante a la hora de abordar potenciales mejoras en el aprendizaje de lenguajes de programación. Sin embargo, estos trabajos no son una herramienta de apoyo al profesor en el feedback formativo al estudiante. Esto se debe a que principalmente no ofrecen un feedback elaborado que favorezca la efectividad en los resultados de aprendizaje, ya que no explican las posibles causas de los errores que den pistas para su solución con ejemplos reales [100].

COLMENA está alineada con propuestas anteriores como la de Jadud, Becker, Watson, Luke y el resto de autores que desarrollan herramientas sobre IDEs, alejándose más de la filosofía de repositorios más genéricos como BlackBox. Se considera que el tratamiento de datos a gran escala es adecuado para realizar predicciones asociadas al rendimiento, pero no para ofrecer un feedback que sea task-specific, correctivo, positivo e inmediato. Los enfoques de los autores mencionados en relación a recoger y tratar los eventos que realizan los estudiantes en el IDE suponen el sustrato del que nace el concepto de COLMENA, aunque en este caso el sistema está incluso más asociado al concepto de error-specific que los trabajos referenciados, al recoger el error asociado a una sesión en la que el profesor imparte un contenido específico en un determinado momento, no a un proyecto que el profesor analiza posteriormente.

2.2. Otros análisis de los datos generados por el usuario

A día de hoy existen herramientas que permiten realizar un análisis de las acciones del usuario frente a los entornos de desarrollo. Éstas herramientas, aunque no estén directamente relacionadas con errores de

programación, sin duda son fuente de referencia, a partir de motivaciones similares.

Un primer ejemplo es el trabajo de J. Spacco en el proyecto Marmoset [93], de la Universidad de Maryland. Su equipo recoge diferentes variables del proceso de entrega de prácticas, testeo y revisiones. A través de una aplicación software, los estudiantes pueden subir su código fuente para que sea evaluado, y conocer los fallos que se cometen.

Matsuzawa et al [69], haciendo uso de un plug-in propio para Eclipse, analiza la iteración constante del usuario con el entorno para así conocer qué tipo de código se está generando en cada momento, con el objetivo de medir los distintos elementos de código frente a las proporciones tiempo dedicado para las diferentes tareas de los estudiantes.

Price et al proponen un sistema denominado Stride, y que facilita la programación de forma más visual, integrándose con otro IDE, en este caso Greenfoot [80]. En su estudio, los investigadores miden dos grupos de estudiantes inexpertos, y detectan que el grupo que usa Stride, pese a no tener diferencias en cuanto a la satisfacción o la frustración, sí que supone una disminución del tiempo en errores de carácter meramente sintáctico.

El proyecto BRACElet por su parte, es una iniciativa multi-institucional concebida con el objetivo de ayudar a que los estudiantes novatos entiendan cómo leer y escribir código [17, 60, 61, 101, 107]. En este contexto, fueron diseñados varios experimentos en distintos workshops de la conferencia ITICSE, en los que se evaluaban distintas tareas. A los estudiantes que formaban parte de estos estudios, se los clasificaba en base a diferentes taxonomías, como la de Bloom o la de SOLO, y constituían una base de conocimiento sobre la que el grupo analizaba el comportamiento de los estudiantes.

Estos sistemas tratados anteriormente, la diferencia fundamental con COLMENA radica en que estas investigaciones, pese a trabajar en tiempo real sobre un entorno de desarrollo, no destacan por proporcionar información gráfica sobre la situación particular de cada estudiante de forma individual o dentro de un grupo. Eso es relevante desde el punto de

vista de ofrecer dashboards o interfaces gráficas amigables para que el docente pueda ofrecer un feedback formativo válido.

2.3 El Feedback formativo y el aprendizaje de la programación

En el entorno concreto de la enseñanza de la programación en informática, existen numerosos sistemas cuya finalidad es recopilar información sobre los estudiantes, especialmente sobre su actividad y comportamiento a la hora de desarrollar, para su posterior análisis [26]. Los estudios se centran normalmente en el uso y la aplicación de Lenguajes de Programación Visuales (VPL) o específicamente diseñados para la introducción y el aprendizaje de los conceptos básicos de programación, con un IDE personalizado para tal efecto [39, 76, 81]. En términos de representación de la información, la tendencia es emplear dashboards o interfaces visuales que aportan feedback formativo tanto para los estudiantes [99, 102] como para los profesores [2, 67].

Por ejemplo, el equipo de Tsai et al creó un VPL que permite a los estudiantes crear programas sencillos para entender mejor conceptos básicos de programación. Los investigadores realizaron un estudio con 180 estudiantes universitarios a lo largo de 17 semanas, durante las cuales desarrollaron proyectos de programación que posteriormente serán analizados de forma grupal entre todos los estudiantes con el apoyo del profesor para generar feedback. Al concluir el estudio, se obtuvieron mejoras por parte de los estudiantes en la comprensión de los conceptos, especialmente en los menos autónomos [99].

Otro caso significativo es el de Yang et al, los cuales desarrollaron en 2019 Pensieve, una herramienta Open Source que permite la visualización y ordenación de los snapshots de código fuente que los estudiantes hacen durante el proceso de evaluación, ofreciendo feedback al profesor para posteriores revisiones y tutorías sobre su código. La herramienta fue probada en un grupo de 207 estudiantes, obteniendo mejoras en el aprendizaje por el feedback proporcionado a los estudiantes en los pasos intermedios hasta llegar a una entrega de código final [76].

Lazarinis et al diseñaron en 2019 elementos multimedia y actividades en un curso de Moodle para potenciar el aprendizaje de los profesores de los lenguajes de programación. El feedback se transmite a través de vídeos de aprendizaje y la interacción directa de los grupos de profesores con el equipo de soporte. La herramienta fue aplicada a un grupo de más de 550 profesores. registrando una participación cercana al 70% y una alta satisfacción por parte de los estudiantes respecto al feedback obtenido del curso [55].

En una línea similar, Anvik et al crearon Program Wars, un juego de cartas interactivo que permite conocer conceptos de programación y de ciberseguridad. El juego, de una forma visual e intuitiva, proporciona a los jugadores feedback en forma de consejos y pasos incluidos en el propio juego para conocer mejor la información sobre las estructuras de control, los bucles o conceptos concretos. El sistema fue probado sobre 46 estudiantes facilitando la asociación de conceptos del juego con estructuras de programación [81].

También como un juego, Watson et al desarrollaron un juego en el que, a través de una interfaz interactiva, los estudiantes pueden visualizar los conceptos de programación y ejecutar el código para conocer el estado de las variables y resultado del mismo, de una manera más simplificada y clara que un IDE o sistema de e-learning convencional, donde se realizaría a través de materiales estáticos de un curso, recibiendo un feedback generalmente positivo [104].

Todos estos estudios presentan herramientas para facilitar la comprensión de conceptos relacionados con programación a los estudiantes. Sin embargo, ninguno de estos estudios utilizan de forma específica los errores que el estudiante genera. Del mismo modo, estos trabajos tampoco agrupan la información sobre los errores facilitando así su análisis y consulta y ofreciendo un feedback enriquecido de forma visual y amigable.

El feedback formativo de COLMENA se sustenta en analíticas de los errores que cometen los estudiantes, ejemplos de errores en el código con su solución y sugerencias de cómo resolverlos, además de la causa que

produce el error, y es por esto que los anteriores estudios, pese a ser excelentes ejemplos de feedback formativo, no van en relación al contexto sobre el que se realiza la investigación. Tal y como se explicó a lo largo de la introducción, el feedback formativo automático está basado fundamentalmente en ejemplos de código para corregir errores y en sugerencias que ofrecen soluciones de cómo los errores pueden ser corregidos. Tomando como análisis los ejemplos de código para corregir errores, en apartados anteriores de este capítulo se describen en detalle iniciativas como la de Denny, Becker o Pettit, que no son concluyentes en relación a este tipo de feedback.

En cuanto las sugerencias de soluciones o hints, se presenta un problema con relación al feedback formativo: no llega a ser orientado a la tarea (task-specific) sin poder concluir si la sugerencia propuesta a través del feedback corrige el error en sí. Este problema ha sido tratado desde diferentes prototipos como HelpMeOut o BlueFix con resultados poco satisfactorios [40, 103]. El primero de los dos es un recomendador social que ayuda y asiste en la depuración de mensajes de error sugiriendo soluciones que han sido aplicadas en el pasado por otros compañeros [40]. Sus creadores realizan una evaluación con estudiantes novatos y concluyen que el sistema es capaz de sugerir soluciones útiles para el 47% de los errores después de dedicar 39 horas programando en un entorno experimental. BlueFix es una herramienta online que está integrada con BlueJ y que fue diseñada para ayudar a los estudiantes de programación en la diagnosis de los errores y la solución de los mismos, sugiriendo fórmulas para resolver errores de sintaxis a través de una base de datos con soluciones de errores. Los autores han realizado una evaluación del plug-in, revelando que 11 estudiantes encontraron la herramienta útil como soporte de ayuda a corregir los errores, y una evaluación inicial de su precisión dice que mejora en torno a un 19.52% sobre HelpMeOut [103].

Además de las anteriores, recientemente ha aparecido una tercera vía para abordar el feedback fundamentada en la mezcla de ejemplos de código para solucionar los errores de compilación con sugerencias o hints. Por ejemplo, Thiselton & Treude desarrollaron dos herramientas software , la primera para extender los errores con una documentación formal

proveyendo ejemplos del uso correcto de las funcionalidades del lenguaje Python, y otro aportando soluciones para identificar errores utilizando las respuestas más valoradas en Stack Overflow [96]. La evaluación que hacen Thiselton & Treude es con 16 participantes (14 programadores profesionales y 2 estudiantes de los que no indican su nivel). La mayoría de los participantes valoran muy positivamente las respuestas de Stack Overflow, pero se ha de tener en cuenta que es una muestra muy pequeña y, básicamente, formada por profesionales que ya tienen una comprensión avanzada de los errores de programación.

Utilizando la ventaja que aporta la colección de datos a gran escala con herramientas como los MOOCS, han aparecido también algunos trabajos que afrontan el reto de "proveer un feedback completamente automático y personalizado para los cursos y ejercicios de introducción a la programación sin requerir un esfuerzo adicional por parte del profesor" [102]. Así, Wang et al han diseñado un framework que denominan "Search, Align, and Repair", cuyo objetivo es automatizar la generación de feedback a partir de la información subida por los estudiantes a entornos como Microsoft CodeHunt o programas concretos de Microsoft EDX. Tras poner en práctica el sistema, el equipo de Wang detectó que su framework ofrecía feedback útil y conciso para casi un 90% de los casos subidos a la plataforma [102]. La mayoría de estas técnicas modelan el problema como un problema de reparación de un programa, lo que consiste en reparar el contenido que un estudiante entrega para hacer algo funcionalmente equivalente. El resultado, en la mayoría de los casos, es que estas herramientas no son muy efectivas, poco prácticas y, en relación a conseguir buenos resultados, unos mínimos ajustes son siempre requeridos. Existen también otras técnicas, basadas en la detección y solución de los errores [1] no obstante, como dicen sus autores, "la precisión en la corrección de un fallo que difiere de un fallo del usuario requiere ser examinada manualmente, y no es práctico hacerlo a gran escala". Por último, existen algunas aplicaciones a gran escala de generación de pruebas o sugerencias como la propuesta por Mangpo y Sridhara [78]. La ventaja de estos enfoques es que gran cantidad de datos permiten aplicar distintas técnicas de generación de sugerencias, ofreciendo alternativas

para cualquier programador independientemente de lo próximo que se encuentre.

Ninguna de las tres formas citadas de proporcionar feedback formativo automático han solucionado el problema de que el feedback consiga ser task-specific, y mucho menos correctivo ni positivo. La razón fundamental radica en una dificultad técnica con doble vertiente, por un lado la generación no es eficiente ni práctica y por otro, existe una dificultad de corregir un error o generar una sugerencia o hints apropiada porque, es difícil proveer un contexto apropiado del error [70]. Sin embargo, el estudiante sigue necesitando un feedback formativo para ser capaz de continuar con su aprendizaje, y los IDEs no están diseñados para proporcionar est feedback ya que desde un punto de vista técnico, no proveen un análisis de errores centrado en el contexto de la tarea específica en la que el error acontece. Esta imposibilidad técnica no proporciona un feedback formativo efectivo ya que tanto los ejemplos como las sugerencias o pistas no están contextualizadas con el error concreto.

Teniendo en cuenta esta situación, la responsabilidad del proceso de enseñanza y aprendizaje recae en el docente, quién debe trabajar de forma frecuente con grupos de varios estudiantes con un proceso de aprendizaje que es específico para cada individuo dentro de ese grupo. Es por tanto necesario para el docente tener herramientas que den apoyo, como COLMENA; que refuercen y proporcionen este feedback ahora sí task-oriented, contextualizado, correctivo y positivo.

3. COLMENA: desde la extracción hasta la visualización de errores de compilación

COLMENA nace con la intención de ser capaces de detectar la falta de soporte e información relacionada con los errores que los estudiantes comenten mientras aprenden a programar. Conocer esta información, creando un repositorio de errores que facilite el entendimiento de los mismos, y emplearlo como instrumento para generar feedback formativo fueron las grandes ideas que motivaron la realización del proyecto.

El prototipo COLMENA se basa en un modelo teórico, conocido como Modelo COLMENA, y una implementación real del mismo, de ahora en adelante el Sistema COLMENA. En el modelo teórico se definen las capas y tipos de operaciones que debe implementar todo sistema que esté orientado al trabajo con IDEs, errores, y cuya finalidad sea generar feedback sobre los mismos a estudiantes y profesores. En el sistema se describen un conjunto de herramientas funcionales e implementadas de acuerdo al estándar definido en el modelo (Figura 4).

3.1 El Modelo COLMENA

La necesidad de la definición de un modelo teórico surgió a partir de las primeras investigaciones en COLMENA [32, 33], donde se llevó a cabo una adaptación de indicadores a partir de la propuesta que Glanh hace de smart indicators dentro de entornos de aprendizaje convencionales [34]. Desde entonces, se ha redefinido para permitir la extracción de datos en un entorno de programación, siendo considerado el punto de partida para todos los experimentos llevados a cabo. El modelo que se propone cumple dos objetivos principales:

1. Ofrecer un sistema que monitoriza permanentemente el proceso de desarrollo de un estudiante, con el fin de recoger y almacenar la información sobre la interacción con el entorno, tomando como base los errores generados.

2. Proporcionar una herramienta de feedback formativo visual sobre la distribución, volumen y tipología de estos errores, para poder construir un feedback de valor sobre el mismo.

El modelo ofrece una ayuda a dos perfiles bien diferenciados en el mismo contexto de programación, y de los que se ha hablado tanto en la introducción como en el estado del arte: el estudiante y el profesor.

En el caso de los primeros, el propósito general de COLMENA es interactuar con el IDE tras los procesos de edición-compilación que éstos llevan a cabo. Ésta información será procesada y transformada por el sistema, generando un feedback formativo específico tanto para los segundos, completando así el flujo de comunicación entre el modelo y ambos tipos de usuarios.

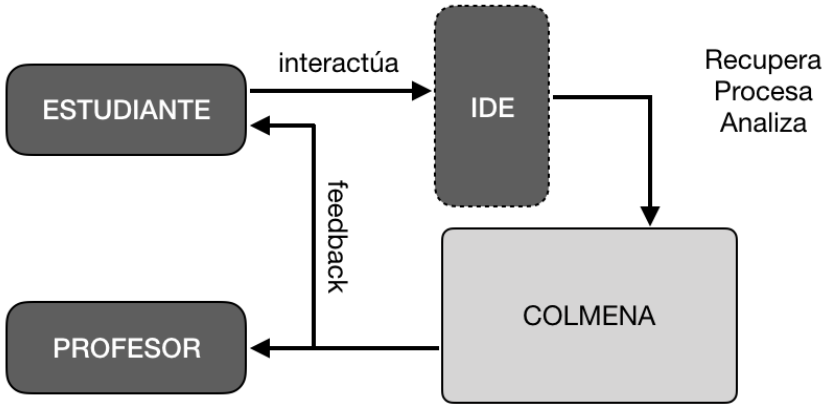


Figura 4. Interacción entre COLMENA, los estudiantes y los profesores.

El modelo de COLMENA describe todas las operaciones y estados que el posterior sistema que implemente dicho modelo debe contemplar. Estas comienzan en la interacción con el los errores y finalizan en la visualización e interpretación de los resultados. Para poder cubrir este conjunto de operaciones, el modelo se define en dos bloques principales:

1. El bloque de Monitorización abarca todos los procesos resultantes de la interacción de los estudiantes con el entorno de desarrollo, presuponiendo un IDE como herramienta de software.
2. El bloque de Respuesta se ocupa de mostrar al usuario (profesor y/o estudiante) el resultado del análisis de los datos obtenidos durante la Monitorización.

De forma más específica, estos dos bloques se subdividen en dos capas cada uno:

1. Capas de extracción y de procesamiento para el bloque de Monitorización.
2. Capa de análisis y de visualización para el bloque de Respuesta.

Estas cuatro capas permiten compartimentar las fases en las que la información va sufriendo transformaciones y facilitar su mantenimiento y escalabilidad (Figura 5).

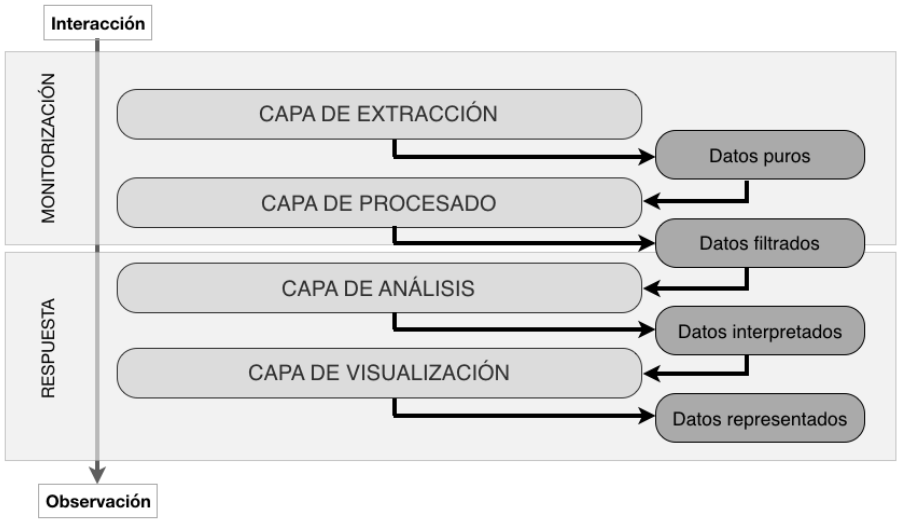


Figura 5. Adaptación del modelo de capas de Glahn a los IDEs: el Modelo COLMENA.

Todas las capas del modelo son independientes de la implementación que se lleve a cabo del mismo y del contexto real que se estudie y donde se vaya a implantar. Al tratarse de un modelo teórico, su objetivo es encajar en cualquier entorno en el que participan estudiantes, docentes y en el que se emplee una herramienta para el desarrollo de software, de la cual se pueda consultar algún tipo de registro de los sucesos que acontecen tras interactuar con la misma..

3.1.1 Capa de extracción

La capa de extracción, es la que se encarga de establecer una conexión con el entorno que se utilice para generar los errores de programación, que como se ha mencionado anteriormente, se presupone será un IDE. Conceptualmente estará distribuida de forma aislada y personalizada en los equipos de cada estudiante, obteniendo los datos brutos resultado del proceso de programación (Figura 6).

El objetivo de la capa de extracción es monitorizar los eventos de edición-compilación de un estudiante, para almacenar la información sobre la compilación en un sistema de persistencia. En el momento que un estudiante hace una modificación o una compilación del código, esta capa debe ocuparse activar los mecanismos apropiados para extraer datos y confeccionar un dataset con información que ofrecen los mensajes de error, como por ejemplo:

- Línea de código fuente donde se produce el error
- Proyecto en el que se produce el error.
- Nombre del fichero que se compiló.
- Ruta del fichero que se compiló.
- Código interno del error generado.
- Tipo de error generado.

