UNIVERSIDAD
COMPLUTENSE
MADRID

Proyecto de Innovación Convocatoria

2024/2025

Nº de proyecto: 71

Mejora de la retroalimentación de los jueces en línea mediante análisis estático e instrumentación

Responsable del proyecto:

Rubén Rafael Rubio Cuéllar

Facultad de Informática

Departamento: Sistemas Informáticos y Computación

Índice general

1 Objetivos propuestos en la presentación del proyecto

En el aprendizaje de la programación es fundamental la práctica y en esta disciplina autoevaluarse es relativamente sencillo: los estudiantes pueden ejecutar sus programas para comprobar si producen los resultados esperados. Un juez automático es una herramienta que facilita esta labor presentando a los estudiantes una serie de problemas a resolver y dándoles la oportunidad de enviar sus soluciones para que compruebe si son correctas. Esto ha convertido a los jueces automáticos en una herramienta muy valiosa en las asignaturas de programación. En la Facultad de Informática de esta universidad hace algún tiempo que se vienen utilizando en diversas asignaturas, desde el primer curso hasta las más avanzadas.

El juez automático toma habitualmente la forma de una página web. Cada problema consta de un enunciado y una serie de casos de prueba con sus respuestas esperadas elaborados por el profesor (y ocultos a los estudiantes, salvo una pequeña muestra). El estudiante sube el código de su pretendida solución a esta página, el juez la ejecuta, compara sus resultados con los esperados y muestra el resultado al poco tiempo. Sin embargo, la respuesta que obtiene el aprendiente es poco informativa y se limita a unas pocas palabras: *correcto*, *respuesta errónea*, *límite de tiempo excedido* o *error de compilación*. En cambio, el profesor tiene acceso a demasiada información sin procesar, las diferencias brutas entre el texto emitido por el programa y el esperado, en las que es complicado encontrar con agilidad la causa del problema. Esta parquedad en la respuesta se debe a que el propósito inicial de los jueces automáticos es dar soporte a concursos de programación, donde es preferible que la respuesta sea escueta.

El análisis automático de código ha mejorado mucho recientemente con herramientas activamente desarrolladas, de código abierto y accesibles para el público en general. No solo se ocupan de meras cuestiones de estilo, que también pueden ser interesantes, sino también de problemas semánticos más profundos. Por un lado, el análisis estático de programas se ocupa de detectar estos errores examinando el código sin llegar a ejecutarlo. Por otro lado, la instrumentación consiste en añadir código adicional a un programa para facilitar la detección de errores u obtener información más precisa cuando estos se producen en ejecución.

El objetivo principal de este proyecto es desarrollar una extensión del juez automático DOMjudge [5] que proporcione información más significativa a los estudiantes y los profesores en sus veredictos utilizando herramientas de análisis de código e instrumentación. Su diseño estará inspirado por los siguientes principios:

- Muchos de los errores habituales de los estudiantes al programar pueden ser diagnosticados por las herramientas de análisis de código e instrumentación existentes.
- El repertorio de comprobaciones que se ejecutará y se expondrá a los estudiantes será escogido a partir de la experiencia previa y el análisis de envíos de cursos pasados. Se evitarán falsos positivos y avisos que puedan confundir al estudiante.
- El profesor podrá escoger las comprobaciones que quiera para cada problema y ajustar la forma en la que esa información se proporciona al estudiante, sin que ello suponga una carga excesiva de trabajo.
- El administrador del servidor podrá añadir nuevas herramientas de diagnóstico e introducir comprobaciones personalizadas con relativa facilidad.
- La herramienta se desarrollará sobre DOMjudge para aprovechar la robustez de un proyecto activo, reducir el esfuerzo de mantenimiento y aumentar la vida útil del programa.
- La extensión se distribuirá como software libre y será fácilmente instalable por usuarios ajenos al desarrollo.

Los objetivos concretos de la propuesta original del proyecto fueron los siguientes (se mantienen los tiempos verbales en su forma original):

(O1) Elaboración de una herramienta para la mejora de la retroalimentación de DOMjudge mediante análisis de código e instrumentación

En el momento en el que se presentó la propuesta, un prototipo inicial ya estaba en desarrollo y gran parte de la funcionalidad deseable se conseguiría incluir al finalizar el TFG de los estudiantes participantes. Sin embargo, se plantean otros aspectos no abordados en el TFG para continuar con el desarrollo del prototipo:

- Recuperar mayor cantidad de información de los diagnósticos de las herramientas para incorporarlos en la retroalimentación a los estudiantes.
- Flexibilidad en las opciones del profesor para mostrar pistas al estudiante. Se desarrollará un lenguaje sencillo para expresar estas opciones.
- Integración de análisis específicos definidos por el profesor para un problema concreto. Se puede utilizar para ello la biblioteca CAC++ [4], desarrollada por profesores de la Universidad de Cádiz, que se construye sobre la API del compilador Clang.
- Integración con una interfaz utilizada en cursos pasados para seguir los envíos en el laboratorio con información detallada y fotografías para identificar mejor a los estudiantes.

(O2) Despliegue de la herramienta en los servidores de jueces automáticos de la Facultad de Informática

Se instalará la herramienta en los servidores de jueces automáticos de la Facultad de Informática, donde estará disponible para todos los profesores que quieran usarla. Eso supondrá instalar los analizadores de código y herramientas de instrumentación necesarias en los *judgehosts*, los entornos aislados de ejecución donde se compilan y ejecutan los envíos de los estudiantes. Será necesario adaptar el código de DOMjudge para integrar la retroalimentación en su interfaz y para activar la ejecución de los analizadores.

(O3) Utilización de la herramienta en diversas asignaturas por parte de los profesores participantes

Se utilizará la herramienta durante el curso en asignaturas de programación, algoritmia y estructuras de datos como *Técnicas Algorítmicas en Ingeniería del Software* o *Estructuras de Datos*, entre otras, que impartirán varios de los profesores participantes. Esto supondrá adaptar la configuración de los problemas para que ejecuten las comprobaciones de la herramienta.

Se evaluará la efectividad de la herramienta durante el curso con el fin de mejorarla de forma continua, tanto en el plano tecnológico como en del aprendizaje (problemas resueltos y no resueltos después de obtener retroalimentación, número de intentos necesarios, reiteración de los errores advertidos en otros problemas, etc.).

(O4) Difusión de la herramienta

Se realizarán actividades encaminadas a difundir su uso entre otros miembros de la comunidad universitaria. Al final del curso, se participará en las jornadas de innovación docente de la Facultad de Informática, donde se mostrará su funcionamiento y los resultados obtenidos. También se enviarán comunicaciones a congresos o jornadas educativas dedicadas al uso de las TIC en el aprendizaje: AprendeTIC, JENUI u otras.

2 Objetivos alcanzados

El objetivo principal de este proyecto ha sido alcanzado: se ha desarrollado una herramienta que mejora la retroalimentación proporcionada a través del juez automático con los resultados del análisis estático y la ejecución instrumentada de los envíos. Esta herramienta, que hemos denominado DOMLab [7], se ha utilizado en varias asignaturas durante el curso 2024/25. Cada uno de los objetivos específicos también se han cumplido, aunque en algunos casos ha habido variaciones respecto al planteamiento inicial. El trabajo partía del TFG [10], aunque el prototipo se ha rehecho desde cero.

(O1) Elaboración de una herramienta para la mejora de la retroalimentación de DOMjudge mediante análisis de código e instrumentación

El proyecto parte del trabajo de fin de grado «Code analysis and instrumentation of submissions for online judges» [10] de los estudiantes miembros de este proyecto, Luis Esteban, Juan Trillo y Rodrigo Burgos, dirigido por Rubén Rubio. En ese trabajo se estudiaron las comprobaciones más relevantes de herramientas como Clang-Tidy [6], Cppcheck [3] o los *sanitizers* de Google [1] y su efectividad en los envíos del juez automático de la asignatura *Técnicas Algorítmicas en Ingeniería del Software* en el curso 2023/2024. También se desarrolló un primer prototipo que explotaba las posibilidades de configuración de DOMjudge e incluso modificaba su implementación para ejecutar las comprobaciones en la infraestructura del juez automático [11, 9]. Los resultados de ese estudio y las modificaciones están disponibles en abierto en las referencias anteriores.

Sin embargo, la herramienta actual que se ha usado durante el curso es muy distinta a la desarrollada en el trabajo de fin de grado, tanto en diseño como en implementación. Las comprobaciones, en lugar de ejecutarse en la infraestructura de DOMjudge, se ejecutan como un servicio externo que obtiene los envíos en tiempo real a través del flujo de eventos del juez automático y obtiene el resto de información a través de su API (y también mediante *web scraping*, por limitaciones de esta). Esta arquitectura es más versátil e interfiere en menor medida con el funcionamiento normal de los jueces automáticos. Se ofrecen más detalles en el apartado 5.