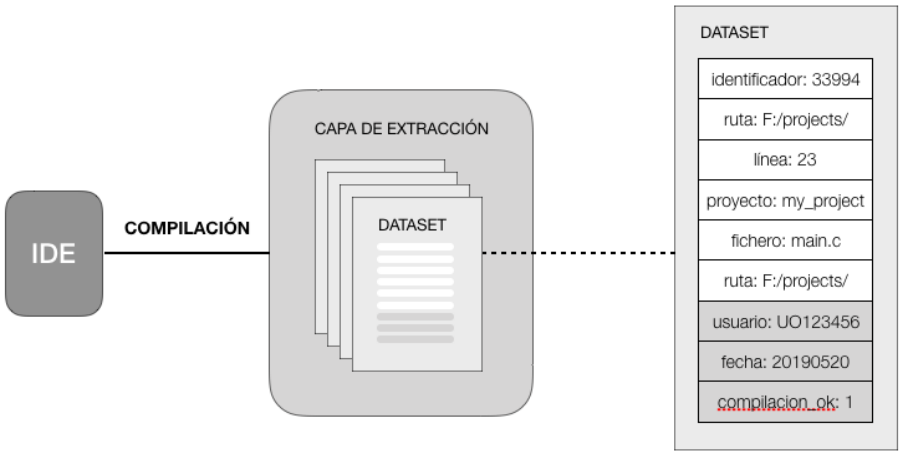


Figura 6. Representación de la interacción del IDE con los datasets generados durante la capa de extracción y el contenido de los mismos.

Durante la extracción de los errores, es ideal asociar parámetros adicionales y específicos al error, con el objetivo de completar cada registro con datos contextuales. De este modo, además de la información individual de cada error, se almacena también una información global de la compilación, de forma que se conozca exactamente si en dicha compilación hubo problemas o no. Por tanto, los parámetros adicionales se completan con la siguiente información:

- Identificador del estudiante que generó el error.
- Hora y fecha en que se produce el evento.
- Compilación exitosa si/no.

3.1.2 Capa de procesado

Tras la recuperación de los datos generados en la interacción con el IDE, la segunda capa se ocupa principalmente de la eliminación de redundancias en los errores detectados, omitiendo así aquellos que ya han sido previamente guardados y permanecen sin cambios. Muchos de estos errores aparecen porque el estudiante no se está focalizando en ellos, por lo que lo ideal es eliminarlos y considerar en cada evento de compilación exclusivamente aquellos errores relacionados con zonas de código donde

haya habido cambios. Establecer esta barrera hace que no aumente de forma desproporcionada la cantidad de errores recogidos, lo que podría hacer que quizás algunos errores generados tengan más importancia que la que realmente deberían tener (Figura 7).

Otro de los puntos clave de esta capa es la asociación y tipificación de los errores detectados. Dado que la variedad de errores detectados en cualquier lenguaje de programación es amplia, se considera importante realizar una categorización de los errores en una tipología que facilite a los usuarios finales, sean estudiantes o profesores, el entendimiento de los mismos. Además de la tipología del error según una familia concreta, es recomendable asociar parámetros de su contexto como la sesión y asignatura a la que pertenece. A partir de esta información y la contextualización de los errores con el ID y el timestamp realizada en la capa anterior, se puede hacer una contextualización "académica".

- Familia o tipo del error.
- Año académico en que se ha cursado una asignatura
- Asignatura sobre la que se va a asociar información
- Sesión práctica en la que se está trabajando.
- Grupo de prácticas de la asignatura

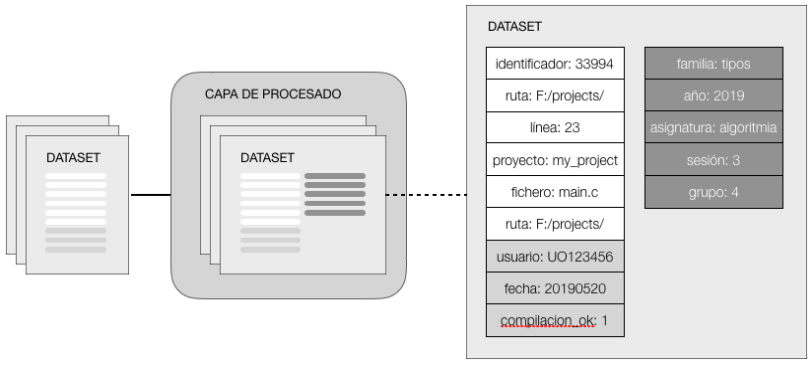


Figura 7. Ampliación de los datasets originales con información acerca del contexto de la asignatura y el curso.

3.1.3 Capa de análisis

Como primera capa del bloque de Respuesta, su principal objetivo es realizar una batería de análisis y transformaciones sobre los datos provenientes de la capa anterior, para confeccionar así la información que mostrará a estudiantes o profesores. Para conseguir dicho propósito, se debe analizar cada una de las variables obtenidas y estudiar qué nuevas variables transformadas se desean mostrar a partir de éstas sobre un estudiante, sesión o asignatura. Una vez decididas estas variables, esta capa realizará las operaciones necesarias para obtenerlas.

Bajo este criterio, se podrán obtener indicadores tanto de los errores acumulados a lo largo de las sesiones prácticas como específicos de un usuario o sesión. También se podrá mostrar información agrupando los errores en base a la familia o al tipo a los que pertenecen, y no únicamente de forma individual. Este análisis sobre los datos ofrece nuevos enfoques adicionales, como son por ejemplo comparativas de la situación concreta de un estudiante respecto al resto del grupo, o tener una visión global de los errores que más han aparecido.

En cuanto a un la conceptualización del análisis idóneo a realizar, se pueden considerar en términos generales las siguientes métricas:

- Métricas basadas en la familia de los errores, donde se contabilizarán los errores generados para cada una de las familias o tipos diferentes de error, tanto para una asignatura como para una sesión concreta dentro de la misma.

- Métricas basadas en el tiempo, en las que se analizará la cantidad de errores generados para cada familia a lo largo del transcurso de las sesiones de una asignatura.

- Métricas basadas en el usuario, donde para cada usuario se definirán los errores más frecuentes, tanto a nivel global, como de asignatura o sesión de la misma.

Estos ejemplos ilustran cómo a partir de unos datos enriquecidos en la capa de procesamiento, se pueden realizar análisis más complejos y más

cercanos al contexto en el que el usuario, sea docente o estudiante, está acostumbrado a interactuar, convirtiendo un conjunto de datos sin refinar en una información más fácil de consumir (Figura 8).

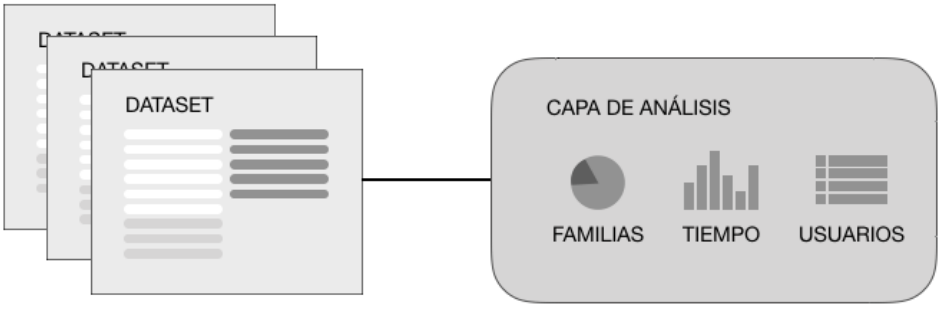


Figura 8. Representación de las funciones de la capa de análisis, donde se transforman los datos para una posterior visualización más entendible por parte del usuario, usando parámetros como las familias de error, tiempo o usuarios

3.1.4 Capa de visualización

En esta última capa el objetivo es representar gráficamente y de forma intuitiva las variables obtenidas en las capas anteriores, especialmente la resultante de los procesos llevados a cabo en la capa de análisis. La información estará disponible en diferentes paneles, confeccionando un layout que represente un "cuadro de mandos" (dashboard) adaptado a los roles del usuario objetivo, sea estudiante o profesor.

La herramienta incluirá información global sobre los datos analizados, como las asignaturas o sesiones de prácticas, desde donde se realizarán acciones variadas como conocer toda su información detallada o establecer comparaciones entre ellas. Además, también sería recomendable ofrecer información detallada sobre el estado de cada estudiante, así como información de sus errores comparados con el resto del grupo. Por último, también deberá dar información de ayuda extendida sobre las familias, los errores y las posibles formas de solucionarlos (Figura 9).

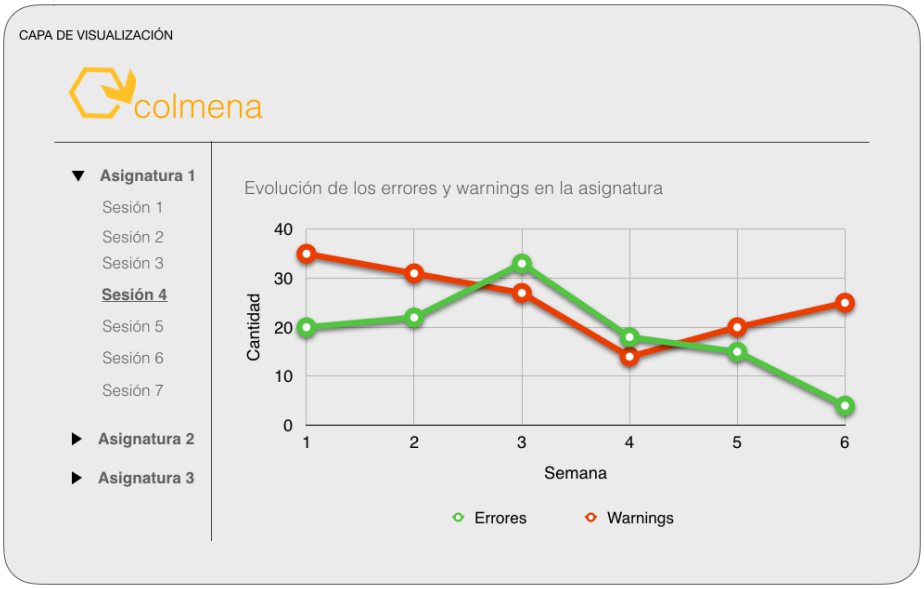


Figura 9. Representación gráfica de un posible feedback generado en la capa de visualización a partir de los datos analizados previamente.

En esta capa se deberán considerar también los mecanismos de notificación y retroalimentación desde los docentes a los estudiantes, de forma que estos reciban las actualizaciones de su estado a lo largo de las sesiones de una asignatura en la plataforma. Para conseguir un feedback formativo correcto y efectivo, es importante proveer a los estudiantes no solo de información sobre su estado, sino también de los errores producidos, ofreciendo la capacidad de conocerlos con más profundidad a través de la ayuda extendida de la herramienta (Figura 10).

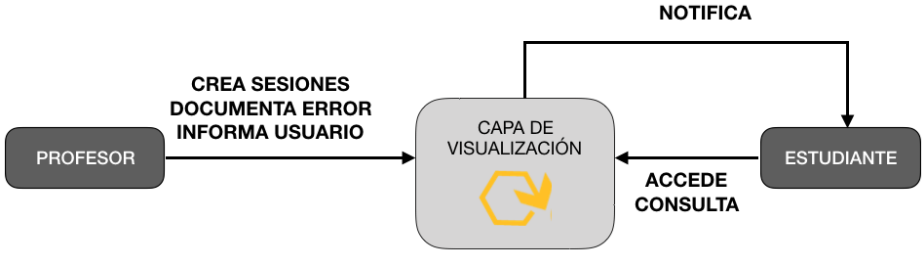


Figura 10. Flujo de comunicación entre los roles de usuario profesor.

3.2 El Sistema COLMENA

El modelo COLMENA no depende de una tecnología concreta ni una infraestructura determinada. Por tanto, al realizar cualquier posible implementación del mismo, se deben emplear los métodos y técnicas más apropiadas para el contexto donde se quiera implantar, analizando la arquitectura del sistema a desarrollar y considerando los problemas típicos que cualquier software puede padecer. En el contexto concreto donde se desarrolla esta investigación, el lenguaje de programación utilizado es Java, al igual que es Eclipse el IDE tomado como referencia, ya que ambos son empleados por los estudiantes durante las lecciones prácticas de las asignaturas de aprendizaje de programación. Por estas razones se ha diseñado un sistema orientado a Java y Eclipse. A esta implementación concreta del modelo COLMENA se la denominará, de ahora en adelante, el Sistema COLMENA.

El Sistema COLMENA es un sistema desarrollado para proporcionar respuesta a las preguntas de investigación planteadas en la introducción, y así constituye una herramienta de soporte utilizada por docente para dar feedback formativo. Una de las partes ha sido desarrollada como un plug-in para Eclipse, que se encarga de recopilar diferentes eventos de compilación de forma similar a trabajos como el de Luke o Jadud. Además, se basa en los principios de Learning Analytics [27] y también de la Interacción Persona Máquina (HCI) [98]. Así, desde la parte analítica se permite a los docentes estar al corriente de forma dinámica sobre la situación de sus estudiantes, mientras que desde la parte de HCI se ofrece un feedback más efectivo sobre los datos que devuelve el compilador, dividido en 3 capas fundamentales (Figura 11):

1. Proveer al estudiante (el programador) un mensaje breve sobre el problema.

2. Proveer una explicación detallada o ejemplos de los errores cometidos.

3. Proveer un nivel de feedback formativo mayor, que contenga potenciales acciones de corrección.

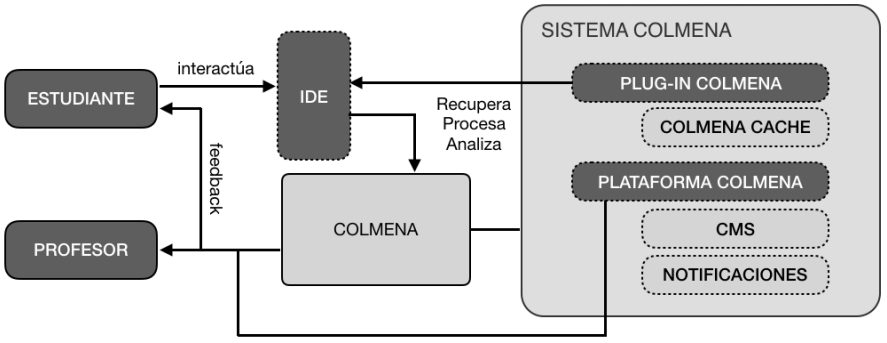


Figura 11. Diagrama inicial de la interacción con el Sistema COLMENA enriquecido con las herramientas desarrolladas para tal efecto.

3.2.1 Arquitectura del sistema COLMENA

El Sistema COLMENA está compuesto por un conjunto de aplicaciones que interactúan entre sí y con los usuarios estudiante y profesor a lo largo del ciclo de enseñanza y aprendizaje (Figura 12):

- **COLMENA Plug-in**: se trata de un plug-in para Eclipse que identifica, detecta los cambios en el código y recolecta los errores producidos en tiempo de compilación, para posteriormente persistirlos en un sistema de persistencia, tales como una base de datos relacional o un fichero.

- **COLMENA Management**: un gestor de contenidos que permite la creación, borrado y edición de asignaturas, calendarios académicos y los temas de las mismas. Además facilita la asociación de los errores capturados en las sesiones prácticas de cada asignatura.

- **COLMENA Platform**: aplicación web que proporciona el feedback en tiempo real sobre los errores almacenados por COLMENA Plug-in y la cual ha sido desarrollada con el objetivo de proveer feedback formativo inmediato y efectivo.

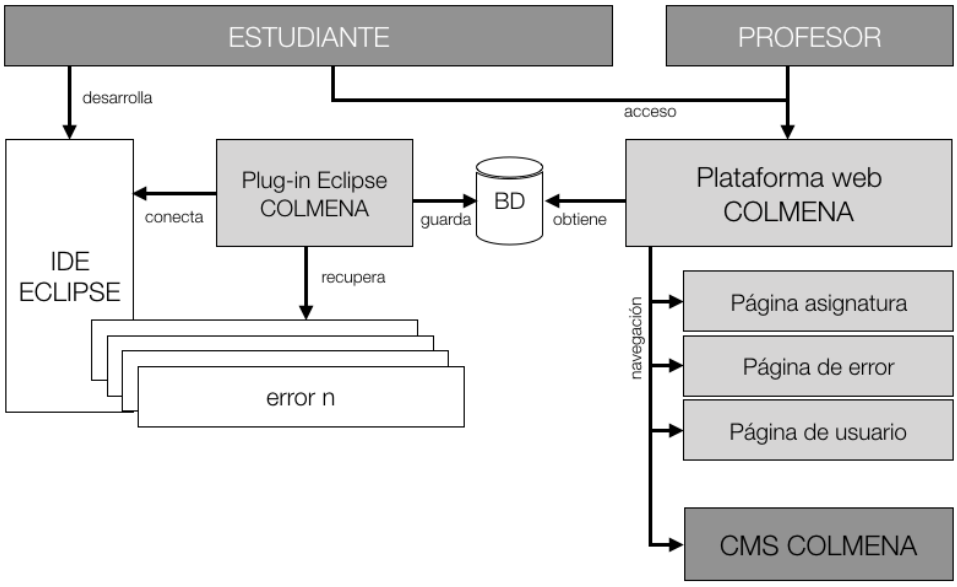


Figura 12: Diagrama de la arquitectura del Sistema COLMENA, donde se representa las acciones de los distintos roles de usuario para con los distintos componentes del sistema.

A través del Sistema COLMENA se da una implementación completa al modelo descrito en la sección anterior, proponiendo una arquitectura orientada a integrarse de forma transparente con el IDE y a su vez compuesta de una aplicación web disponible desde cualquier equipo con acceso a internet y un navegador web. Estos tres sistemas están involucrados así en el proceso completo de aprendizaje de la programación, que comienza con un estudiante atendiendo a una clase de prácticas a desarrollar código y finaliza con el feedback formativo que el profesor proporciona al estudiante a partir de los errores generados previamente (Figura 13).

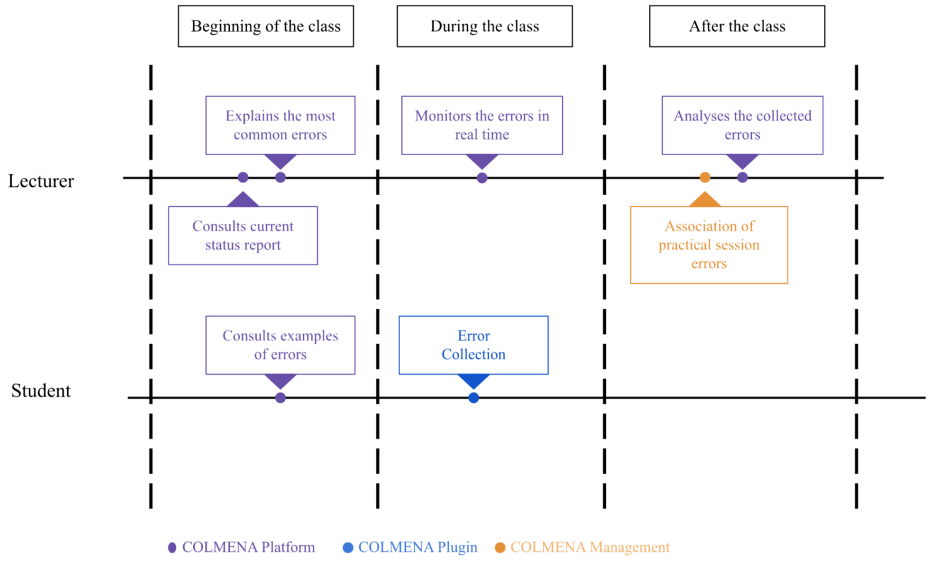


Figura 13. Línea de tiempo representando las interacciones entre el estudiante y el profesor con los diferentes componentes del Sistema COLMENA.