Respecto a los objetivos concretos especificados en el apartado 1:

- La captura de los mensajes de diagnóstico se ha mejorado para incluir la ubicación del error en el código y recuperar su información asociada. Además, se recupera un mayor número de tipos de error y se han implementado comprobaciones específicas basadas en el análisis del árbol de sintaxis abstracta (AST). Los errores soportados se especifican en formato JSON junto con una descripción breve y una explicación larga en castellano para el estudiante, un nivel de severidad y los veredictos del juez que podrían darse como consecuencia del error, para focalizar los mensajes mostrados.

- La implementación de la herramienta está divida en dos componentes:

  1. DOMLab, que se encarga de la interacción con el juez automático (seguimiento de los envíos, obtención del mensaje de retroalimentación, etc.). Funciona como un oyente de los eventos del juez automático y a la vez como un servidor que permite integrar los diagnósticos en DOMjudge con una pequeña modificación en una de sus plantillas. Además, proporciona una interfaz web separada para que los profesores puedan acceder a los diagnósticos.
  2. Codegavel, una biblioteca y programa que se encarga de ejecutar los análisis sobre los envíos. Soporta análisis estáticos predefinidos con Clang-Tidy, análisis estáticos

personalizados con la biblioteca `clang.cindex` y ejecuciones instrumentadas con los *sanitizers*. Los programas se ejecutan en un entorno confinado con límites de tiempo y memoria, sin acceso a internet y otras restricciones.

DOMLab utiliza Codegavel, pero el segundo puede utilizarse de forma independiente (y se ha utilizado como ayuda en la corrección de pruebas de evaluación de forma local).

No se ha desarrollado un lenguaje sencillo para controlar la información que se muestra al estudiante, aunque Codegavel está parametrizado por unos archivos JSON de descripción de los avisos soportados que incluyen explicaciones configurables.

- No se ha utilizado la biblioteca CAC++ [4] a la que se hacía referencia en la propuesta inicial, pero programar comprobaciones utilizando el envoltorio de Python `clang.cindex` sobre la API de C de Clang es relativamente sencillo. El programa Codegavel incluye algunas comprobaciones predefinidas desarrolladas utilizando este método. Una razón para no utilizar CAC++ es que está desarrollada para una versión antigua de Clang y no funciona tal cual con versiones más recientes.
- Se ha integrado la retroalimentación producida por Codegavel en una interfaz web para el profesor que muestra en tiempo real los envíos al juez, su veredicto e identifica a los autores (mostrando sus nombres y opcionalmente una fotografía).

(O2) Despliegue de la herramienta en los servidores de jueces automáticos de la Facultad de Informática

La herramienta DOMLab se ha instalado en un servidor de la facultad, desde donde se conecta a los flujos de eventos de diversos concursos alojados en tres de los cuatro servidores de jueces automáticos de la facultad. En estos servidores se ha dado de alta a un usuario para permitir el acceso a DOMLab y se ha realizado una pequeña modificación en las plantillas de DOMjudge para integrar los mensajes.

(O3) Utilización de la herramienta en diversas asignaturas por parte de los profesores participantes

La herramienta se ha utilizado parcialmente en diversas asignaturas por parte de los profesores participantes: en *Técnicas Algorítmicas en Ingeniería del Software* (tercer curso del Grado en Ingeniería del Software), *Fundamentos de la Programación 2* (primer curso del doble grado en Ingeniería Informática y Matemáticas) y *Estructuras de Datos* (segundo curso del mismo doble grado). También se ha usado ocasionalmente en otras asignaturas y en exámenes, aunque sin mostrar los avisos a los estudiantes. El prototipo se ha ido mejorando continuamente a partir de la experiencia de su utilización práctica.

(O4) Difusión de la herramienta

La herramienta se ha presentado en la VIII Jornada de Innovación Docente de la Facultad de Informática celebrada el lunes 9 de junio a las 10:10[1]. Varios profesores se interesaron en activarla en sus asignaturas para el próximo curso. Antes de ello, la herramienta también se ha dado a conocer informalmente a algunos profesores cercanos y se ha utilizado ocasionalmente en otras asignaturas. Se prevé que con los resultados obtenidos, tal vez sumados a los que se pudieran obtener en cursos posteriores, se prepare una comunicación a alguno de los congresos o jornadas educativas enumeradas en la planificación.

[1] https://informatica.ucm.es/viii-jornada-de-innovacion-docente

3 Metodología empleada en el proyecto

En la propuesta inicial del proyecto se identificaron las siguientes nueve tareas. No ha habido desviaciones significativas sobre el plan previsto, salvo que la tarea 9 no se ha llegado a realizar por falta de tiempo. La figura 1 muestra la distribución temporal de las tareas del proyecto.

- **Tarea 1.** Estudio, filtrado y categorización de las comprobaciones disponibles en una selección de analizadores de código.
- **Tarea 2.** Desarrollo de la herramienta para analizar el código de los envíos y proporcionar retroalimentación.
- **Tarea 3.** Despliegue de la herramienta en los jueces automáticos utilizados en los laboratorios de la FdI.
- **Tarea 4.** Configuración de los problemas propuestos para aplicarles las comprobaciones.
- **Tarea 5.** Pruebas de usabilidad de la integración en DOMjudge.
- **Tarea 6.** Utilización de la herramienta en clase.
- **Tarea 7.** Revisión de los resultados obtenidos en la detección de errores y en el aprendizaje por parte de los estudiantes para mejorar la herramienta.
- **Tarea 8.** Difusión de la herramienta entre profesores.
- **Tarea 9.** Cuestionario a los estudiantes y los profesores sobre la utilidad de los comentarios automáticos.

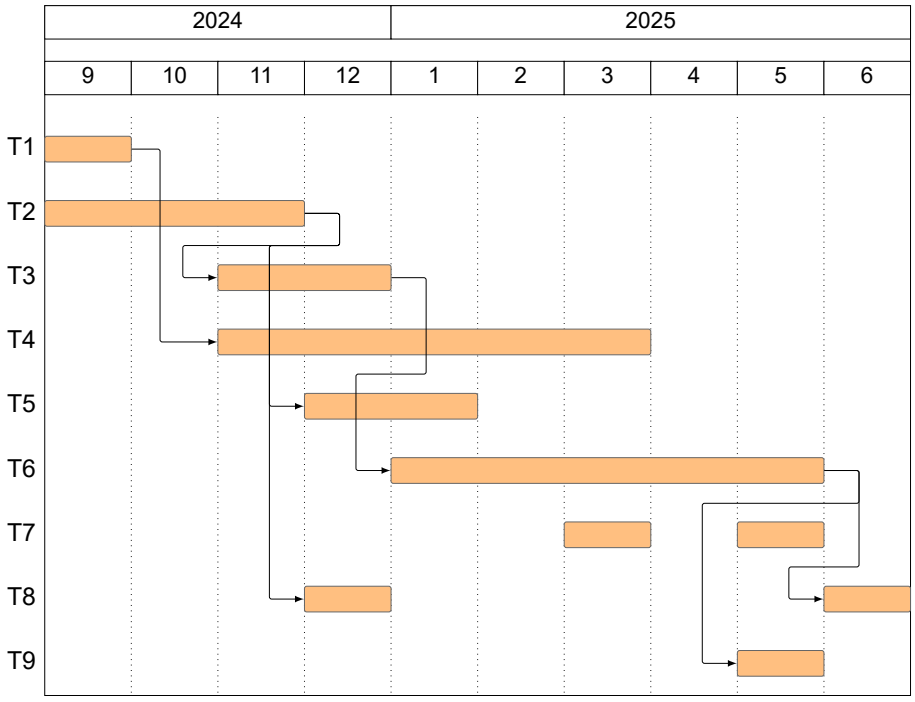


Figura 1: Diagrama de Gantt de las tareas del proyecto.

4 Recursos humanos

El equipo está formado por cinco profesores (Enrique Martín, Manuel Montenegro, Adrián Riesco, Rubén Rubio y Alberto Verdejo) y tres estudiantes (Luis Esteban, Juan Trillo y Rodrigo Burgos). Estos últimos son los autores del TFG [10], dirigido por Rubén Rubio, que ha servido de base a este proyecto.

- Rubén Rubio es profesor ayudante doctor desde 2022 en la Facultad de Informática de la UCM, donde ha impartido múltiples asignaturas de programación y algoritmos de primer, segundo y tercer curso. En varias de ellas ha utilizado jueces automáticos y en particular DOMjudge, así como en los concursos de programación en los que ha sido voluntario. Ha recibido una evaluación muy positiva en Docentia y ha participado en 3 proyectos Innova-Docencia y en diversas jornadas de innovación.

- Adrián Riesco ha impartido docencia en las facultades de Informática y Matemáticas, en las que ha usado jueces para la evaluación. Asimismo, ha impartido docencia en el Máster en Letras Digitales, cuya naturaleza semipresencial hace especialmente importante el uso de herramientas de evaluación no presenciales que ofrezcan retroalimentación inmediata. Ha participado en 12 proyectos Innova-Docencia desde 2011 y ha obtenido una evaluación “excelente” en las dos últimas evaluaciones trienales del programa Docentia.