3.2.2 COLMENA Plug-in: detección y captura de errores

La primera de las herramientas desarrolladas ha sido COLMENA Plug-in, con el objetivo de conseguir una integración con el IDE, y así detectar el guardado o la compilación de un fichero de código fuente. Dentro del Modelo COLMENA, el plug-in abarca las operaciones que se realizan dentro de las capas de extracción y procesado, sin intervenir en sus labores de programación. En líneas generales, el plug-in analiza el conjunto de compilaciones que hay por cada fichero que el estudiante intenta compilar durante una sesión, contando la aparición de los errores en cada bloque de código [105].

A pesar de que en sus versiones iniciales únicamente era posible ejecutar el proceso de recopilado de errores de forma manual, del mismo modo que cualquier otra acción similar del IDE (depurar, compilar, guardar, etc.), se convirtió en una necesidad hacer que el plug-in fuese independiente, funcionando en segundo plano y de forma transparente, sin requerir ninguna acción manual, aunque permite también activar y desactivar la funcionalidad de forma manual.

La mecánica de funcionamiento del plug-in es la siguiente: se inicializa automáticamente y se ejecuta en cada evento de compilación o guardado, ya que son considerados como los momentos en el que un estudiante ha terminado de escribir o de modificar el código. El ciclo de vida del plug-in comienza cuando el usuario comienza a desarrollar código en el IDE Eclipse y guarda o compila el fichero que está editando. En este proceso de "edición-compilación", el plug-in se encarga de recolectar los errores existentes, analizarlos y persistirlos, en caso de que sea necesario (Figura 14).

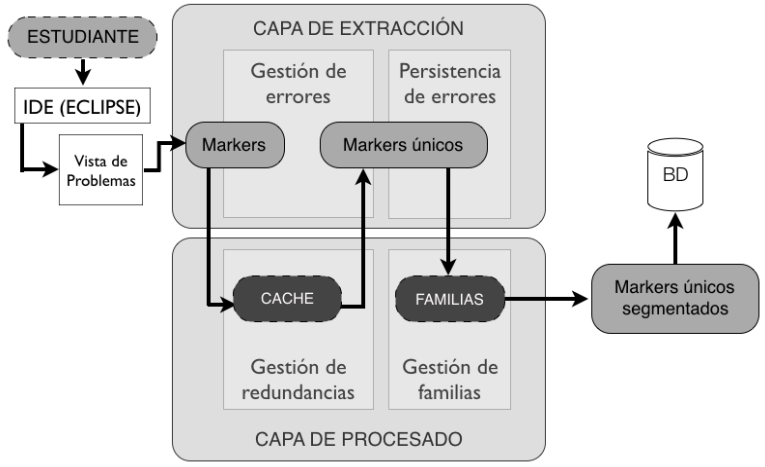


Figura 14. Desglose de las operaciones de las capas de extracción y procesado, realizadas por COLMENA Plug-in

Para conocer estos errores o warnings, el plug-in accede a una de las vistas por defecto de Eclipse, a la que se conoce como la vista de problemas. En esta vista se refleja cada error o warning, definidos como "marker", según la terminología de Eclipse [16]. De cada marker se identifican unos parámetros básicos como el código y tipo del error (warning o error), la ruta relativa, línea del código fuente donde se ha producido.

3.2.2.1 Diseño e implementación de *ColmenaCache*

Para facilitar el posterior análisis de errores se considera importante guardar solamente una vez cada ocurrencia de error siempre y cuando no

se repita de forma consciente. De este modo, si un usuario modifica el código donde hay un error y éste no se ha subsanado, este error sí será considerado para su posterior persistencia. En caso contrario, es ignorado, ya que se considera que el error se ha guardado previamente en su primera aparición, y se presupone que el usuario no está prestando atención a esa zona del código fuente para resolver el error.

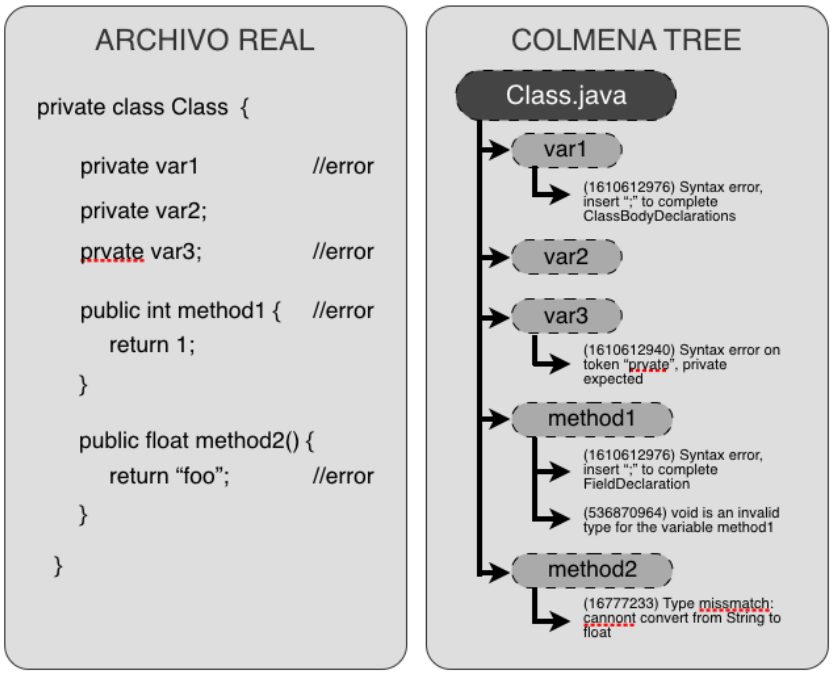


Figura 15. Paralelismo entre código fuente real cel ColmenaTree.

Para detectar esta redundancia de errores se desarrolló una funcionalidad incorporada en el plug-in, denominada ColmenaCache. Su objetivo es descartar todos los errores o warnings sobre los que el estudiante no está actualmente trabajando pero que, en cada compilación o edición, se siguen registrando igualmente.

Para realizar estas operaciones, ColmenaCache utiliza la estructura interna de Eclipse para analizar el código fuente generado, y hace uso de ella para construir un árbol, que se conoce por ColmenaTree, donde están

representados todos los elementos estructurales del código fuente, similar al árbol AST que estructura los lenguajes de programación. De esta forma, cada vez que se guarde o compile el proyecto en el IDE, ColmenaCache recorrerá la estructura del código actual, creando un nuevo árbol y comparándolo con su versión anterior, y seleccionando únicamente los errores de aquellos elementos del árbol que son realmente nuevos (Figura 15).

3.2.2.2 Enriquecimiento del dataset con parámetros contextuales

Una vez filtradas y eliminadas las redundancias y posibles duplicidades a través de ColmenaCache, El plug-in se ocupará de enriquecer los markers resultantes con información relevante acerca de su contexto de programación. Siguiendo la misma línea sugerida por otros autores de acuerdo al trabajo con grupos de estudiantes, es esencial para un análisis posterior clasificar los datos en base a la asignatura y el grupo para permitir a los profesores un estudio e interpretación en mayor profundidad. Para llevar a cabo este objetivo, el sistema ha de transformar la información inicial recogida en cada capa del sistema en una versión más refinada y entendible del mismo. Como datos adicionales se aportarán el timestamp de la detección del error, la asignatura que se está impartiendo, la sesión concreta dentro de la misma, el grupo al que pertenece el usuario y su identificador. También se completará el dataset con información sobre si hubo errores en esa compilación o no.

3.2.2.3 Definición y asociación de familias a los errores y warnings

Tras añadir información relacionada con el contexto académico del usuario, el plug-in realizará una asociación del error a una familia de error concreta, organizándose bajo un criterio. Esta práctica no es una aportación novedosa, ya que existen en la literatura científica varios trabajos sobre la clasificación y tipología de los errores en programación. Ben-Ari estableció una clasificación de los errores más comunes en Java basándose principalmente en tres grandes bloques: sintácticos, estructurales y semánticos [74]. Hristova et al agruparon los 20 errores más relevantes seleccionados por profesores y expertos de un total inicial de 62 errores y usaron su propia clasificación en 3 grupos: sintácticos,

semánticos y lógicos [44]. McCall y Kölling desarrollan y validan sus propias categorías centrándose más en los errores propiamente que en los mensajes de error. Para validarlas, proponen una metodología basada en el acuerdo entre investigadores los cuales analizan de manera manual el código de cada estudiante. Su resultado está en la misma línea que las investigaciones previas, dando lugar a errores de tipo semántico, sintáctico y de lógica, los cuales no llegan a explicar en detalle [71]. Por otra parte, Ettles et al consideran nuevamente su propia clasificación, utilizando como categorías los errores algorítmicos y de más entendimiento para los errores que detectan en su investigación [21, 30].

De forma análoga, COLMENA también clasifica cada error o warning en una familia concreta de una lista confeccionada y personalizada a partir de la tipología interna que Eclipse utiliza para agrupar a sus propios errores, y que se define internamente en la interfaz iProblem.java de su core[10]. Para esto, se toma como referencia el identificador único del error y la referencia a la familia que Eclipse le asocia. La razón del porqué se elige la clasificación interna que utiliza Eclipse es debido a que, tras estudiar las diferentes taxonomías, Eclipse ha implementado su propio compilador, denominado Eclipse Compiler for Java (ECJ) (Tabla 1). Esto permite a Eclipse devolver una clasificación de errores más completa e independiente de los errores que devuelve el JDK en la versión específica, lo que aporta una ventaja a al análisis ya que las clasificaciones propuestas en por Jadud [49] o Jackson et al [46], están condicionadas por las diferentes versiones del compilador usado para hacer los estudios [71].

Nombre de la familia	Variables relacionadas	Descripción y causas del mismo
Tipos (Type)	type_warnings type_errors	Errores relacionados con los tipos del lenguaje Java
Sintaxis (Syntax)	syntax_warnings syntax_errors	Errores relacionados con la sintaxis del lenguaje de programación Java

[10]https://help.eclipse.org/neon/index.jsp?topic=%2Forg.eclipse.jdt.doc.isv%2Freference%2Fapi%2Forg%2Feclipse%2Fjdt%2Fcore%2Fcompiler%2FIProblem.html

Estructural (Structural)	structural_warnings structural_errors	Errores en la estructura y organización de las clases en Java
Variables (Variable)	variable_warnings variable_errors	Errores relacionados con el uso de las variables
Métodos (Method)	method_warnings method_errors	Errores relacionados con la estructura de los métodos
Constructores (Constructor)	constructor_warnings constructor_errors	Errores sobre el constructor y su uso
Imports (Import)	import_warnings import_errors	Errores relacionados con la importación de paquetes y librerías externas

Tabla 1: Tipología de errores según la taxonomía basada en la clasificación interna de Eclipse y otros estudios.

3.2.2.4 Persistencia del dataset

Finalizada la recolección de información en la capa de extracción y su enriquecimiento en la capa de procesado, el dataset final de parámetros queda confeccionado para su persistencia (Tabla 2). Tal y como se aprecia, gracias a los mecanismos de selección y cribado de información del plug-in, y al procesado de otras variables contextuales, se ha logrado evolucionar de un dataset estrictamente relacionado con el error a un conjunto de parámetros enriquecidos que ofrecen la posibilidad de realizar un amplio espectro de estudios y trabajos.

Datos Brutos		Capa de Extracción		Capa de Procesado	
Nombre	Descripción	Nombre	Descripción	Nombre	Descripción
project	Proyecto donde aparece el error o warning	user_id	Identificador del usuario	family	Familia asociada al error

path	Ruta relativa al fichero .java donde está ubicado el error o warning	timestamp	Fecha y hora en que se produjo el error o warning	year	Curso académico en que se está realizando las tareas de programación
error_id	Identificador interno (de Eclipse) del error o warning	compilations_errors	Number of compilation with errors	subject	Asignatura dentro de la cual se realizan las tareas de programación
line	Línea del código donde se produjo el error o warning			session	Sesión concreta de prácticas dentro de la asignatura
				group	Grupo de prácticas al que pertenece el estudiante

Tabla 2. La información de las variables capturadas por el sistema COLMENA en la implementación de las primeras capas del modelo, correspondientes al Bloque de Monitorización (capa de extracción y capa de procesado)

Con el dataset enriquecido con todos los nuevos parámetros, la última de las tareas que debe realizar el plug-in es la asociada a la persistencia de los datos. El plug-in ofrece realizar la persistencia en un fichero local de log o también guardado en una base de datos MySQL en un servidor remoto, evitando así tener que realizar las acciones de importación de forma manual. Además, permite guardar a través de FTP el fichero de log generado. Todas estas opciones, así como más parámetros de configuración están disponibles en la vista de preferencias del plug-in, accesible a través del menú de preferencias y configuración de Eclipse.

3.2.3 COLMENA Platform: visualización de errores

Para la implementación de las dos últimas capas del modelo es necesario desarrollar algún tipo de solución software capaz de realizar los análisis y la transformación del dataset para que los datos del mismo puedan ser representados de forma amigable en una interfaz gráfica de usuario. A diferencia de las capas de Extracción y Procesado, con una mayor independencia entre sí, en el caso de las capas del Bloque de Respuesta existe una gran vinculación entre las mismas, debido a que esencialmente la capa de Análisis se encargará de ejecutar las operaciones y agrupaciones necesarias sobre el dataset para representarse gráficamente en la capa de Visualización.

3.2.3.1 Antecedentes: implementación integrada en el IDE, *ColmenaView*

Las primeras muestras de integración de elementos gráficos en algún tipo de interfaz surgió de la mano de las versiones iniciales del plug-in [32], debido a la necesidad de representar los primeros análisis que se realizaban sobre el código fuente. A nivel de diseño, se concibió un panel integrado en el IDE Eclipse en el que se pudieran visualizar métricas sobre el DataSet obtenido (Figura 16).

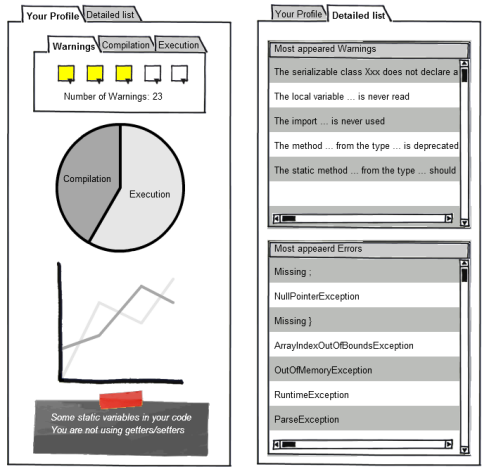


Figura 16. Wireframe que ilustra la tipología de datos a mostrar en un hipotético panel asistencial integrado en el IDE.

A la hora de realizar la implementación de la vista, se decidió por desarrollar un panel exclusivo dentro de Eclipse, llamada ColmenaView. Este panel, al igual que el resto de paneles y pestañas del IDE, es configurable y permite ser colocado en cualquier lugar del entorno, por su flexibilidad a la hora de ajustar la interfaz a gusto del desarrollador. En cuanto a las variables incluidas dentro del panel, se eligieron una serie de parámetros relacionados con el dataset obtenido, como la media de errores de ese usuario, su desviación típica con respecto al resto de estudiantes o los errores más frecuentes. También se muestran parámetros relacionados con el segmento donde se encuentra el estudiante en función del total de errores y warnings cometidos.

3.2.3.2 COLMENA Platform como aplicación web

A partir del desarrollo de ColmenaView surgió la necesidad de centralizar el contenido y los paneles en algún tipo de solución que no fuera tan limitada ni dependiente del IDE, y que estructurara mejor los indicadores gráficos para que tuvieran un mayor entendimiento. Las limitaciones que los paneles de Eclipse ofrecen dificultan este enfoque, además de ofrecer información demasiado individualizada y sin posibilidades de navegación.

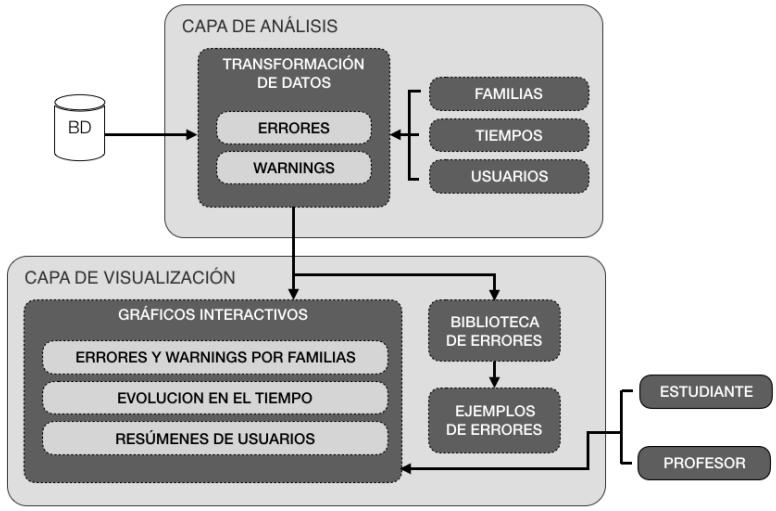


Figura 17. Distribución de elementos que intervienen en COLMENA Platform.

Debido a su comodidad y facilidad de uso por encima de otro tipo de aplicaciones software, se tomó la decisión de desarrollar una aplicación web, al no tener dependencia del sistema operativo y no necesitar un trabajo adicional de instalación y configuración (Figura 17).

El propósito de la aplicación se extiende desde la versión original de ColmenaView, la cual era más orientada al estudiante, al estar integrada en el IDE. A través de COLMENA Platform se desea que el profesor, durante y después de una sesión de prácticas, pueda evaluar la tipología de los errores con el objetivo de ser capaz de explicar durante la clase y las siguientes sesiones, si fuera necesario, las diferentes causas y razones por las cuales se produce el error, que el contexto en el que se desarrolla esta investigación es el ya citado en numerosas ocasiones feedback formativo. En vista de esto, se sugieren dos ejemplos de escenarios beneficiosos para el profesor al utilizar la herramienta:

1. Un repositorio de errores que puede ser analizado detalladamente desde una sesión hasta la siguiente. En este escenario, si no hay tiempo suficiente para solucionar todos los errores dentro (al final) de una sesión, por cualquier que sea la razón, es posible empezar la siguiente sesión solucionando y revisando los errores de la anterior sesión, a modo de recordatorio.

2. Tener una situación actualizada de los errores que han ocurrido en cada sesión práctica. En este escenario, el profesor tiene la capacidad de ver los errores más frecuentes y la manera de solucionarlos. Si hubo errores específicos por la tipología de la sesión, el profesor puede buscar una forma particular de solucionarlos y transmitir esta información al grupo.

En ambos escenarios el contexto o fragmento de código donde el error ha acontecido está presente y se tiene en cuenta, lo que hace que en este caso el feedback si sea orientado específicamente al error (specific-error), además de inmediato, positivo y correctivo. En esta línea, Prather et al [79] ya demostraron en su momento la efectividad de las herramientas de feedback más allá de las propias herramientas de desarrollo, como es COLMENA en este caso.

El acceso a COLMENA Platform es a través de un login (usuario y contraseña) y ofrece pantallas de diferente tipología (Tabla 3) cumpliendo con visualización de errores de forma similar a Jadud y Luke [48, 62]. Otra característica a destacar es que se da importancia a los 3 primeros errores y a los 3 errores más frecuentes, siguiendo las estrategias de facilitar el recuerdo de los mismos al profesor, como indica Gobet et al [35].

Información sobre la asignatura
Visualización de la información general del curso
Visualización de las sesiones que componen curso
Visualización del top de errores
Visualización de los estudiantes del curso
Información sobre la sesión práctica
Visualización de la información general de la sesión y comparar sesiones
Visualización de la información de los errores de la sesión
Visualización de los estudiantes de la sesión
Información sobre estudiantes
Visualización de las asignaturas en las que participó un estudiante
Visualización de los errores de un usuario
Visualización de los 3 errores más frecuentes y por sesión
Visualización de los 3 errores del estudiante en la sesión práctica
Información sobre los errores
Visualización sobre el error y sus ejemplos
Creación de un nuevo ejemplo

Tabla 3. Resumen de funcionalidades de COLMENA Platform.