- Enrique Martín ha participado en 12 proyectos Innova-Docencia desde 2007, siendo el responsable de 5 de ellos. La mayoría de estos proyectos han tratado sobre el desarrollo de herramientas informáticas para la mejora del aprendizaje de lenguajes de programación, su utilización en el aula y la evaluación de su impacto real en aprendizaje. Concretamente, ha participado en diversos proyectos de innovación educativa relacionados con el juez DOMjudge.

- Manuel Montenegro ha participado en 9 proyectos Innova-Docencia, siendo el responsable de tres de ellos. Durante los últimos cinco cursos académicos ha aplicado la metodología de clase invertida en la asignatura de Estructuras de Datos en la Facultad de Informática, donde los estudiantes hacen un uso intensivo de DOMjudge para la realización de prácticas. Tiene experiencia en desarrollo de aplicaciones que hacen uso de la API de DOMjudge, y ha codirigido un TFG sobre retroalimentación de jueces [8] (DomFeed) y otro sobre jueces multilenguaje [2] (Scholarjudge). Su experiencia en el desarrollo de aplicaciones web (impartió esta asignatura en el Grado en Ingeniería del Software) garantiza su idoneidad para la realización de las tareas propuestas.

- Alberto Verdejo ha impartido docencia relacionada con la algoritmia y las estructuras de datos durante 25 años. Es coautor de 3 libros dedicados a la docencia de la informática y ha participado en 9 proyectos de innovación educativa. Desde 2015 codirige concursos de programación en distintos niveles educativos utilizando jueces automáticos como DOMjudge. En ese mismo año comenzó a emplear este tipo de jueces en sus asignaturas, tanto para prácticas como para exámenes. Desde 2019 se ha encargado además de la configuración de DOMjudge en las asignaturas de programación y algoritmia de la facultad.

- Rodrigo Burgos, Juan Trillo y Luis Esteban son estudiantes del grado en Ingeniería Informática. Han cursado múltiples asignaturas en las que se utilizaban jueces automáticos y son conocedores de la problemática que supone la falta de información en la retroalimentación.

5 Desarrollo de las actividades

En esta sección se detalla el desarrollo de los objetivos y tareas descritas en los apartados anteriores.

5.1 Reimplementación del prototipo

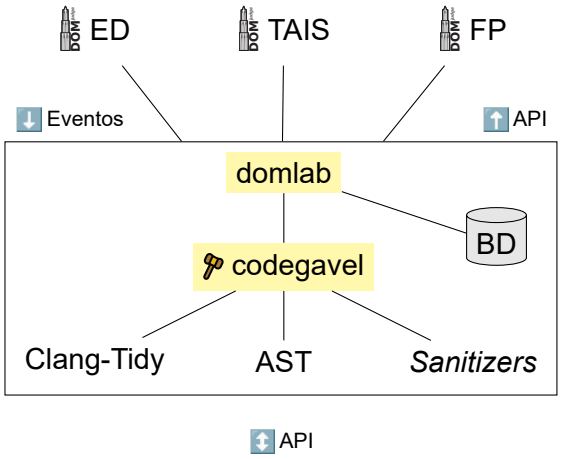


Figura 2: Arquitectura de DOMLab y Codegavel.

El apartado 2 ya ha introducido una visión general de la arquitectura actual de la herramienta elaborada, que se resume en la figura 2. El primer prototipo de la herramienta, desarrollado en [10], consistía en unas ingeniosas modificaciones en la configuración y el código PHP del juez automático DOMjudge para incluir las comprobaciones pretendidas como lotes especiales de casos de prueba. La principal ventaja de este método es que está integrado en el juez y no se necesita ningún servicio o equipo externo para ejecutarlo. Sin embargo, tiene algunos inconvenientes que hemos solventado con un diseño sustancialmente distinto en el nuevo prototipo desarrollado durante el proyecto:

- La Facultad de Informática dispone de más de cinco servidores de jueces automáticos distintos, aproximadamente uno por asignatura. Si el prototipo estuviera integrado en el juez y fuera necesario hacer una corrección o modificación durante el desarrollo, habría que replicarla en todas las instancias. Además, un posible error en la configuración podría afectar o dejar sin servicio a los jueces automáticos, también para aquellos grupos que no hacen uso de nuestra herramienta.

  El nuevo prototipo se ejecuta como un servidor externo que escucha los flujos de eventos de la API de DOMjudge. A través de él obtiene notificaciones de los envíos recibidos, se los descarga y los procesa de forma centralizada en una sola máquina. El acceso al servidor para hacer cambios es más sencillo y estos se aplican directamente a todas las instancias.