Otra funcionalidad importante y realmente útil para los usuarios objetivo, es el desarrollo de roles de acceso. Disponer de roles permite que en función de los mismos (estudiante, profesor y administrador) la plataforma facilite una navegación u otra. Así, las pantallas para los profesores pueden estar enriquecidas con contenido específico que facilite que el feedback sea más contextual e inmediato, permitiendo así que también sea más task-specific (Tabla 4).

	Estudiante	Profesor	Administrador
Listado de errores	Acceso total	Acceso total	Acceso total
Ficha del error	Acceso total	Acceso total	Acceso total
Listado de estudiantes	NO	Los participantes en las asignaturas asociadas	Acceso total
Ficha de estudiante	La propia del estudiante	Los participantes en las asignaturas asociadas	Acceso total
Ficha de asignatura para el estudiante	Las propias del estudiante	Los participantes en las asignaturas asociadas	Acceso total
Acceso al listado de asignaturas	Solo las suyas	Todas las asignadas	Acceso total
Acceso a la ficha de asignatura	NO	Todas las asignadas	Acceso total
Acceso a la ficha de sesión	NO	Las incluidas en las asignaturas asociadas	Acceso total
Comparación de fichas de sesiones	NO	Las incluidas en las asignaturas asociadas	Acceso total

Tabla 4. Roles que los usuarios tienen en la aplicación y los permisos de acceso a las distintas secciones de COLMENA Platform.

Para ofrecer un feedback formativo, son especialmente destacables algunas de las pantallas de COLMENA Platform. Una de las secciones más importantes de la aplicación es la ficha del error desde donde el profesor puede indicar a los estudiantes ejemplos de solución para un error (Figura 18).

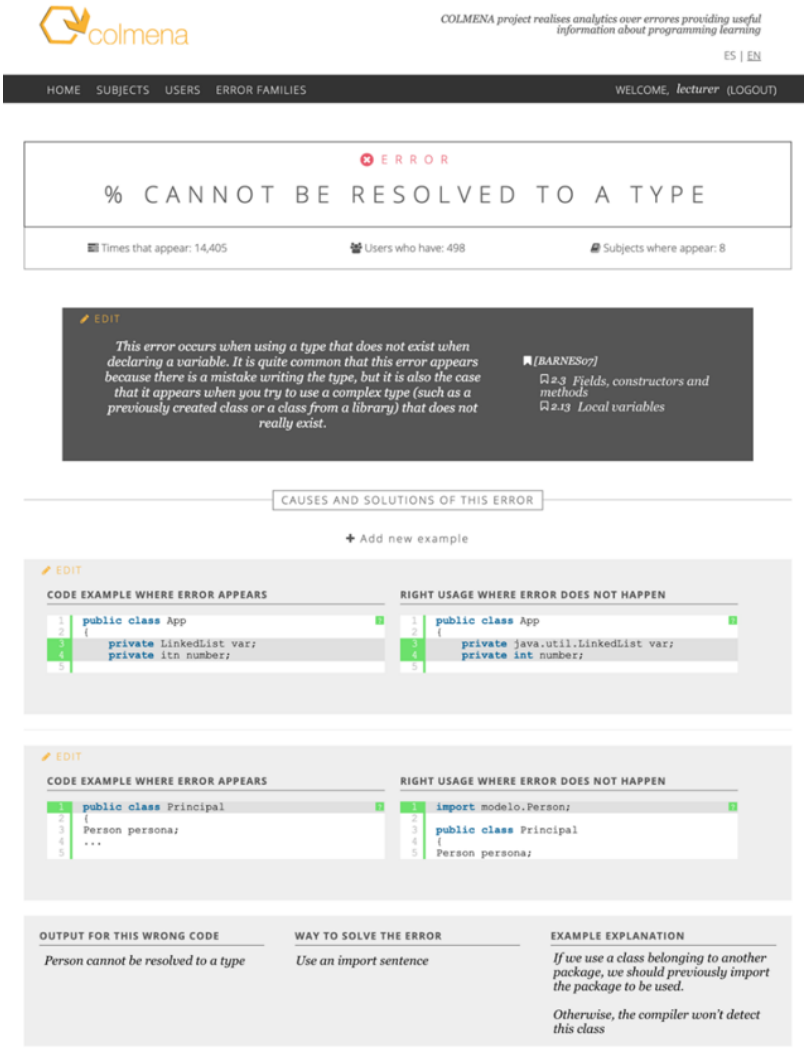


Figura 18. Pantalla con los ejemplos de un error mostrada al profesor al principio de la sesión práctica.

Otra sección interna de la aplicación web que es de especial relevancia es la sección correspondiente al listado de errores, donde un profesor puede consultar la totalidad de errores producidos de un tipo concreto, conociendo así en tiempo real si es un error muy frecuente en una sesión concreta (Figura 19).

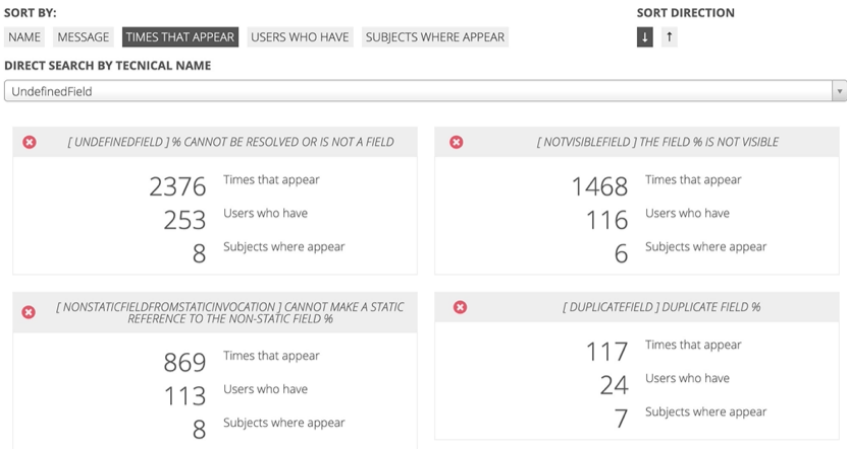


Figura 19. Lista de errores de un tipo específico disponibles para la consulta del profesor.

3.2.4 COLMENA Management: gestor de contenidos web

A partir de las últimas versiones de COLMENA Platform se incorporó también un gestor de contenidos (CMS) web. El CMS posee una interfaz amigable y su propósito es la definición de asignaturas, horarios, sesiones de prácticas, así como la carga de usuarios. También facilita las operaciones de vinculación de errores con relación a su momento de captura a través del plug-in.

4. Análisis de los errores de los estudiantes en contextos formativos

4.1 Análisis de datos de forma estática y estudio de la distribución de los estudiantes

El primer enfoque experimental surgió de la idea de conocer qué tipo de distribución seguían los estudiantes de programación, para así dar una respuesta a la research question 1. A partir de la experiencia como docentes y en las distribuciones que de forma convencional siguen los grupos de estudiantes, se consideró que los estudiantes, en cuanto a la generación de errores, estaban siguiendo un modelo de distribución normal.

4.1.1 Metodología

Para efectuar la extracción de los errores y warnings de cada proyecto, se utilizó una de las primeras versiones del Sistema COLMENA, compuesta por una versión inicial del plug-in para Eclipse, la cual ofrecía la posibilidad de analizar de forma síncrona, y no en tiempo real, el código fuente del proyecto sobre el que se ejecutaba en busca de errores y/o warnings. El resultado de esta búsqueda se almacenó en un fichero de texto plano sobre el que no se efectuó ningún procesamiento adicional

4.1.2 Muestra

Para conocer si realmente los estudiantes seguían dicha distribución, se llevó a cabo el primer caso de estudio con datos reales, que tuvo lugar a finales del segundo semestre del curso 2010/2011. Concretamente fueron analizadas 75 prácticas de estudiantes de la asignatura de Estructura de Datos y de la Información, en la que se trabajan conceptos avanzados de programación como el uso de colecciones, algoritmos de ordenación, y análisis de la complejidad de ciertos algoritmos (Tabla 5).

Para este caso de estudio se analizaron los proyectos correspondientes a la entrega final de la parte práctica de la asignatura, ya finalizados por

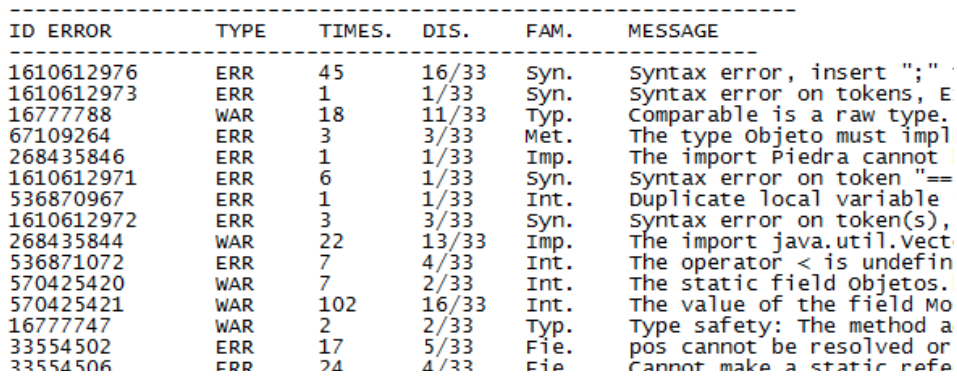
parte de los estudiantes y listos para ser evaluados por el personal docente. Por este motivo, la mayoría de los proyectos carecían de errores de programación, teniendo únicamente warnings sin resolver.

Curso	Sesiones consideradas	Tamaño de la muestra
2010/2011	1	75

Tabla 5. Sesión y usuarios considerados para el estudio puntual.

4.1.3 Análisis de los datos

De los datos obtenidos en este formato fue necesario realizar una transformación para así obtener una información más refinada y resumida. Para conseguirlo, se desarrolló un módulo adicional en el Sistema COLMENA, con la función de interpretar los datos brutos, haciendo un recuento y unas estadísticas generales sobre el total de los errores. A este sistema ad-hoc se le conoce como COLMENA Analyzer, una aplicación Java independiente de lo que es hoy en día la capa de análisis del Modelo COLMENA (Figura 20).



ID ERROR	TYPE	TIMES.	DIS.	FAM.	MESSAGE
1610612976	ERR	45	16/33	Syn.	Syntax error, insert ";"
1610612973	ERR	1	1/33	Syn.	Syntax error on tokens, E
16777788	WAR	18	11/33	Typ.	Comparable is a raw type.
67109264	ERR	3	3/33	Met.	The type Objeto must impl
268435846	ERR	1	1/33	Imp.	The import Piedra cannot
1610612971	ERR	6	1/33	Syn.	Syntax error on token "==
536870967	ERR	1	1/33	Int.	Duplicate local variable
1610612972	ERR	3	3/33	Syn.	Syntax error on token(s),
268435844	WAR	22	13/33	Imp.	The import java.util.Vect
536871072	ERR	7	4/33	Int.	The operator < is undefin
570425420	WAR	7	2/33	Int.	The static field Objetos.
570425421	WAR	102	16/33	Int.	The value of the field Mo
16777747	WAR	2	2/33	Typ.	Type safety: The method a
33554502	ERR	17	5/33	Fie.	pos cannot be resolved or
33554506	ERR	24	4/33	Fie.	Cannot make a static refe

Figura 20. Captura de pantalla de los resultados devueltos por COLMENA Analyzer tras analizar los errores de una sesión concreta.

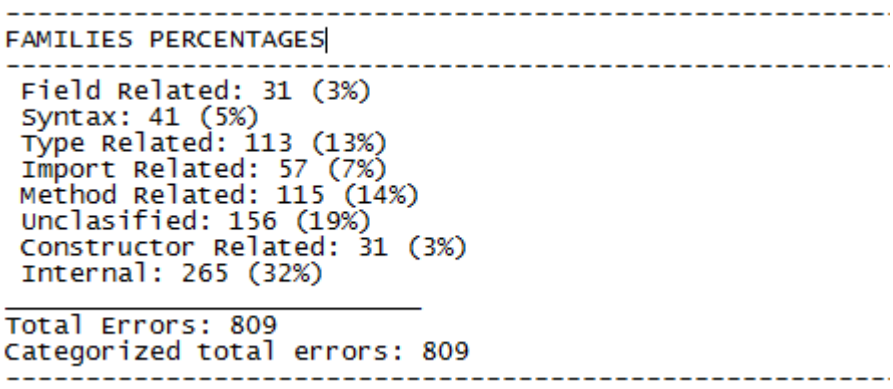


Figura 21. Resumen del total de errores detectados con COLMENA Analyzer.

A partir de la salida devuelta por COLMENA Analyzer, mucho más entendible y fácil de interpretar, se estudió la cantidad de errores que cada estudiante genera, agrupando a los estudiantes por niveles teniendo en cuenta la media ($\overline{X}$) y la desviación típica (σ) de los errores cometidos (Figura 21). El objetivo de esta agrupación en base a la media y la desviación típica es conseguir ver la similitud de los grupos generados y su tamaño con respecto a la anteriormente citada distribución normal.

Nivel	Rango
Inicial	$> \overline{X} + 2\sigma$
Básico	$(\overline{X} + 2\sigma)$ - $(\overline{X} + \sigma)$
Medio	$(\overline{X} + \sigma)$- 0
Avanzado	0- $(\overline{X} - \sigma)$
Experto	$(\overline{X} + \sigma)$- $(\overline{X} + 2\sigma)$
Muy Experto	$< \overline{X} + 2\sigma$

Tabla 6. Grupos definidos a partir de la ubicación del estudiante dentro de la distribución normal basándose en la media de errores.

Tras recolectar la información sobre los estudiantes en una sesión de una asignatura de segundo curso del Grado en Ingeniería Informática del Software, se analizaron los datos para ver si encajan con un modelo de distribución normal. En caso de que los estudiantes siguieran un modelo de distribución normal, el grupo podría segmentarse en diferentes grupos en función de la media (X) y la desviación típica (σ), parámetros habituales a la hora de segmentar una distribución de este tipo (Tabla 6).

4.1.4 Resultados

4.1.4.1 RQ 1. ¿Se comportan los estudiantes de acuerdo a una distribución normal en función a los errores y warnings?

Al representar los estudiantes en un gráfico de columnas (Figura 22), se forman 5 grupos bien diferenciados. Respecto al comportamiento de los estudiantes, analizando los resultados de la gráfica, se aprecia que el grupo se comporta siguiendo una distribución normal.

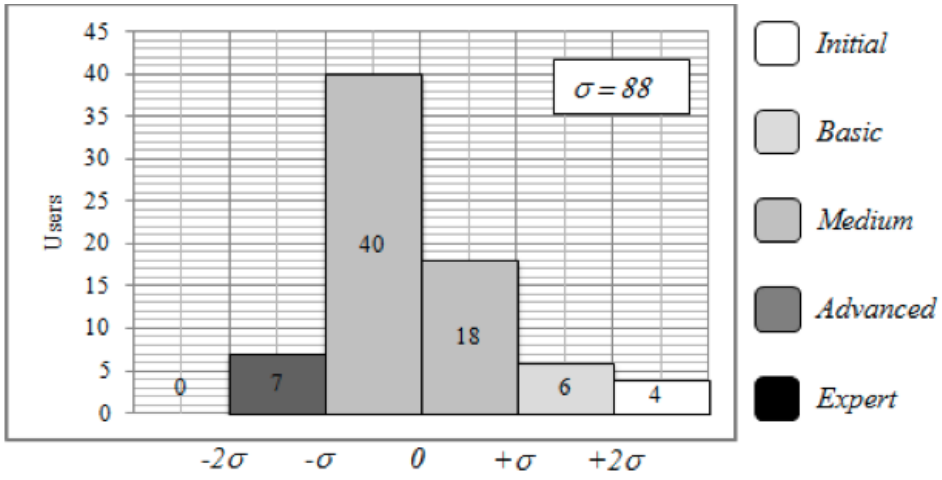


Figura 22. Distribución de los estudiantes en grupos para el caso de estudio.

4.1.5 Discusión

4.1.5.1 RQ 1. ¿Se comportan los estudiantes de acuerdo a una distribución normal en función a los errores y warnings?

Los 75 usuarios de la muestra se dividen de una forma bastante homogénea entre los grupos, donde se puede apreciar que más del 50% de los estudiantes se concentra en el grupo Avanzado (Advanced). Este hecho tiene lógica, ya que se trata de un grupo de estudiantes de segundo curso de Ingeniería Informática, donde ya hay un dominio bastante elevado del lenguaje, produciéndose menos errores. Para este caso en particular se produce uno de los grupos, el más experto (Muy experto o High Expert), se encuentra desierto. No es frecuente encontrar estudiantes muy expertos en programación en asignaturas donde realmente aún se les está enseñando a hacer uso de los algoritmos de búsqueda, ordenación o de estructuras de datos más complejas.

Si se observan los grupos en más profundidad, se puede estudiar la relación de estudiantes en cada grupo con respecto a los porcentajes ideales de una distribución Normal (Tabla 7).

Segmento de pertenencia de los estudiantes	Porcentaje ideal distribución normal	Porcentaje obtenido con COLMENA Plug-in
$\overline{X} \pm \sigma$	68,3%	77,33%
$\overline{X} \pm 2\sigma$	27,2%	17,33%
$\overline{X} \pm 3\sigma$	4,2%	5,33%

Tabla 7. Comparación de los valores estándar de una distribución normal frente a los datos reales obtenidos.

Tal y como ilustra la tabla, el grupo $\overline{X} \pm \sigma$ tiene un tamaño sensiblemente mayor que en el modelo ideal, mientras que $\overline{X} \pm 2\sigma$ posee un tamaño menor. Este fenómeno puede deberse a que el grupo genera un número de errores muy similar, lo que hace que se concentren más en los

grupos medio y avanzado. También se debe tener en cuenta que en este caso de estudio se recopilaron únicamente los errores de tipo warning, y una vez finalizado el ejercicio práctico, por lo que influye también el grado de valor que los estudiantes le den a los warnings, con respecto a los errores. En ocasiones los estudiantes no les dan importancia y los desprecian sin atenderlos, caso que supone un incremento de warnings a la hora de analizar las prácticas de este tipo de estudiantes. Para llevar a cabo un análisis comparativo, se ejecutó una herramienta de terceros, PMD[11], cuya finalidad es también analizar el código de forma estática en busca de variables no usadas, bloques try-catch que nunca se ejecutarán o que están vacíos, objetos inicializados nunca utilizados, etc. Tras ejecutar la herramienta con los 75 estudiantes de la muestra, los resultados se distribuyeron en un gráfico de columnas, del mismo modo que previamente se hizo para COLMENA Plug-in.

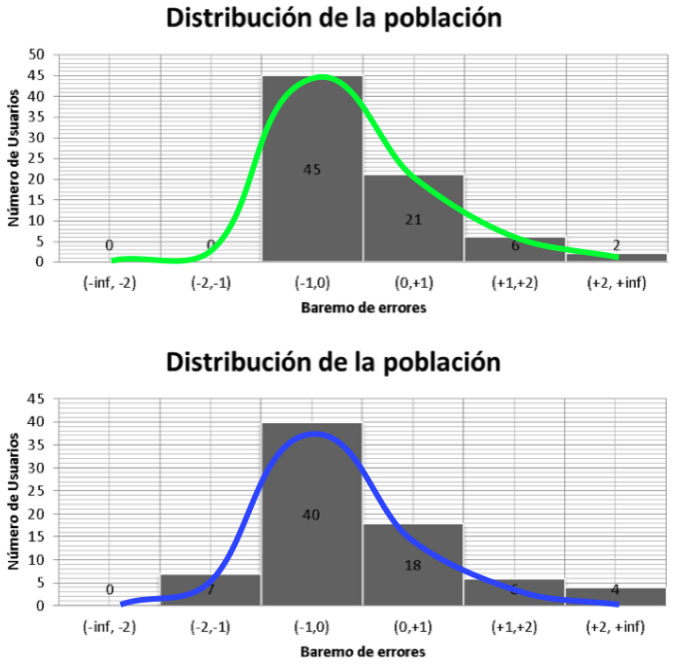


Figura 23. Comparación de los resultados para PMD (Arriba) y COLMENA (Abajo).

[11] https://pmd.github.io/

Los resultados con PMD son muy similares a COLMENA, manteniendo una tendencia hacia el modelo de distribución normal. La diferencia principal entre los resultados de PMD y COLMENA es que PMD genera más errores en determinados estudiantes, haciendo que éstos pertenezcan al grupo medio y avanzado, lo que hace que no exista un grupo experto (Figura 23).

4.2. Análisis de los errores durante un curso académico

El segundo caso de estudio tuvo como propósito conocer exactamente qué errores/warnings concretos y qué familias de los mismos eran las más frecuentes en un entorno de estudiantes de programación, tal y como se indica en la research question 2.

4.2.1 Metodología

Como punto de partida se configuró el entorno de desarrollo Eclipse en los laboratorios en los que se iba a trabajar con COLMENA Plug-in, de tal modo que los datos fueran capturados y almacenados en tiempo real en dicho equipo de forma aislada. Una vez finalizada cada sesión de prácticas, se recopilaron los ficheros en los laboratorios para clasificarlos y cribar todos los proyectos que se hubieran generado por accidente o relativos a otras asignaturas. Para llevar a cabo estas acciones de forma automática se desarrolló un complemento específico a COLMENA Analyzer, dentro del Sistema COLMENA, cuyo cometido fue realizar la persistencia de los errores de los errores y warnings en texto plano a un base de datos relacional, asociándose con el estudiante y la sesión específica de la asignatura (Figura 24).

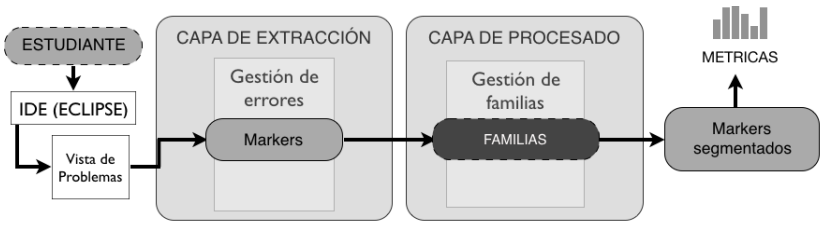


Figura 24. Diagrama de funcionamiento de las Capas de Procesado y Extracción.

4.2.2 Muestra

Para conocer esta información, a lo largo del curso 2012/2013 se analizaron los errores generados por 43 estudiantes en la asignatura de Algoritmia, dentro del segundo semestre en el segundo curso del Grado en Ingeniería Informática del Software (Tabla 8).

Curso	Sesiones consideradas	Tamaño de la muestra
2012/2013	6	43

Tabla 8. Sesión y usuarios considerados para el estudio puntual.

Respecto a la asignatura, ésta estaba dividida en varias sesiones prácticas de temática diferente. Las sesiones, de periodicidad semanal, estaban enfocadas en aprender y desarrollar conocimientos generales de programación de algoritmos, aplicados en el lenguaje Java (Tabla 9).

Sesión	Temática de la sesión	Conceptos de programación
Sesión 1 (S1)	Programación Dinámica	Uso de variables simples y arrays
Sesión 2 (S2)	Programación Dinámica	Uso de variables simples y arrays
Sesión 3 (S3)	Backtracking	Uso de variables y métodos recursivos
Sesión 4 (S4)	Ramifica y Poda	Genericidad
Sesión 5 (S5)	Ramifica y Poda	Genericidad
Sesión 6 (S6)	Divide y vencerás	Uso de objetos y programación en paralelo

Tabla 9. Temas de cada sesión de prácticas y los conceptos de programación con los que están relacionados.

4.2.3 Análisis de los datos

Una vez extraídos los resultados durante las 6 sesiones prácticas de la asignatura, éstos fueron agrupados por familias para cada una de las sesiones, para que sea más entendible para el docente. Si el profesor es capaz de conocer qué tipos de errores predominan en una sesión concreta, en posteriores cursos y sesiones similares tendrá la oportunidad de advertir a sus estudiantes sobre ellos.

4.2.4 Resultados

4.2.4.1 RQ 2. ¿Hay tipos de errores representativos? ¿Cuales son durante un curso académico?

Tras analizar durante 6 sesiones los errores y warning del grupo de estudiantes que conforman la muestra, se estableció una tabla "tabla de top de errores" y se agrupó en un gráfico de barras las familias y la cantidad de errores recolectados para cada una de ellas durante las sesiones.

A la tabla del top de errores se añadió una explicación exhaustiva del error, tratando de acercar más el concepto a los docentes, para en el futuro transmitirlo también a estudiantes que tengan dificultades entendiendo los mensajes de error del IDE o del compilador, que en ocasiones son particularmente crípticos [7, 72, 103]. Estos mensajes simplificados representan una primera versión de la capa de visualización del Modelo COLMENA, cuyo cometido es ofrecer un feedback formativo enriquecido sobre el estado general de un estudiante en materia de programación (Tabla 10).

Mensaje del error/warning	Familia del error/warning		Explicación del error
	Compilador	Concepto	

Value of field/local variable % is not used	Estructura	Warning de tipos	Una variable o campo es declarada pero nunca es utilizada.
Syntax error, insert % to complete statement.	Sintaxis	Error sintáctico	Estructura de código incompleta, falta un punto y coma, paréntesis, llave, etc.
% cannot be resolved to a variable	Variables	Error sintáctico	Una variable usada no ha sido declarada previamente.
% cannot be resolved to a type	Tipos	Error sintáctico	Se está intentando utilizar una variable de un tipo que no existe.
Type mismatch: cannot convert from % to %	Tipos	Error semántico	Posible pérdida de precisión entre variables al no coincidir el tipo.
X is a raw type. References to generic type % <T> should be parameterized	Tipos	Warning de genericidad	La interfaz X es usada pero nunca ha sido parametrizada con el tipo de objeto que implementa dicha interfaz.
The import % is never used	Imports	Warning de imports	Se está importando una librería o Clase que nunca se utiliza en el código.
References to generic type %<T> should be parameterized	Tipos	Warning de genericidad	El método X devuelve un tipo genérico. El tipo de retorno debería ser parametrizado.
The method %(parameters) is undefined for the type Y	Métodos	Error semántico	El método X es invocado pero no ha sido declarado en la Clase Y.

Tabla 10. Ranking de los errores más frecuentes en relación a sus familias, tanto del compilador asociando un concepto y una explicación.

Además de un resumen de los errores más populares a lo largo de las sesiones con sus explicaciones asociadas, también fue foco de estudio la frecuencia de las familias de errores a lo largo del cada semana, con el

objetivo de conocer si alguna familia predomina especialmente según la tipología de la práctica o si existe algún patrón determinante en los errores de los estudiantes (Tabla 11).

Familia	Sesión 1 (S1)	Sesión 2 (S2)	Sesión 3 (S3)	Sesión 4 (S4)	Sesión 5 (S5)	Sesión 6 (S6)
Variables (Variables)	9%	13%	32%	12%	10%	22%
Sintaxis (Syntax)	17%	16%	12%	6%	6%	5%
Tipos (Types)	12%	13%	17%	47%	49%	13%
Imports (Import)	4%	5%	4%	2%	4%	7%
Métodos (Method)	7%	8%	10%	10%	6%	14%
Constructores (Constructor)	1%	1%	0%	1%	2%	3%
Estructurales (Structural)	47%	40%	23%	18%	19%	32%

Tabla 11. Reparto de cada una de las familias de errores a lo largo de las sesiones prácticas de la asignatura.

Si se representan gráficamente los porcentajes y las familias para cada sesión, se construye un gráfico de columnas donde se agrupan por familia en el eje de las abscisas, y donde gráficamente se aprecia el cambio en el porcentaje para cada tipo de error a lo largo de las sesiones, como si de una evolución de ese tipo de familia se tratase (Figura 25).

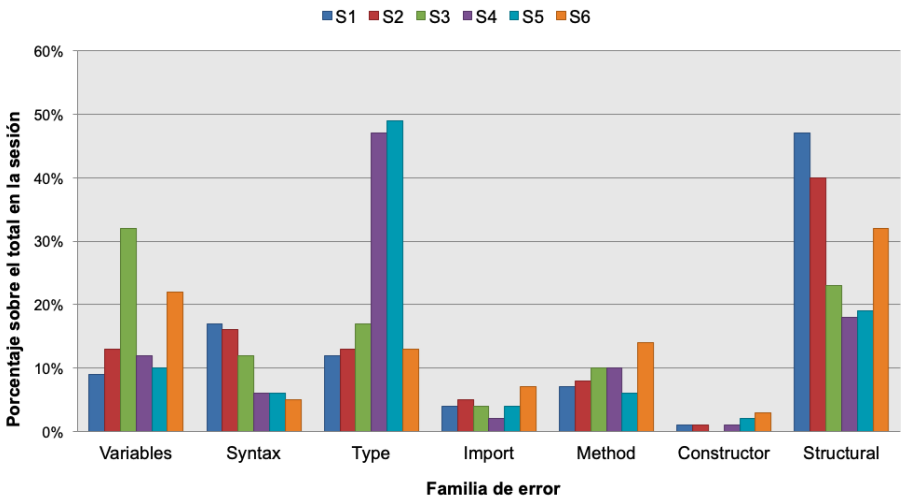


Figura 25. Representación gráfica del porcentaje de familias.

Si se representan los datos colocando en el eje de las abscisas a las sesiones, se obtiene un gráfico en el que se puede apreciar la representatividad de cada familia de error dentro de una misma sesión (Figura 26).

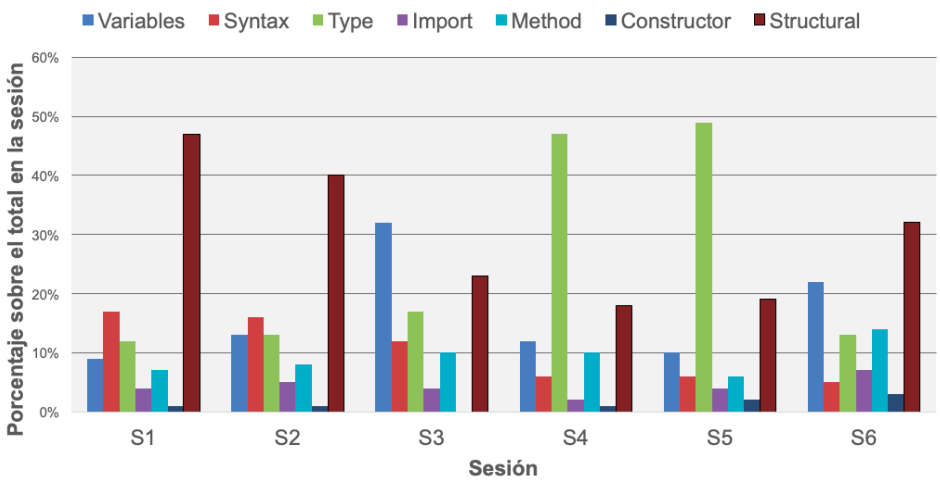


Figura 26. Representación gráfica de familias frente a sesión.

4.2.5 Discusión

4.2.5.1 RQ 2. ¿Hay tipos de errores representativos? ¿Cuales son durante un curso académico?

El error más frecuente se trata de un warning relacionado con la declaración de variables nunca usadas, muy frecuente a la hora de desarrollar código, donde los estudiantes declaran variables a la hora de esquematizar el problema y que en el futuro no serán empleadas para nada. Tras este warning, aparecen errores clásicos relacionados con la sintaxis, relacionados con elementos como los paréntesis, puntos y comas o llaves. Este tipo de errores suelen tener que ver con "descuidos" y no con el conocimiento del lenguaje en sí mismo. Los siguientes errores, relacionados con tipos inexistentes o no inicializados también son muy frecuentes, especialmente en el desarrollo de algoritmos donde se emplean muchas variables auxiliares no consideradas. Además, si los estudiantes han aprendido lenguajes que no requieren la declaración previa de las variables (como es el caso de lenguajes dinámicos) es un error que se puede producir fácilmente en el desarrollo de software en Java.

En relación a los tipos, también aparece un error de falta de precisión, también frecuente cuando se hacen conversiones de enteros a decimal, donde se puede perder precisión. Lo mismo ocurre al convertir el tipo double a float, ambos relacionados con decimales pero con diferente precisión. Tras este error aparecen en la lista warnings relacionados con el casting y la genericidad, librerías importadas pero no utilizadas y tipos de retorno genéricos en lugar de Clases concretas. Estos warnings también son típicos de estudiantes de segundo que utilizan librerías, hacen uso de la genericidad y tratan de atomizar métodos para que sean genéricos. Por último, cierra la lista otro error muy clásico en la programación orientada a objetos, donde se trata de invocar un método en una clase que no lo tiene definido.

En lo que respecta a los porcentajes de cada familia en cada sesión, se puede apreciar que los errores estructurales son los más representativos, con unos porcentajes de aparición entre el 20% y el 50% en cada una de las sesiones. Esta familia, relacionada con el uso del lenguaje en general,

disminuye su presencia en las sesiones 4 y 5, donde los errores de tipos adquieren mucha más importancia, al tratarse de las sesiones relacionadas con Ramifica y Poda, en las que los tipos se utilizan más que en el resto de sesiones y se hace hincapié en la genericidad, concepto en el que los tipos también participan muy activamente.

Centrándonos en la familia de sintaxis, se ve un claro descenso del volumen de errores cometidos con el paso de las semanas, lo que indica que los estudiantes centran sus errores en otro tipo de contextos no tan relacionados en cómo escribir el propio lenguaje y más en su uso. Esto se manifiesta en la tendencia ascendente de las familias de Imports, Métodos y Constructores, que por el contrario experimentan un crecimiento progresivo. Los 3 conceptos de estas familias están más ligados a la evolución de los conceptos teóricos a lo largo de las sesiones, lo cual hace lógica la aparición en mayor medida de estos tipos de errores.

Este caso de estudio se caracterizó por ser el primero en el que existe una continuidad temporal en la captura de errores a lo largo de las sesiones prácticas, y sirve como ejemplo para demostrar que las tipologías de los errores varían en función de la temática de la sesión práctica. Si el docente tiene esta información en un curso, puede aplicarla a una sesión de temática similar con otros grupos de estudiantes o en cursos posteriores, haciendo hincapié en las familias de errores que más hayan destacado.

4.3 Evaluación del Feedback Formativo

A partir de los casos de estudio anteriores, se realizó un tercer caso de estudio que evalúe la efectividad desde las dos perspectivas anteriores, tanto desde el mensaje de error como desde el estudiante (RQ3 y RQ4). En relación a los mensajes de error se desestimó continuar analizando las familias de error, centrando el estudio en el listado de errores concretos debido a la poca homogeneidad que presentan las familias.

4.3.1 Metodología

El objetivo de este caso de estudio es por tanto probar la propuesta de potenciar el feedback formativo del docente con herramientas de soporte como COLMENA, y que así realmente sea efectiva tanto para los estudiantes como para los errores. Para llevar a cabo lo indicado, se realizó una comparativa grupo control-grupo experimental que permite abordar la hipótesis y sus research questions asociadas.

4.3.2 Muestra

El caso de estudio fue realizado sobre una muestra de 155 estudiantes. La muestra fue obtenida de estudiantes principiantes que comenzaban el Grado en Ingeniería Informática del Software durante 2 cursos académicos. La distribución de los estudiantes entre los años son 77 el primer año y 78 el segundo año respectivamente (Tabla 12).

Curso	Sesiones consideradas	Tamaño de la muestra
Primer Curso	4	77
Segundo Curso		78

Tabla 12. Sesión y usuarios considerados para el estudio puntual.

En relación al contexto de la materia de la asignatura monitorizada, el resultado esperado era una adquisición de las habilidades de implementación de algoritmos por parte de los estudiantes, como divide y

vencerás, backtracking, programación dinámica o ramificación y poda. En cuanto al número y la duración de las prácticas, el caso de estudio contempló 4 sesiones de prácticas de 120 minutos de duración, común a todos los cursos. Las cuatro sesiones fueron las siguientes:

1. Divide y Vencerás (Divide and Conquer, o D&C)
2. Algoritmos Voraces (Greedy Algorithms, o Greedy)
3. Programación Dinámica (Dynamic Programming o DP)
4. Backtracking (BT)

Para establecer los grupos de estudio, todos los estudiantes de primer año fueron parte del grupo de control y COLMENA solo fue usada para capturar los errores de compilación generados en cada una de las sesiones prácticas analizadas. El feedback proporcionado por el profesor en este curso fue únicamente un diálogo durante las sesiones de prácticas para analizar su progreso y ayudar a desbloquear los temas que supusieron problemas, todo ello sin emplear los datos obtenidos a través de COLMENA. A la finalización del primer curso, se recopilaron un total de 5.136 errores, relacionados con 40 tipos diferentes de mensajes de error por parte del compilador. Utilizando y procesando este conjunto, se extrajo un subconjunto de 4.389 errores (un 85,46% del total) relacionados con 18 mensajes de error comunes a las 4 prácticas. Estos 18 errores proporcionaron las bases del conocimiento para el profesor, usados como herramienta ideal a la hora de generar feedback a los estudiantes el próximo año.

En cuanto al segundo año académico, el experimento se definió en base a unos estándares similares al año anterior. Las sesiones prácticas fueron enseñadas por el mismo profesor, con los mismos temas y conceptos de la materia y las mismas sesiones que el año anterior. La única diferencia real fue que durante este curso el profesor utilizó la información generada en relación a los 18 tipos de errores detectados para la gran mayoría durante el curso académico anterior.

4.3.3 Procedimiento de Feedback

El proceso de incorporación y suministro de feedback fue la diferencia principal entre el primer y el segundo año. A través de COLMENA, el procedimiento seguido consistió en dos fases principales: antes de comenzar el curso del grupo experimental y la sesión práctica en la que se encontrara el grupo [88] (Tabla 13).

Errores de compilación comunes a lo largo de las cuatro sesiones de prácticas realizadas por el grupo de control (primer curso) y de los que cuyo feedback fue aplicado al grupo experimental (segundo año).		
Código	Mensaje de error	Errores
CE_1	[PLACEHOLDER] cannot be resolved to a variable	1337
CE_2	Syntax error, insert [PLACEHOLDER] to complete ReturnStatement.	406
CE_3	[PLACEHOLDER] cannot be resolved to a type.	403
CE_4	The method contains ([PLACEHOLDER]) in the type [PLACEHOLDER] is not applicable for the arguments ([PLACEHOLDER]).	368
CE_5	Type mismatch: cannot convert from [PLACEHOLDER] to [PLACEHOLDER].	295
CE_6	The declared package [PLACEHOLDER] does not match the expected package	280
CE_7	Cannot make a static reference to the non-static field [PLACEHOLDER].	229
CE_8	This method must return a result of type [PLACEHOLDER].	174
CE_9	Syntax error on token [PLACEHOLDER], delete this token.	159
CE_10	Syntax error on token [PLACEHOLDER], Statement expected after this token.	151

CE_11	The constructor [PLACEHOLDER] is undefined.	148
CE_12	Syntax error on token [PLACEHOLDER], [PLACEHOLDER] expected.	114
CE_13	The local variable [PLACEHOLDER] may not have been initialized.	87
CE_14	Duplicate local variable [PLACEHOLDER].	84
CE_15	Cannot make a static reference to the non-static method [PLACEHOLDER] from the type [PLACEHOLDER].	63
CE_16	Syntax error on token(s), misplaced construct(s).	35
CE_17	Syntax error on token [PLACEHOLDER], invalid Type.	34
CE_18	Return type for the method is missing.	22

Tabla 13. Los campos y códigos de error utilizados a lo largo del caso de estudio para hacer referencia a los mensajes de error específicos.