- Las posibilidades de DOMjudge para introducir retroalimentación a los estudiantes son limitadas. La fase de compilación del envío sí proporciona información (la salida del compilador) al usuario, pero no es personalizable a nivel de problema ni a nivel de concurso. La fase de ejecución solo permite añadir retroalimentación a través de una característica muy reciente `teammessages.txt` que no funciona completamente bien.

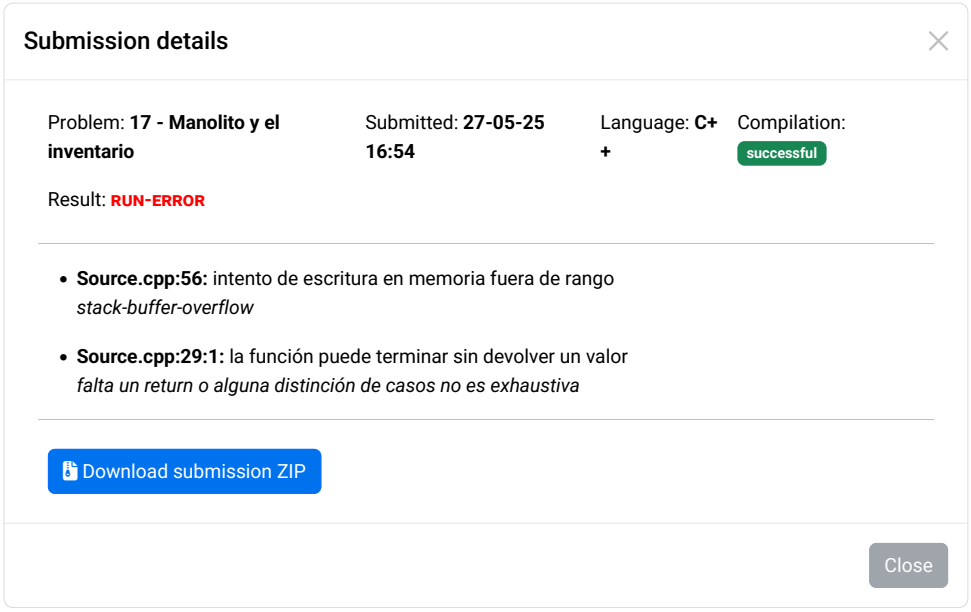


Figura 3: Captura del cuadro de información con un ejemplo de errores detectados.

El nuevo prototipo añade la retroalimentación a través de una modificación menor de la plantilla del cuadro de diálogo de información detallada del envío. Ahí se introduce un breve fragmento de JavaScript que descarga la retroalimentación del servidor que ejecuta la herramienta y la muestra en dicho cuadro de diálogo, como se puede ver en la figura 3.

- Generar la retroalimentación en la fase de ejecución del juez automático entraña ciertas dificultades si se quiere enriquecer con información del contexto, ya que este no es fácilmente accesible. Por ejemplo, se podría querer hacer depender la retroalimentación de la fecha o del número de intentos (más general al principio, más precisa en intentos posteriores). Las ejecuciones del juez automático tienen lugar en un entorno confinado sin acceso al exterior y por tanto es difícil hacer estos cálculos integrados en DOMjudge.

  En el nuevo prototipo, la generación de la retroalimentación tiene acceso a una base de datos con todos los envíos previos y los avisos que se generaron para ellos. Esto permite personalizar mejor la retroalimentación, aunque no se ha explorado en profundidad esta posibilidad en el proyecto.

No obstante, además de las ventajas, la arquitectura escogida tiene algunos inconvenientes que han dificultado el desarrollo del proyecto. El servidor necesita una configuración especial para que el flujo de eventos funcione en tiempo real, este no incluye toda la información deseable y a veces presenta ciertas inconsistencias[2]. Por otro lado, cierta información necesaria para ejecutar las comprobaciones (casos de prueba, límites de tiempo y memoria, etc.) no están disponibles a través de la API de DOMjudge y es necesario hacer *web scraping* (i.e. interactuar con la página web como hiciera un internauta) para obtenerla.

La herramienta se ha ido adaptando y mejorando continuamente durante el curso. Por ejemplo, el número de envíos con errores de ejecución que consigue explicar la herramienta es del 50 % a lo largo del curso, pero supera el 90 % en el tramo final.

[2]Son ejemplos de fallos reportados o mejoras sugeridas durante el proyecto: https://github.com/DOMjudge/domjudge/issues/2974 y https://github.com/DOMjudge/domjudge/issues/2728.

Asignatura	Matriculados	Envíos	Con aviso (%)	Avisos
ED (DG)	35	3294	9,87	564
FP2 (DG)	35	2396	16,32	543
TAIS (GIS)	119	8194	8,20	1107
Examen ED	35	225	27,56	106
Examen FP2	35	196	19,90	63

Cuadro 1: Asignaturas con número de envíos y avisos a 8 de junio.