4.3.3.1 Antes de comenzar el curso grupo experimental

A partir de la información obtenida a partir de la interacción con el IDE por los grupos de control, los 18 errores fueron añadidos a la COLMENA Platform y completados con información adicional que pueda facilitar su comprensión y su origen:

- Error de compilación generado por el compilador de Java
- Descripción comprensiva con lenguaje menos técnico
- Referencias a bibliografía adicional que referencie el error
- Temática o familia del error (tipos, variables, constructores, sintaxis, estructurales, métodos o imports)

Por cada uno de los mensajes de error (18) y por cada sesión (4), el profesor generó una explicación acerca de la causa y origen del error, así

como sugerencias de potenciales soluciones en fragmentos de código en forma de ejemplos (2) para comprender la dimensión del error. A partir de los datos de entrada, por tanto, se generaron 144 ejemplos diferentes de error. La decisión de generar esta base de conocimiento en cada sesión se explica de acuerdo al principio de que el feedback debe ser específico de acuerdo al error para así conseguir que se cumpla además que sea correctivo y positivo. Para lograr esto, es por tanto vital que los errores están estrechamente relacionados con el contexto y temática de la práctica.

4.3.3.2 Al comienzo de cada sesión de prácticas del grupo experimental

En este escenario el profesor presenta a través de COLMENA Platform en un proyecto, como aconteció la sesión, cuáles fueron los principales errores y cómo sucedieron, además de las posibles soluciones para esa sesión particular, tal y como se comentó anteriormente. Es importante remarcar que el profesor conoce cuales son los errores más frecuentes en cada sesión, gracias a la información previamente analizada a partir del grupo de control. Si el estudiante lo pide, el profesor puede ayudar individualmente, de la misma manera que se hizo durante el primer año, pero usando los errores y las explicaciones a través de COLMENA Platform.

Para ejemplificar cómo el feedback es incorporado a una clase, se explica a continuación el caso de Miguel, un estudiante de un grupo experimental. En la sesión práctica, el profesor introduce los objetivos de la sesión así como el código fuente de la práctica que se va a llevar a cabo, junto con la explicación de los errores de la anterior sesión a través de COLMENA. Después de esta explicación, Miguel empieza a programar. Más adelante, la monitorización de los estudiantes a través del plug-in indica que Miguel está generando errores, cómo “Cannot be resolved to a type” (un error relacionado con tipos muy frecuentes). El profesor, utilizando la vista de error en COLMENA, explica algunas situaciones en las cuales este error podría acontecer. Esta información, nueva para Miguel, le ayuda a entender la razón detrás del error, haciendo que se reduzcan las ocurrencias del mismo por esta causa en el futuro.

4.3.4 Análisis de datos

En el presente estudio, COLMENA recuperó 7786 errores de compilación, agrupados en 40 mensajes de error diferentes. En el grupo de control fueron recuperados 5136 errores de compilación frente a los 2650 recopilados en el grupo experimental. Los 18 tipos de errores de compilación representan, como se ha comentado anteriormente, un 85,41% del total de errores generados por los estudiantes, y de forma más concreta 4389 para el grupo de control y 2261 para el grupo experimental. La distribución de los errores en cada sesión puede observarse a continuación (Figura 27) (5).

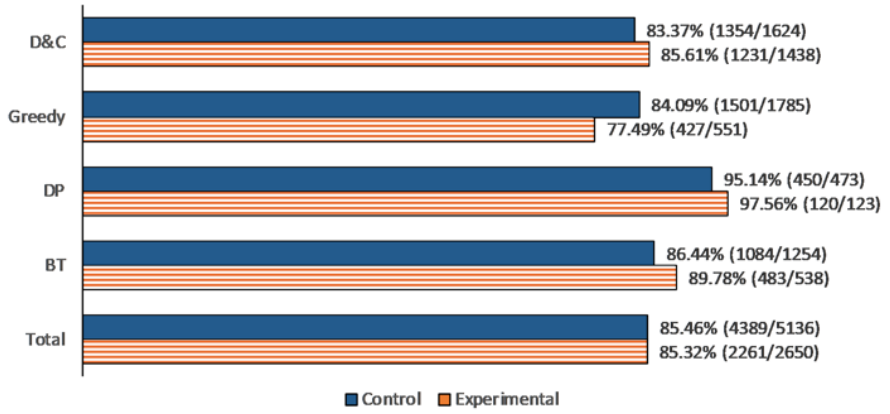


Figura 27. Porcentaje cubierto por los mensajes de los errores de compilación para el grupo de control y el grupo experimental, tanto a para cada sesión como a nivel global. El número de compilaciones totales se indica entre paréntesis.

Para poder responder a las preguntas de investigación, se han empleado métodos estadísticos con el SPSS[12] v26. Para crear los pares de datos se empleó la prueba de rangos Wilcoxon y para los datos desparejados se empleó el test U de Mann-Whitney. Para todos los casos se emplearon test de dos colas con un nivel alpha de significación de 0.05. El tamaño del efecto fue calculado para todas las pruebas, y se expresa como r [86]. La interpretación del tamaño ha sido hecha siguiendo el criterio establecido por Cohen [18, 19], quien indica que:

[12] https://www.ibm.com/es-es/spss

- r = .10 es un tamaño de efecto pequeño
- r = .30 es un tamaño de efecto mediano
- r = .50 es un tamaño de efecto grande

4.3.5 Resultados

La Figura 28 muestra la distribución de cada uno de los 18 mensajes de error estudiados tanto para el grupo de control como para el grupo experimental. En un primer vistazo parece indicar que la distribución de los mensajes de error es muy similar para ambos grupos, encontrando la máxima variación en CE_2, con un 2.14%.

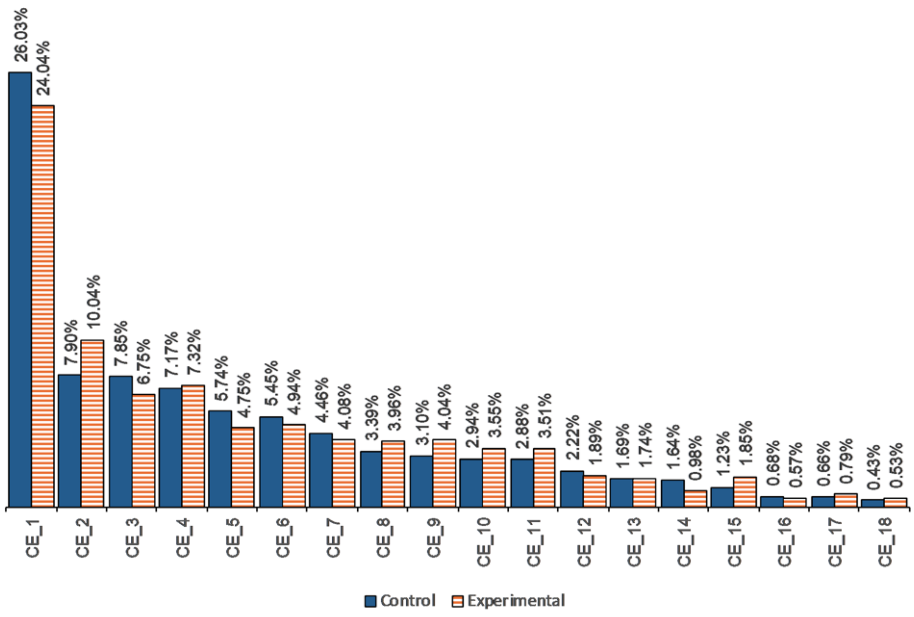


Figura 28. Distribución de los 18 mensajes de error analizador para los grupos experimental y de control al final del caso de estudio.

De acuerdo a los 18 mensajes de error estudiados, la tabla 14 presenta los estadísticos descriptivos de los errores de compilación detectados para cada grupo, tanto a nivel de sesión como en total. De la misma forma, la tabla 15 presenta los estadísticos descriptivos de los

errores de compilación por estudiante para cada grupo y para cada sesión, además de en total. El resto de estadísticos relacionados con cada mensaje de error (desde CE_1 hasta CE_18) tanto para cada sesión como en total, se detallan en el Apéndice.

	Media	Desv. Tip.	Mediana	Min.	Max.
Total					
Control	243.83	300.6	155	22	1337
Experimental	125.61	144.59	99.50	14	637
D&C					
Control	75.22	83.25	46.50	2	363
Experimental	68.39	71.63	60.50	2	319
Greedy					
Control	83.39	98.38	53	7	424
Experimental	23.72	24.61	17.50	2	88
DP					
Control	25	35.55	16.5	4	155
Experimental	6.67	6.80	4	0	23
BT					
Control	60.22	89.80	34	3	395
Experimental	26.83	51.06	15	2	224

Tabla 14. Estadísticos descriptivos de los errores de compilación para cada sesión.

	Media	Desv. Tip.	Mediana	Min.	Max.
Total					
Control	56.27	49.58	45.5	2	267
Experimental	29.36	28.88	20	0	123
D&C					
Control	17.36	18.22	13	0	67
Experimental	15.99	18.66	11	0	111
Greedy					
Control	19.24	25.71	10.5	0	120
Experimental	5.55	7.89	1	0	40
DP					
Control	5.77	8.88	1	0	42
Experimental	1.56	4.46	0	0	21
BT					
Control	13.9	21.33	5.5	0	120
Experimental	6.27	15.59	0	0	99

Tabla 15. Estadísticos descriptivos de los errores de compilación por estudiante.

4.3.5.1 RQ 3. Por mensaje de error, ¿es el feedback efectivo?

La prueba de los rangos de signo de Wilcoxon (dos colas) mostró que el grupo experimental (Mdn = 99.50) tuvo significativamente menos errores de compilación que el grupo de control (Mdn = 155), z = -3.73, p $<$ .001, tamaño de efecto grande (r = .62)

A continuación se estudió cuando el feedback era efectivo en cuanto a que lograba reducir el número total de errores de compilación en una sesión de forma independiente. En este punto, es importante destacar que el feedback fue aplicado a los 18 mensajes de compilación, que son comunes a todas las sesiones.

- En la primera sesión (D&C), no se encontraron diferencias estadísticamente significativas entre los grupos experimental (Mdn = 60.50) y de control (Mdn = 46.50), z = -1.138, p = .25 en lo que se refiere al número de errores de compilación en los que el feedback fue aplicado. Por tanto, en esta primera sesión, los estudiantes de ambos grupos se comportaron de una forma similar.
- En la segunda sesión (Greedy), el test de rangos de Wilcoxon (dos colas) encontró que el grupo experimental (Mdn = 17.50) hizo significativamente menos errores de compilación que el grupo de control (Mdn = 53), z = -3.681, p < .001, con un tamaño de efecto grande (r = .61).
- En la tercera sesión (DP), el test de rangos de Wilcoxon (dos colas) encontró que el grupo experimental (Mdn = 4) hizo significativamente menos errores de compilación que el grupo de control (Mdn = 16.50), z = -3.681, p < .001, con un tamaño de efecto grande (r = .61).
- En la cuarta sesión y final (BT), el test de rangos de Wilcoxon (dos colas) encontró que el grupo experimental (Mdn = 15) hizo significativamente menos errores de compilación que el grupo de control (Mdn = 34), z = 3.680, p < .001, con un tamaño de efecto grande (r = .61).

4.3.5.2 RQ 4. Por estudiante, ¿es el feedback efectivo?

Analizando el número total de errores de compilación en los cuales el feedback fue proporcionado (a través de los 18 mensajes de error), la prueba U de Mann-Whitney encontró que el grupo experimental (Mdn = 20) tuvo significativamente menos errores de compilación que el grupo de control (Mdn = 45.50), U = 1889, z = - 3.99, p < .001, tamaño de efecto medio (r = .32).

Para cada uno de los 18 mensajes de error en los cuales el feedback fue aplicado,se analizó el número total de errores creados por el estudiante. Así, como se muestra en la Tabla 16, la prueba U de Mann-Whitney encontró que el grupo experimental generó significativamente nos errores de compilación por estudiante que el grupo de control para 8 de los 18 mensajes de error: CE_1, CE_2, CE_4, CE_5, CE_6, CE_7, CE_10 y CE_16 respectivamente.

Error code	U	Z	p	r
CE_1	2061.50	-3.38	.002	.27 (medium)
CE_2	2441	-2.03	.043	.16 (small)
CE_4	2168.50	-3.05	.002	.24 (small)
CE_5	2113	-3.30	.001	.026 (small)
CE_6	2169.50	-3.11	.002	.24 (small)
CE_7	2342	-2.66	.008	.21 (small)
CE_10	2408.50	-2.23	.026	.18 (small)
CE_16	2595	-2.02	.043	.16 (small)

Tabla 16. Prueba U de Mann-Whitney y sus resultados significativos obtenidos sobre los 18 mensajes de error iniciales.

Haciendo un análisis más exhaustivo, se analizaron el número de errores de compilación por estudiante en cada una de las sesiones, tanto para el global de errores como para el subconjunto de los 18 errores analizados, aplicando la prueba en U de Mann-Whitney:

- Para la primera sesión (D&C) no hubo diferencias significativas en lo que corresponde al grupo experimental con relación al grupo de control, tanto para el total como para los 18 errores seleccionados.
- En la segunda sesión (Greedy), el grupo experimental generó significativamente menos errores de compilación por estudiante ($Mdn = 1$) que el grupo de control ($Mdn = 10.50$), $U = 1917$, $z = -3.99$, $p < .001$, con un tamaño de efecto medio ($r = .32$). En

análisis individuales, tal y como se muestra en la Tabla 17, el grupo experimental hizo también significativamente menos errores en 12 de los 18 errores: CE_1, CE_4, CE_5, CE_6, CE_7, CE_8, CE_10, CE_11, CE_12, CE_13, CE_14, y CE_16.

- En la tercera sesión (DP), el grupo exprimental generó significativamente menos errores de compilación por estudiante (Mdn = 1) que el grupo de control (Mdn = 1), U = 1878, z = -4.7, p < .001, con un tamaño de efecto medio (r = .38). En análisis individuales, el grupo experimental hizo también significativamente menos errores en 6 de los 18 errores (all Mdn = 0): CE_1 [U = 2250, z = -3.77, p < .001, tamaño de efecto medio (r = .30)], CE_2 [U = 2435, z = -2.91, p = .004, tamaño de efecto pequeño (r = .23)], CE_3 [U = 2660.50, z = -2.55, p = .011) tamaño de efecto pequeño (r = .20), CE_5 (U = 2699, z = -2.19, p = .029, tamaño de efecto pequeño (r = .18)], CE_8 [U = 2616.50, z = -2.62, p = .009, tamaño de efecto pequeño (r = .21)], CE_16 [U = 2810.50, z = -2.25, p = .024, tamaño de efecto pequeño (r = .18)].
- En la cuarta y última sesión (BT), el grupo exprimental generó significativamente menos errores de compilación por estudiante (Mdn = 0) que el grupo de control (Mdn = 5.50), U = 4812.50, z = -4.47, p < .001, con tamaño de efecto medio (r = .36). En análisis individuales, el grupo experimental hizo también significativamente menos errores en 5 de los 18 errores CE_1 [MdnExp = 0, MdnCtrl = 1, U = 2159, z = -3.36, p < .001, tamaño de efecto pequeño (r = .27)], CE_2 [U = 2489.50, z = -2. 46, p = .014, tamaño de efecto pequeño (r = .20)], CE_3 [U = 2553.50, z = -2.36, p = .018, tamaño de efecto pequeño (r = .19)], CE_6 [U = 2481.50, z = -2.87, p = .004, tamaño de efecto pequeño (r = .23)], CE_11 [U = 2661, z = -2.06, p = .039, tamaño de efecto pequeño (r = .17)].

Error code	U	Z	p	r
CE_1	2029	-3.82	< .001	.31 (medium)

CE_4	2064.50	-4.05	< .001	.32 (medium)
CE_5	2021.50	-4.18	< .001	.34 (medium)
CE_6	2342	-3.98	< .001	.32 (medium)
CE_7	2564	-2.38	.017	.19 (small)
CE_8	2459	-2.66	.008	.21 (small)
CE_10	2575	-1.99	.048	.16 (small)
CE_11	2607	-2.18	.029	.18 (small)
CE_12	2503	-2.80	.005	.22 (small)
CE_13	2651.50	-2.32	.020	.19 (small)
CE_14	2581.50	-3.14	.002	.25 (small)
CE_16	2731.50	-2.18	.29	.18 (small)

Tabla 17. Prueba U de Mann-Whitney con resultados significativos para una de las sesiones analizadas (Greedy)

La tabla a continuación (Tabla 18) muestra los tamaños de efecto para cada sesión, tanto en total como para tipo de error. En la tabla, "↑" representa efecto grande, "↓" representa efecto pequeño y por último "↔" corresponde con efecto medio. "." significa que no existen diferencias significativas.

Error code	Greedy	DP (Dynamic Programming)	BT (Backtracking)
Globally	↔	↔	↔
CE_1	↔	↔	↓
CE_2	.	↓	↓
CE_3	.	↓	↓
CE_4	↔	.	.

CE_5	↔	↓	.
CE_6	↔	.	↓
CE_7	↓	.	.
CE_8	↓	↓	.
CE_9	.	.	.
CE_10	↓	.	.
CE_11	↓	.	↓
CE_12	↓	.	.
CE_13	↓	.	.
CE_14	↓	.	.
CE_15	.	.	.
CE_16	↓	↓	.
CE_17	.	.	.
CE_18	.	.	.

Tabla 18. Tamaños de efecto tanto por sesión como por error.

4.3.6 Discusión

El propósito del caso de estudio fue determinar en qué ocasiones el feedback formativo proporcionado por el profesor a través de la plataforma COLMENA ayudaba a reducir el número de errores de compilación generado por los estudiantes en los contextos de aprendizaje cara a cara. Esta discusión se enfoca en base a las preguntas de investigación planteadas para este estudio.

4.3.6.1 RQ 3. Por mensaje de error, ¿es el feedback efectivo?

La respuesta más rápida es simplemente decir que el feedback formativo proporcionado por el profesor con la ayuda de COLMENA es efectivo. En el estudio propuesto se analizó esta cuestión desde dos puntos de vista:

1. Considerando la suma total de errores producidos, hay una reducción significativa de los errores, casi del 50%.
2. Considerando la suma total de errores por sesión, los resultados indican que en 3 de las 4 sesiones hay una reducción manifiesta.

Estos resultados demuestran que el enfoque propuesto por COLMENA (el cual está focalizado en conceptos específicos de error) es más adecuado que otras propuestas como la de Jadud [48] o Luke [62]. Por tanto, la colección de errores relacionada con una sesión de prácticas específica durante la cual el profesor aporta contenidos específicos en un momento concreto aporta un feedback formativo más eficiente e inmediato que los que se pueden hacer de forma natural. La características que tiene COLMENA de ser tan inmediato ha permitido que el feedback sea correctivo y por tanto, ayuda a que sea también positivo. Así, la reducción de hasta un 50% de la cantidad de errores producidos en el grupo experimental, confirma la necesidad de tener herramientas como COLMENA que acojan este feedback formativo que dé el docente. Comparando con otras herramientas o enfoques basados en enriquecer mensajes de error [24], detección y solución automática de errores [1], reparación de programas [102], se encuentra que éstos no han sido tan prácticos a la hora de generar feedback automáticamente porque la natural del feedback es mínimo [25].

Un análisis más detallado, en este caso a nivel de sesión, nos permite bucear más en esta cuestión. De este modo, en la primera sesión (D&C), donde los estudiantes implementan el algoritmo de Divide y Vencerás, no se encuentran diferencias significativas. En esta sesión, el profesor hace el primer contacto con los estudiantes, y para los estudiantes de ambos grupos (tanto el de control como el experimental) se trata de su primera

sesión. Los datos nos indican la gran cantidad de errores que existen en esa primera sesión, donde los estudiantes se están empezando a familiarizar con la asignatura, profesor y entorno de trabajo. Además, los estudiantes necesitan algo de conocimiento para ser capaces de implementar el algoritmo de Divide y Vencerás, con estrategias como la recursividad, uno de los conceptos más complejos de enseñar [20, 36, 41, 95] y de entender. Todo ello es lo que potencia la gran cantidad de errores en ambos grupos. Es importante destacar que en el grupo experimental hay igualmente menos errores de compilación que en el grupo de control, lo que es indicativo de cómo COLMENA nuevamente es una ayuda valiosa para el profesor, aunque es cierto que en este caso, la primera sesión de prácticas, esa ayuda no es significativa.

La falta de diferencias significativas en esta primera sesión y la gran cantidad de errores en programación va en relación a trabajos previos en los que se concluyó que los estudiantes no tenían el background y los conocimientos necesarios [23]. La falta de este conocimiento hace difícil para los estudiantes crear conexiones críticas entre el feedback y el trabajo realizado [87]. Esto es precisamente lo que puede suceder al comparar las sesiones de D&C y BT, explicando por qué en la primera sesión no hay una reducción significativa de errores de compilación y sin embargo en la última si. En BT, los estudiantes ya han asimilado el conocimiento explicado previamente en D&C y por tanto, son más capaces de interpretar y aplicar el feedback.

4.3.6.2 RQ 4. Por estudiante, ¿es el feedback efectivo?

En esta pregunta de investigación, se analiza donde el feedback ayudó a reducir los errores de compilación por estudiante, tanto a nivel global como a nivel de sesión. En líneas generales, en 8 de los 18 mensajes analizados (60,37% del total de errores cometidos), se encontró una reducción significativa del número de errores creados por cada estudiante. De esta forma, el feedback proporcionado al grupo experimental es más efectivo que en el grupo de control. Esto ocurre especialmente en los errores más frecuentes (CE_1 a CE_7). Existe sin embargo una excepción en el mensaje CE_3, el cual representa a un total de 6,75% de los errores totales generados por el grupo experimental. Este mensaje de error ocurre

normalmente cuando el estudiante trata de utilizar una clase que no se ha importado o que no está accesible desde un fragmento de código desde la cual es reverenciada. Como se observa en los ejemplos proporcionados por COLMENA, es común que los estudiantes cometan errores importando librerías externas a través del IDE. Este aspecto se considera importante porque COLMENA no proporciona ninguna información en cómo trabajar con IDE, por lo que el error CE_3 es un buen ejemplo para indicar la necesidad de añadir explicaciones complementarias en las herramientas acerca de cómo utilizar el IDE. Estas explicaciones adicionales formarían un feedback más elaborado [100]que podría ser incluido en las herrerianas que dan feedback. Sin embargo, en el momento presente, este feedback más elaborado no se considera como feedback generado automáticamente, ya que como se comenta en el estado del arte, ninguna de las formas mencionadas de generar feedback formativo ha solucionado el problema del feedback orientado a tareas, ni tampoco por tanto correctivo o positivo.

Considerando las sesiones de forma individual, el feedback ayudó a reducir el número de errores por estudiante en las mismas sesiones que en la pregunta de investigación anterior, ocurriendo de nuevo el fenómeno con la primera sesión (D&C) donde el feedback no ayudó a reducir el número de errores por estudiante. En las sesiones donde el feedback fue efectivo, la reducción de errores por estudiante ocurrió de forma desigual en función del mensaje de error. La primera vez que ocurre una reducción para algunos de los errores más frecuentes (CE_1, CE_4, CE_5 y CE_6), se observa que mucho mayor en las primeras sesiones que en las últimas, moviéndonos de un tamaño medio de efecto a un tamaño de efecto pequeño. Esto es un indicativo de que el feedback del grupo experimental es especialmente efectivo en las primeras sesiones, reduciendo su efectividad a lo largo de las sesiones posteriores. El hecho de que estas diferencias se vayan haciendo cada vez más pequeñas podría ser debido al efecto positivo y correctivo del feedback. Por ejemplo, que tras ver un error y arreglarlo debido al feedback recibido, los estudiantes estuvieran en condición de generar menos ese error en el futuro [23].

En la línea de otros autores [23] se observa que hay existencia de errores asociados con falta de conocimiento. Por ejemplo, en 4 errores

(CE_9, CE_15, CE_17 y CE_18), no hay diferencia entre el grupo de control y el grupo experimental. Esto es debido a que no hay cambio de comportamiento entre los estudiantes en estos 4 casos de error en los dos grupos porque se necesita un background de conocimiento previo, el cual al no existir, lleva a que acontezcan dichos errores. Por ejemplo, el error CE_18 (Return type for the method is missing) en el panel de errores, es un error típico de estudiantes noveles que probablemente no entiendan por completo la funcionalidad de un método y los resultados que puede retornar. Otro ejemplo es el CE_9 (Syntax error on token [PLACEHOLDER], delete this token). Esta referencia se debe a múltiples razones, desde la falta de un operador de comparación a la hora de asignar las variables ("=") hasta el uso inadecuado de operaciones lógicas. De hecho, el propio uso de expresiones lógicas en las técnicas de Algoritmos Voraces (Greedy), Programación Dinámica (DP) y Backtracking (BT) permite a los algoritmos funcionar. De una manera concreta, en BT hay que explorar 3 potenciales soluciones a un problema, mientras que en Greedy y DP estos valores son seleccionados, de acuerdo al algoritmo, lo que permite que funcione correctamente. Si los estudiantes no entienden las técnicas, es muy complicado para ellos hacer el uso apropiado de estas expresiones lógicas. Una vez más todo esto confirma la necesidad de un feedback correctivo dado por el profesor, gracias a COLMENA. Incluso en el caso de que el feedback esté relacionado con una sesión específica, tiene un efecto positivo y además reduce la aparición de los errores en el futuro.

Todos los resultados mencionados anteriormente no son obtenidos con mensajes de error enriquecidos [8], los que en algunas ocasiones son problemáticos, especialmente para estudiantes noveles [24]. Tampoco son obtenidos con herramientas de detectar errores y reparar programas que han demostrado no ser especialmente útiles en su propósito [102]. Es incluso imposible conseguirlos implementando alguna técnica que permita la generación de feedback personalizado automatizado para el estudiante, ya sea mostrando pistas, sugerencias [78] o ejemplos [23], y que utilice muestras a gran escala ya que no es práctico hacerlo a gran escala porque, al final, se requerirá un examen manual de la solución del error [1].

5. Conclusiones

El feedback ayuda a reducir los errores, pero para que sea efectivo ha de ser específico para el error, inmediato, positivo y correctivo. Actualmente, las investigaciones están orientadas a generar un feedback automático que está muy lejos de ser efectivo porque no consigue las características mencionadas. En esta situación, sigue siendo el profesor un agente fundamental en el proceso de enseñanza-aprendizaje de la programación. Sin embargo, a pesar de su protagonismo, dispone de recursos limitados para ofrecer el feedback adecuado en escenarios educativos con muchos estudiantes y poco tiempo.

Con algunos de los casos de estudio de este trabajo se ha demostrado que soportando el feedback formativo del profesor con las herramientas adecuadas, dicho feedback puede ser efectivo. A través de un caso de estudio con estudiantes novatos se ha llegado a la conclusión que el feedback de las características mencionadas anteriormente reduce los errores de compilación y favorece que los estudiantes cometan menos errores. Por tanto, es necesario ayudar al profesor con herramientas como la que se ha desarrollado en este trabajo y que se denomina COLMENA, porque la generación del feedback automático no está lo suficientemente avanzado para ofrecer ese feedback al estudiantes sin la intervención del profesor.

Se ha encontrado a través de un estudio comparativo incluyendo dos grupos que el grupo de control (donde el profesor da feedback formativo a los estudiantes sin ningún apoyo) tiene más errores asociados con los 18 mensajes de error con relación al grupo experimental (en el que el profesor proporcionó feedback formativo con la ayuda de COLMENA). El resultado anterior está complementado con otros resultados más exhaustivos que demuestran la efectividad del feedback formativo del profesor gracias a la tecnología COLMENA. Así, para cada mensaje de error, el feedback es efectivo en diferentes aspectos:

1. El número de compilaciones con errores es significativamente menor en el grupo experimental.

2. En un estudio más detallado durante 4 sesiones, se encontró que en 3 de ellas había significativamente menos errores de compilaciones en los grupos de estudiantes que recibieron el feedback formativo del profesor a través de COLMENA.

3. Por estudiante en el grupo experimental generaron menos errores que en el grupo de control.

4. Para los 18 mensajes de error analizados en los cuales el feedback fue aplicado, en 8 de ellos el número de errores es significativamente menor gracias al soporte de COLMENA.

En un análisis detallado de los 18 mensajes nos permite concluir que en 3 de las 4 sesiones, el feedback con COLMENA es efectivo, con la reducción de errores respectiva. Desde un punto de vista holístico del curso, los errores se han reducido para cada tipo de error.

6. Trabajo Futuro

A pesar de todos los esfuerzos que se hacen para proporcionar feedback a los estudiantes, todavía hay un margen que requiere un estudio profundo.

En esta investigación se ha evidenciado que el feedback formativo ayuda a corregir los errores de compilación, sin embargo, también:

- La necesidad de un feedback más elaborado en dos líneas paralelas y complementarias, esto es, el feedback asociado al uso de los IDEs y el asociado a los errores de compilación relacionados con la falta de conocimientos específicos de cada sesión.
- Realizar investigación con mayores muestras en diferentes contextos y culturas y con cursos asíncronos, en formatos de cursos online o de mayor duración.
- Extender los análisis para la búsqueda de paralelismos con otros lenguajes de programación.

7. Publicaciones

El trabajo realizado en este tesis doctoral ha sido publicado en las siguientes revistas y congresos:

- Sanchez-Santillan, M. et al. An empirical evaluation of the formative feedback supported by dashboard in the context of compilation error. Computer Applications in Engineering Education. n/a, n/a. DOI:https://doi.org/10.1002/cae.22640.
- Fernandez-Medina, C. et al. 2013. Assistance in computer programming learning using educational data mining and learning analytics. *Proceedings of the 18th ACM conference on Innovation and technology in computer science education* (2013), 237–242.
- Fernández-Medina, C. et al. 2011. COLMENA : Collaborative knowledge and user classification environment based on programming experience. (Ciudad Real, Spain, 2011), 50–58.

8. Referencias

[1] Ahmed, T. et al. 2022. SynShine: Improved Fixing of Syntax Errors. *IEEE Transactions on Software Engineering*. (2022), 1–13. DOI:https://doi.org/10.1109/TSE.2022.3212635.

[2] Ala-Mutka, K.M. 2005. A Survey of Automated Assessment Approaches for Programming Assignments. *Computer Science Education*. 15, 2 (Jun. 2005), 83–102. DOI:https://doi.org/10.1080/08993400500150747.

[3] Altadmri, A. and Brown, N.C.C. 2015. 37 Million Compilations: Investigating Novice Programming Mistakes in Large-Scale Student Data. (2015), 522–527.

[4] Barnes, D. and Kolling, M. 2016. *Objects First with Java: A Practical Introduction Using BlueJ*.

[5] Becker, B. 2015. *An Exploration of the Effects of Enhanced Compiler Error Messages for Computer Programming Novices*.

[6] Becker, B.A. 2016. A New Metric to Quantify Repeated Compiler Errors for Novice Programmers. *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education* (New York, NY, USA, 2016), 296–301.

[7] Becker, B.A. 2016. An Effective Approach to Enhancing Compiler Error Messages. (2016), 126–131.

[8] Becker, B.A. et al. 2019. Compiler Error Messages Considered Unhelpful: The Landscape of Text-Based Programming Error Message Research. *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education* (New York, NY, USA, Dec. 2019), 177–210.

[9] Becker, B.A. et al. 2016. Effective compiler error message enhancement for novice programming students. *Computer Science Education*. 26, 2–3 (Jul. 2016), 148–175. DOI:https://doi.org/10.1080/08993408.2016.1225464.

[10] Ben-Ari, M. 1998. Constructivism in computer science education. *ACM SIGCSE Bulletin*. 30, 1 (Mar. 1998), 257–261. DOI:https://doi.org/10.1145/274790.274308.

[11] Bennedsen, J. and Caspersen, M.E. 2007. Failure rates in introductory programming. *ACM SIGCSE Bulletin*. 39, 2 (Jun. 2007), 32–36. DOI:https://doi.org/10.1145/1272848.1272879.

[12] BLOOM, B.S. 1984. The 2 Sigma Problem: The Search for Methods of Group Instruction as Effective as One-to-One Tutoring. *Educational Researcher*. 13, 6 (Jun. 1984), 4–16.

DOI:https://doi.org/10.3102/0013189X013006004.
[13] Brown, N.C.C. et al. 2014. Blackbox: a large scale repository of novice programmers' activity. (2014), 223–228.
[14] Brown, N.C.C. et al. 2018. Blackbox, Five Years On: An Evaluation of a Large-scale Programming Data Collection Project. *Proceedings of the 2018 ACM Conference on International Computing Education Research* (Espoo, Finland, Aug. 2018), 196–204.
[15] Bubica, N. and Boljat, I. 2014. Teaching of Novice Programmers: Strategies, Programming Languages and Predictors. (Jun. 2014).
[16] Clayberg, E. and Rubel, D. 2008. *Eclipse Plug-ins*. Addison-Wesley Professional.
[17] Clear, T. et al. 2008. Reliably classifying novice programmer exam response using the SOLO taxonomy. *21st Annual conference of the National Advisory Committee on Computing Qualifications (NACCQ 2008)*. (Jul. 2008).
[18] Cohen, J. 1992. A power primer. *Psychological Bulletin*. 112, 1 (1992), 155–159. DOI:https://doi.org/10.1037/0033-2909.112.1.155.
[19] Cohen, J. 1988. *Statistical power analysis for the behavioral sciences*. L. Erlbaum Associates.
[20] Dale, N.B. 2006. Most difficult topics in CS1: results of an online survey of educators. *ACM SIGCSE Bulletin*. 38, 2 (Jun. 2006), 49–53. DOI:https://doi.org/10.1145/1138403.1138432.
[21] Denny, P. et al. 2012. All syntax errors are not equal. *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education* (2012), 75–80.
[22] Denny, P. et al. 2011. CodeWrite: supporting student-driven practice of java. *Proceedings of the 42nd ACM technical symposium on Computer science education* (New York, NY, USA, Mar. 2011), 471–476.
[23] Denny, P. et al. 2014. Enhancing syntax error messages appears ineffectual. *Proceedings of the 2014 conference on Innovation & technology in computer science education* (New York, NY, USA, Jun. 2014), 273–278.
[24] Denny, P. et al. 2021. On Designing Programming Error Messages for Novices: Readability and its Constituent Factors. *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (New York, NY, USA, May 2021), 1–15.
[25] Denny, P. et al. 2021. Promoting Early Engagement with Programming Assignments Using Scheduled Automated Feedback. *Proceedings of the 23rd Australasian Computing Education*

Conference (New York, NY, USA, Mar. 2021), 88–95.
[26] Douce, C. et al. 2005. Automatic test-based assessment of programming: A review. *Journal on Educational Resources in Computing*. 5, 3 (Sep. 2005), 4-es. DOI:https://doi.org/10.1145/1163405.1163409.
[27] Duval, E. 2011. Attention please! Learning analytics for visualization and recommendation. *1st International Conference on Learning Analytics (LAK11* (2011).
[28] Edwards, S. et al. 2017. DevEventTracker: Tracking Development Events to Assess Incremental Development and Procrastination. (Jun. 2017), 104–109.
[29] Edwards, S.H. and Perez-Quinones, M.A. 2008. Web-CAT: automatically grading programming assignments. *Proceedings of the 13th annual conference on Innovation and technology in computer science education* (New York, NY, USA, Jun. 2008), 328.
[30] Ettles, A. et al. 2018. Common Logic Errors Made by Novice Programmers. *Proceedings of the 20th Australasian Computing Education Conference* (New York, NY, USA, 2018), 83–89.
[31] Ferguson, P. 2011. Student perceptions of quality feedback in teacher education. *Assessment & Evaluation in Higher Education*. 36, 1 (Jan. 2011), 51–62. DOI:https://doi.org/10.1080/02602930903197883.
[32] Fernandez-Medina, C. et al. 2013. Assistance in computer programming learning using educational data mining and learning analytics. *Proceedings of the 18th ACM conference on Innovation and technology in computer science education* (2013), 237–242.
[33] Fernández-Medina, C. et al. 2011. COLMENA : Collaborative knowledge and user classification environment based on programming experience. (Ciudad Real, Spain, 2011), 50–58.
[34] Glahn, C. et al. 2008. Smart indicators to support the learning interaction cycle. *International Journal of Continuing Engineering Education and Life-Long Learning*. 18, 1 (2008), 98–117.
[35] Gobet, F. and Clarkson, G. 2004. Chunks in expert memory: evidence for the magical number four ... or is it two? *Memory (Hove, England)*. 12, 6 (Nov. 2004), 732–747. DOI:https://doi.org/10.1080/09658210344000530.
[36] GoldmanKen et al. 2010. Setting the Scope of Concept Inventories for Introductory Computing Subjects. *ACM Transactions on Computing Education (TOCE)*. (Jun. 2010). DOI:https://doi.org/10.1145/1789934.1789935.
[37] Gomes, A. and Mendes, A. 2007. *Learning to program -*

difficulties and solutions.
[38] Gómez-Aguilar, D.A. et al. 2015. Tap into visual analysis of customization of grouping of activities in eLearning. *Computers in Human Behavior*. 47, (Jun. 2015), 60–67. DOI:https://doi.org/10.1016/j.chb.2014.11.001.
[39] Gómez-Albarrán, M. 2005. The Teaching and Learning of Programming: A Survey of Supporting Software Tools. *The Computer Journal*. 48, 2 (Jan. 2005), 130–144. DOI:https://doi.org/10.1093/comjnl/bxh080.
[40] Hartmann, B. et al. 2010. What would other programmers do: suggesting solutions to error messages. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (2010), 1019–1028.
[41] Hertz, M. and Ford, S.M. 2013. Investigating factors of student learning in introductory courses. *Proceeding of the 44th ACM technical symposium on Computer science education* (New York, NY, USA, Mar. 2013), 195–200.
[42] Holland, S. et al. 1997. Avoiding object misconceptions. *ACM SIGCSE Bulletin*. 29, 1 (Mar. 1997), 131–134. DOI:https://doi.org/10.1145/268085.268132.
[43] Houten, R.V. 1980. *Learning Through Feedback: A Systematic Approach for Improving Academic Performance*.
[44] Hristova, M. et al. 2003. Identifying and correcting Java programming errors for introductory computer science students. *ACM SIGCSE Bulletin* (2003), 153–156.
[45] Identifying At-Risk Novice Java Programmers Through the Analysis of Online Protocols: 2009. *https://www.semanticscholar.org/paper/Identifying-At-Risk-Novice-Java-Programmers-Through-Tabanao/4050e2c4ca906231291a2055cea8f907b0c89828*. Accessed: 2021-07-04.
[46] Jackson, J. et al. 2005. Identifying top Java errors for novice programmers. *Frontiers in Education, 2005. FIE'05. Proceedings 35th Annual Conference* (2005), T4C-T4C.
[47] Jadud, M.C. 2005. A First Look at Novice Compilation Behaviour Using BlueJ. *Computer Science Education*. 15, 1 (Mar. 2005), 25–40. DOI:https://doi.org/10.1080/08993400500056530.
[48] Jadud, M.C. 2006. *An Exploration of Novice Compilation Behaviour in BlueJ*. University of Kent.
[49] Jadud, M.C. 2006. Methods and tools for exploring novice compilation behaviour. *Proceedings of the second international workshop on Computing education research* (2006), 73–84.

[50] Karvelas, I. and Becker, B.A. 2022. Sympathy for the (Novice) Developer: Programming Activity When Compilation Mechanism Varies. *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education - Volume 1* (New York, NY, USA, Feb. 2022), 962–968.

[51] Kazerouni, A.M. 2020. Measuring the Software Development Process to Enable Formative Feedback. (Apr. 2020).

[52] Keuning, H. et al. 2016. Towards a Systematic Review of Automated Feedback Generation for Programming Exercises. *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education* (Arequipa, Peru, Jul. 2016), 41–46.

[53] Knobbout, J. and van der Stappen, E. 2020. Where is the Learning in Learning Analytics? A Systematic Literature Review on the Operationalization of Learning-Related Constructs in the Evaluation of Learning Analytics Interventions. *IEEE Transactions on Learning Technologies*. (2020), 1–1. DOI:https://doi.org/10.1109/TLT.2020.2999970.