5.2 Utilización en las clases

La herramienta se ha utilizado en varias asignaturas impartidas por miembros del proyecto:

- *Técnicas Algorítmicas en Ingeniería del Software* (TAIS) es una asignatura del primer cuatrimestre del tercer curso del grado en Ingeniería del Software, impartida por Rubén Rubio. La herramienta ha estado activa en dos fases separadas: en los meses finales del curso en una versión preliminar y un mes antes del examen de la convocatoria extraordinaria con la versión ya depurada. Se informó a los estudiantes de la existencia de esta retroalimentación en clase y mediante un aviso del campus virtual.
- *Fundamentos de la Programación 2* (FP2) es una asignatura de primer curso en la mayoría de grados de la Facultad de Informática. El grupo donde se ha utilizado es el del doble grado de Ingeniería Informática y Matemáticas con Alberto Verdejo como profesor.
- *Estructuras de Datos* (ED) es una asignatura del segundo cuatrimestre del segundo curso en la mayoría de grados de la facultad. Se ha utilizado en el grupo del doble Grado en Ingeniería Informática y Matemáticas, con Alberto Verdejo como docente.

También se ha utilizado en algunos exámenes de estas y otras asignaturas, aunque la retroalimentación solo era visible para los profesores. El cuadro 1 muestra el número de estudiantes, envíos, avisos de retroalimentación y el porcentaje de envíos con retroalimentación en las asignaturas mencionadas a fecha del 8 de junio de 2025.

5.3 Análisis de resultados

Se han analizado algunos aspectos de los resultados obtenidos con la herramienta en su aplicación en las asignaturas anteriormente expuestas. En particular, se han analizado los tipos de aviso más frecuentes en las distintas asignaturas. El apéndice 6.3 muestra los resultados detallados, pero cabe destacar que los accesos a memoria fuera de rango son el aviso más habitual. En las asignaturas ED y FP2, las fugas de memoria son un error muy frecuente, pero no así en TAIS, donde no se reserva memoria expresamente. Por último, un aviso con mucha incidencia es `performance-avoid-endl`, pero en la mayoría de los casos no es acertado. La posibilidad de limitar ciertos avisos a problemas específicos sería útil aquí para limitarlo a contextos donde es significativo.

No se ha realizado de momento un análisis de la efectividad de la retroalimentación, es decir, de cómo ayuda esta al estudiante a resolver los problemas de sus soluciones. Se aprecian casos donde el mensaje de la herramienta va seguido de un nuevo envío en el que se corrige el problema, pero es necesario un análisis exhaustivo y sistemático para determinar si este patrón es significativo. Este análisis será posible gracias a los registros recopilados sobre envíos, diagnósticos y consulta de los mismos a través de DOMjudge.

6 Anexos

6.1 Bibliografía

[1] *AddressSanitizer, ThreadSanitizer, MemorySanitizer*. 2011. URL: https://github.com/google/sanitizers (visitado 2025-06-23).

[2] Marta Estévez Bravo, Pablo Morientes Lavín y Víctor Manuel Cavero Gracia. «Juez multilenguaje para el aprendizaje de la programación». TFG en Ingeniería Informática. Universidad Complutense de Madrid, 2022. HDL: 20.500.14352/3243.

[3] *Cppcheck - A tool for static C/C++ code analysis*. 2007. URL: https://cppcheck.sourceforge.io/ (visitado 2025-06-23).

[4] Pedro Delgado-Pérez e Inmaculada Medina-Bulo. «Customizable and scalable automated assessment of C/C++ programming assignments». En: *Comput. Appl. Eng. Educ.* 28.6 (2020), págs. 1449-1466. DOI: 10.1002/CAE.22317.

[5] Jaap Eldering, Nicky Gerritsen et al. *DOMjudge – Programming Contest Jury System*. 2025. URL: https://www.domjudge.org (visitado 2025-06-23).

[6] *Extra Clang Tools documentation - clang-tidy*. 2009. URL: https://clang.llvm.org/extra/clang-tidy/ (visitado 2025-06-23).

[7] Rubén Rubio et al. *DOMlab and Codegavel – Tracker and improved feedback by code analysis for DOMjudge contests*. 2025. URL: https://github.com/ningit/domlab (visitado 2025-06-23).

[8] Ricardo Enrique Freire Sacco et al. «Generador de retroalimentación detallada para DOMjudge». TFG en Ingeniería Informática. Universidad Complutense de Madrid, 2022. HDL: 20.500.14352/3185.