[54] Lahtinen, E. et al. 2005. A study of the difficulties of novice programmers. (Sep. 2005), 14–18.

[55] Lazarinis, F. et al. 2019. A blended learning course for playfully teaching programming concepts to school teachers. *Education and Information Technologies*. 24, 2 (Mar. 2019), 1237–1249. DOI:https://doi.org/10.1007/s10639-018-9823-2.

[56] Le, N.-T. et al. 2013. A Review of AI-Supported Tutoring Approaches for Learning Programming. *Advanced Computational Methods for Knowledge Engineering* (Heidelberg, 2013), 267–279.

[57] Leinonen, J. et al. 2022. A Comparison of Immediate and Scheduled Feedback in Introductory Programming Projects. *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education - Volume 1* (New York, NY, USA, Feb. 2022), 885–891.

[58] Li, F.W.B. et al. 2010. An Adaptive Course Generation Framework. *International Journal of Distance Education Technologies*. 8, 3 (Jul. 2010), 47–64. DOI:https://doi.org/10.4018/jdet.2010070104.

[59] Li, Q. et al. 2008. Technology supports for distributed and collaborative learning over the internet. *ACM Trans. Internet Techn.* 8, (Feb. 2008). DOI:https://doi.org/10.1145/1323651.1323656.

[60] Lister, R. et al. 2009. Naturally occurring data as research instrument: Analyzing examination responses to study the novice

programmer. *ACM SIGCSE Bulletin*. 41, (Dec. 2009), 156–173. DOI:https://doi.org/10.1145/1709424.1709460.
[61] Lopez, M. et al. 2008. Relationships between reading, tracing and writing skills in introductory programming. (Jan. 2008), 101–112.
[62] Luke, J.A. 2015. Continuously Collecting Software Development Event Data As Students Program. *undefined*. (2015).
[63] Luxton-Reilly, A. et al. 2018. Introductory Programming: A Systematic Literature Review. (Dec. 2018). DOI:https://doi.org/10.1145/3293881.3295779.
[64] Madison, S.K. 1995. *A Study of College Students' Construct of Parameter Passing Implications for Instruction*.
[65] Mangaroska, K. and Giannakos, M. 2019. Learning Analytics for Learning Design: A Systematic Literature Review of Analytics-Driven Design to Enhance Learning. *IEEE Transactions on Learning Technologies*. 12, 4 (Oct. 2019), 516–534. DOI:https://doi.org/10.1109/TLT.2018.2868673.
[66] Marceau, G. et al. 2011. Mind your language: on novices' interactions with error messages. *Proceedings of the 10th SIGPLAN symposium on New ideas, new paradigms, and reflections on programming and software* (New York, NY, USA, Oct. 2011), 3–18.
[67] Martinez-Maldonado, R. et al. 2015. MTFeedback: Providing Notifications to Enhance Teacher Awareness of Small Group Work in the Classroom. *IEEE Transactions on Learning Technologies*. 8, 2 (Apr. 2015), 187–200. DOI:https://doi.org/10.1109/TLT.2014.2365027.
[68] Matcha, W. et al. 2020. A Systematic Review of Empirical Studies on Learning Analytics Dashboards: A Self-Regulated Learning Perspective. *IEEE Transactions on Learning Technologies*. 13, 2 (Apr. 2020), 226–245. DOI:https://doi.org/10.1109/TLT.2019.2916802.
[69] Matsuzawa, Y. et al. 2013. Programming process visualizer: a proposal of the tool for students to observe their programming process. *Proceedings of the 18th ACM conference on Innovation and technology in computer science education* (2013), 46–51.
[70] McCall, D. 2016. *Novice Programmer Errors - Analysis and Diagnostics*. University of Kent,.
[71] McCall, D. and Kölling, M. 2014. Meaningful categorisation of novice programmer errors. *2014 IEEE Frontiers in Education Conference (FIE) Proceedings* (Oct. 2014), 1–8.
[72] McCauley, R. et al. 2008. Debugging: a review of the literature from an educational perspective. *Computer Science Education*. 18, 2

(Jun. 2008), 67–92.
DOI:https://doi.org/10.1080/08993400802114581.

[73] McCracken, M. et al. 2001. A Multi-national, Multi-institutional Study of Assessment of Programming Skills of First-year CS Students. *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education* (New York, NY, USA, 2001), 125–180.

[74] Mordechai (Moti) Ben-Ari 2011. *Compile and Runtime Errors in Java*. Rice University.

[75] Parihar, S. et al. 2017. Automatic Grading and Feedback using Program Repair for Introductory Programming Courses. *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education* (New York, NY, USA, Jun. 2017), 92–97.

[76] Pensieve | Proceedings of the 50th ACM Technical Symposium on Computer Science Education: 2019. *https://dl.acm.org/doi/abs/10.1145/3287324.3287483*. Accessed: 2020-04-05.

[77] Pettit, R.S. et al. 2017. Do Enhanced Compiler Error Messages Help Students? Results Inconclusive. *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (New York, NY, USA, Mar. 2017), 465–470.

[78] Phothilimthana, P.M. and Sridhara, S. 2017. High-Coverage Hint Generation for Massive Courses: Do Automated Hints Help CS1 Students? *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education* (New York, NY, USA, Jun. 2017), 182–187.

[79] Prather, J. et al. 2017. On Novices' Interaction with Compiler Error Messages: A Human Factors Approach. *Proceedings of the 2017 ACM Conference on International Computing Education Research* (New York, NY, USA, Aug. 2017), 74–82.

[80] Price, T.W. et al. 2016. Evaluation of a Frame-based Programming Editor. *Proceedings of the 2016 ACM Conference on International Computing Education Research* (New York, NY, USA, 2016), 33–42.

[81] Program Wars | Proceedings of the 50th ACM Technical Symposium on Computer Science Education: 2019. *https://dl.acm.org/doi/abs/10.1145/3287324.3287496*. Accessed: 2020-04-05.

[82] Putnam, R.T. et al. 1986. A Summary of Misconceptions of High School Basic Programmers. *Journal of Educational Computing*

Research. 2, 4 (Nov. 1986), 459–472. DOI:https://doi.org/10.2190/FGN9-DJ2F-86V8-3FAU.

[83] Python Is Now the Most Popular Introductory Teaching Language at Top U.S. Universities: *https://cacm.acm.org/blogs/blog-cacm/176450-python-is-now-the-most-popular-introductory-teaching-language-at-top-us-universities/fulltext*. Accessed: 2021-07-07.

[84] Ragonis, N. and Ben-Ari, M. 2005. A long-term investigation of the comprehension of OOP concepts by novices. *Computer Science Education*. 15, 3 (Sep. 2005), 203–221. DOI:https://doi.org/10.1080/08993400500224310.

[85] Rodrigo, Ma.M.T. et al. 2009. Affective and Behavioral Predictors of Novice Programmer Achievement. *Proceedings of the 14th Annual ACM SIGCSE Conference on Innovation and Technology in Computer Science Education* (New York, NY, USA, 2009), 156–160.

[86] Rosenthal, R. 1994. Parametric measures of effect size. *undefined*. (1994).

[87] Sadler, D.R. 2010. Beyond feedback: developing student capability in complex appraisal. *Assessment & Evaluation in Higher Education*. 35, 5 (Aug. 2010), 535–550. DOI:https://doi.org/10.1080/02602930903541015.

[88] Sanchez-Santillan, M. et al. An empirical evaluation of the formative feedback supported by dashboard in the context of compilation error. *Computer Applications in Engineering Education*. n/a, n/a. DOI:https://doi.org/10.1002/cae.22640.

[89] Scheeler, M.C. et al. 2016. Providing Performance Feedback to Teachers: A Review: *Teacher Education and Special Education*. (Aug. 2016). DOI:https://doi.org/10.1177/088840640402700407.

[90] Shute, V.J. 2007. Focus on Formative Feedback. *ETS Research Report Series*. 2007, 1 (2007), i–47. DOI:https://doi.org/10.1002/j.2333-8504.2007.tb02053.x.

[91] Sinclair, H. and Cleland, J. 2007. Undergraduate medical students: Who seeks formative feedback? *Medical education*. 41, (Jul. 2007), 580–2. DOI:https://doi.org/10.1111/j.1365-2923.2007.02768.x.

[92] Soloway, E. 1986. Learning to program= learning to construct mechanisms and explanations. *Communications of the ACM*. 29, 9 (1986), 850–858.

[93] Spacco, J. et al. 2013. Towards improving programming habits to create better computer science course outcomes. *Proceedings of the 18th ACM conference on Innovation and technology in computer*

science education (2013), 243–248.
[94] Tabanao, E.S. et al. 2011. Predicting at-risk novice Java programmers through the analysis of online protocols. *Proceedings of the seventh international workshop on Computing education research* (2011), 85–92.
[95] Tew, A.E. and Guzdial, M. 2011. The FCS1: a language independent assessment of CS1 knowledge. *Proceedings of the 42nd ACM technical symposium on Computer science education* (New York, NY, USA, Mar. 2011), 111–116.
[96] Thiselton, E. and Treude, C. 2019. Enhancing Python Compiler Error Messages via Stack Overflow. arXiv.
[97] Thurlings, M. et al. 2013. Understanding feedback: A learning theory perspective. *Educational Research Review*. 9, (Jun. 2013), 1–15. DOI:https://doi.org/10.1016/j.edurev.2012.11.004.
[98] Traver, V.J. 2010. On Compiler Error Messages: What They Say and What They Mean. *Advances in Human-Computer Interaction*. 2010, (Jan. 2010). DOI:https://doi.org/10.1155/2010/602570.
[99] Tsai, C.-Y. 2019. Improving students' understanding of basic programming concepts through visual programming language: The role of self-efficacy. *Computers in Human Behavior*. 95, (Jun. 2019), 224–232. DOI:https://doi.org/10.1016/j.chb.2018.11.038.
[100] Van der Kleij, F.M. et al. 2015. Effects of Feedback in a Computer-Based Learning Environment on Students' Learning Outcomes: A Meta-Analysis. *Review of Educational Research*. 85, 4 (Dec. 2015), 475–511. DOI:https://doi.org/10.3102/0034654314564881.
[101] Venables, A. et al. 2009. A closer look at tracing, explaining and code writing skills in the novice programmer. (Jan. 2009), 117–128.
[102] Wang, K. et al. 2018. Search, align, and repair: data-driven feedback generation for introductory programming exercises. *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Jun. 2018), 481–495.
[103] Watson, C. et al. 2012. BlueFix: using crowd-sourced feedback to support programming students in error diagnosis and repair. *International Conference on Web-Based Learning* (2012), 228–239.
[104] Watson, C. et al. 2011. Learning Programming Languages through Corrective Feedback and Concept Visualisation. *Advances in Web-Based Learning - ICWL 2011* (Berlin, Heidelberg, 2011), 11–20.
[105] Watson, C. et al. 2014. No tests required: comparing traditional and dynamic predictors of programming success. *Proceedings of the*

45th ACM technical symposium on Computer science education (New York, NY, USA, Mar. 2014), 469–474.

[106] Watson, C. et al. 2013. Predicting Performance in an Introductory Programming Course by Logging and Analyzing Student Programming Behavior. (Jul. 2013), 319–323.

[107] Whalley, J. and Lister, R. 2009. The BRACElet 2009.1 (Wellington) specification. (Jan. 2009), 9–18.

[108] Williams, L. et al. 2002. In Support of Pair Programming in the Introductory Computer Science Course. *Computer Science Education*. 12, (Sep. 2002). DOI:https://doi.org/10.1076/csed.12.3.197.8618.

[109] Zeller, A. 2000. Making Students Read and Review Code. *Proceedings of the 5th Annual SIGCSE/SIGCUE ITiCSEconference on Innovation and Technology in Computer Science Education* (New York, NY, USA, 2000), 89–92.

Apéndice: estadísticos descriptivos de los errores de compilación

	Control					Experimental				
	Mean	SD	Median	Min.	Max.	Mean	SD	Median	Min.	Max.
CE_1	4.65	8.68	0.5	0	54	4.14	5.94	2	0	32
CE_2	1.33	1.76	1	0	8	1.36	1.84	1	0	9
CE_3	1.67	5.42	0	0	46	1.38	2.40	0	0	11
CE_4	1.38	2.92	0	0	19	1.45	3.62	0	0	27
CE_5	1.21	2.37	0	0	12	0.90	1.79	0	0	10
CE_6	1.82	2.58	0	0	12	1.19	1.69	0	0	7
CE_7	0.87	2.62	0	0	16	0.87	3.29	0	0	25
CE_8	0.91	1.61	0	0	6	0.94	1.84	0	0	8
CE_9	0.62	1.16	0	0	5	0.78	1.38	0	0	7
CE_10	0.47	0.82	0	0	4	0.47	0.80	0	0	3
CE_11	0.58	1.44	0	0	7	0.79	1.44	0	0	9
CE_12	0.40	1.28	0	0	9	0.36	0.72	0	0	4
CE_13	0.55	1.42	0	0	8	0.43	01.04	0	0	5
CE_14	0.26	0.69	0	0	3	0.26	0.85	0	0	6
CE_15	0.26	0.75	0	0	4	0.38	1.23	0	0	8
CE_16	0.13	0.41	0	0	2	0.13	0.47	0	0	3
CE_17	0.23	0.66	0	0	4	0.13	0.52	0	0	3
CE_18	0.03	0.16	0	0	1	0.03	0.23	0	0	2

Tabla A1. Estadísticos descriptivos de los errores de compilación por estudiante. Sesión Divide y Vencerás (D&C).

	Control					Experimental				
	Mean	SD	Median	Min.	Max.	Mean	SD	Median	Min.	Max.
CE_1	5.44	10.96	1	0	72	0.96	2.11	0	0	11
CE_2	1.86	2.95	0	0	13	1.14	1.96	0	0	11
CE_3	1.51	3.19	0	0	14	0.69	1.78	0	0	10
CE_4	02.06	3.42	0	0	18	0.38	0.99	0	0	6
CE_5	1.92	3.25	0	0	17	0.40	01.03	0	0	4
CE_6	0.87	02.05	0	0	9	0.04	0.25	0	0	2
CE_7	01.06	2.62	0	0	14	0.26	01.04	0	0	8
CE_8	0.65	1.17	0	0	5	0.19	0.51	0	0	2
CE_9	0.71	1.56	0	0	7	0.25	0.57	0	0	2
CE_10	0.64	1.16	0	0	5	0.35	0.89	0	0	4
CE_11	0.78	2.21	0	0	14	0.25	1.11	0	0	9
CE_12	0.45	01.01	0	0	6	0.16	0.69	0	0	5
CE_13	0.29	0.77	0	0	4	0.06	0.30	0	0	2
CE_14	0.28	0.75	0	0	4	0.03	0.23	0	0	2
CE_15	0.32	0.99	0	0	6	0.21	0.64	0	0	4
CE_16	0.18	0.60	0	0	4	0.03	0.16	0	0	1
CE_17	0.12	0.43	0	0	3	0.05	0.28	0	0	2
CE_18	0,09	0,37	0	0	2	0,1	0,45	0	0	3

Tabla A2. Estadísticos descriptivos de los errores de compilación por estudiante. Sesión Algoritmos Voraces (Greedy).

	Control					Experimental				
	Mean	SD	Median	Min.	Max.	Mean	SD	Median	Min.	Max.
CE_1	1.99	4.51	0	0	24	0.26	0.97	0	0	6

CE_2	0.79	1.48	0	0	6	0.30	01.05	0	0	5
CE_3	0.29	0.97	0	0	7	0.05	0.32	0	0	2
CE_4	0.32	0.88	0	0	5	0.14	0.53	0	0	3
CE_5	0.26	0.73	0	0	4	0.08	0.42	0	0	3
CE_6	0.17	0.89	0	0	6	0.08	0.68	0	0	6
CE_7	0.40	1.95	0	0	15	0.03	0.23	0	0	2
CE_8	0.27	0.70	0	0	4	0.05	0.28	0	0	2
CE_9	0.28	0.79	0	0	4	0.16	0.51	0	0	3
CE_10	0.40	1.42	0	0	10	0.19	0.59	0	0	3
CE_11	0.06	0.37	0	0	3	0.03	0.16	0	0	1
CE_12	0.06	0.29	0	0	2	0.08	0.39	0	0	3
CE_13	0.09	0.51	0	0	4	0.04	0.25	0	0	2
CE_14	0.10	0.38	0	0	2	0.01	0.11	0	0	1
CE_15	0.10	0.41	0	0	2	0.03	0.23	0	0	2
CE_16	0.06	0.25	0	0	1	0.00	0.00	0	0	0
CE_17	0.05	0.22	0	0	1	0.03	0.16	0	0	1
CE_18	0.06	0.34	0	0	2	0.01	0.11	0	0	1

Tabla A3. Estadísticos descriptivos de los errores de compilación por estudiante. Sesión Programación Dinámica (DP).

	Control					Experimental				
	Mean	SD	Median	Min.	Max.	Mean	SD	Median	Min.	Max.
CE_1	05.06	9.57	1	0	48	2.91	8.31	0	0	48
CE_2	1.22	2.68	0	0	13	0.65	2.54	0	0	18
CE_3	1.69	4.98	0	0	26	0.21	0.68	0	0	4
CE_4	0.95	2.33	0	0	16	0.55	1.75	0	0	13

CE_5	0.40	01.01	0	0	5	0.26	1.24	0	0	10
CE_6	0.73	2.93	0	0	25	0.39	2.43	0	0	21
CE_7	0.60	2.10	0	0	14	0.25	0.76	0	0	3
CE_8	0.40	0.90	0	0	5	0.18	0.58	0	0	3
CE_9	0.44	0.97	0	0	5	0.21	0.52	0	0	3
CE_10	0.42	1.17	0	0	7	0.21	0.68	0	0	4
CE_11	0.47	1.79	0	0	14	0.14	0.64	0	0	5
CE_12	0.55	3.57	0	0	31	0.05	0.22	0	0	1
CE_13	0.18	0.72	0	0	4	0.06	0.41	0	0	3
CE_14	0.44	2.96	0	0	26	0.04	0.25	0	0	2
CE_15	0.13	0.52	0	0	4	0.03	0.16	0	0	1
CE_16	0.08	0.35	0	0	2	0.04	0.19	0	0	1
CE_17	0.04	0.19	0	0	1	0.06	0.30	0	0	2
CE_18	0.10	0.35	0	0	2	0.04	0.25	0	0	2

Tabla A4. Estadísticos descriptivos de los errores de compilación por estudiante. Sesión Backtracking (BT)

	Control					Experimental				
	Mean	SD	Median	Min.	Max.	Mean	SD	Median	Min.	Max.
CE_1	17.14	20.44	11.5	0	116	8.27	11.34	5	0	61
CE_2	5.21	5.37	3	0	21	3.45	4.39	2	0	27
CE_3	5.17	8.94	2	0	46	2.32	2.90	2	0	12
CE_4	4.72	5.93	3	0	37	2.52	4.13	1	0	27
CE_5	3.78	4.59	2	0	19	1.64	2.67	1	0	12
CE_6	3.59	4.97	2	0	28	1.70	2.97	0	0	21
CE_7	2.94	5.00	0.5	0	29	1.40	4.34	0	0	33

CE_8	2.23	2.88	1	0	13	1.36	1.93	1	0	8
CE_9	02.04	2.63	1	0	14	1.39	1.76	1	0	8
CE_10	1.94	2.65	1	0	14	1.22	1.89	1	0	10
CE_11	1.90	3.76	0	0	20	1.21	1.84	0	0	9
CE_12	1.46	3.82	0	0	31	0.65	0.98	0	0	5
CE_13	1.12	1.98	0	0	8	0.60	1.33	0	0	7
CE_14	01.08	3.21	0	0	26	0.34	0.90	0	0	6
CE_15	0.81	1.48	0	0	7	0.64	1.64	0	0	10
CE_16	0.45	0.92	0	0	6	0.19	0.51	0	0	3
CE_17	0.44	0.85	0	0	4	0.27	0.70	0	0	4
CE_18	0.28	0.68	0	0	3	0.18	0.58	0	0	3

Tabla A5. Estadísticos descriptivos de los errores de compilación por estudiante. Totales.