[9] Rodrigo Burgos Sosa et al. *Versión de DOMjudge modificada en el TFG [10]*. 2024. URL: https://github.com/robu02/DOMjudge.

[10] Luis Esteban Velasco, Juan Trillo Carreras y Rodrigo Burgos Sosa. «Code analysis and instrumentation of submissions for online judges». TFG en Ingeniería Informática. Universidad Complutense de Madrid, 2024. HDL: 20.500.14352/107032.

[11] Luis Esteban Velasco, Juan Trillo Carreras y Rodrigo Burgos Sosa. *Repositorio principal de código del TFG [10]*. 2024. URL: https://github.com/Jantri-3/analysis-of-DOMjudge-submissions.

6.2 Ejemplos de mensajes de retroalimentación

Se muestra a continuación algunos ejemplos más de mensajes de retroalimentación integrados en el juez automático.

Submission details

Problem: **25 - Jan el olvidado** Submitted: **26-05-25 14:06** Language: **C++** Compilation: successful

Result: **RUN-ERROR**

- **calculadora.cpp:22:15:** división entre cero
  comprueba que el divisor no se pueda anular
- **calculadora.cpp:28:1:** la función puede terminar sin devolver un valor
  falta un return o alguna distinción de casos no es exhaustiva

Download submission ZIP

Close

Submission details

Problem: **13 - Sistema de gestión de reservas de vuelos** Submitted: **09-05-25 17:01** Language: **C++** Compilation: successful

Result: **RUN-ERROR**

- excepción no capturada de tipo std::out_of_range
  basic_string::substr: __pos (which is 1) > this->size() (which is 0)

Download submission ZIP

Close

Submission details

Problem: **BM2.0 - BuscaminasV2** Submitted: **29-04-25 18:32** Language: **C++** Compilation: successful

Result: **TIMELIMIT**

- **tablero.cpp:28:** aserto incumplido
  casilla_valida(casilla)

Download submission ZIP

Close

Submission details

Problem: **17 - Manolito y el inventario** | Submitted: **27-05-25 16:00** | Language: **C++** | Compilation: successful

Result: **TIMELIMIT**

- **manolito.cpp:15:5:** el argumento de tipo `vector` con copia costosa se pasa por valor
  el paso por valor implica la copia del objeto que se pasa como argumento en cada llamada, usa const & si es posible para evitar operaciones innecesarias

Download submission ZIP

Close

Submission details

Problem: **10 - ¿A qué hora pasa el próximo tren?** | Submitted: **25-05-25 01:06** | Language: **C++** | Compilation: successful

Result: **TIMELIMIT**

- **p1.cpp:10:1:** este bucle no termina, revísalo
  this loop is infinite; none of its condition variables (trenes, horas) are updated in the loop body

Download submission ZIP

Close

Submission details

Problem: **15 - Unidad Curiosa de Monitorización** | Submitted: **25-05-25 20:18** | Language: **C++** | Compilation: successful

Result: **WRONG-ANSWER**

- **main.cpp:82:60:** las operaciones aritméticas con enteros han producido desbordamiento
  comprueba que los tipos utilizados tienen un tamaño adecuado a las restricciones del problema

Download submission ZIP

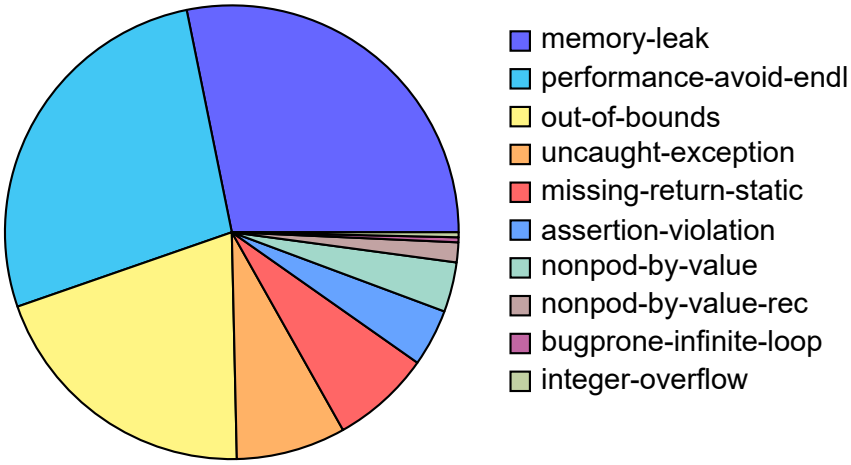
Close

6.3 Incidencia de los tipos de diagnóstico por asignatura

A continuación se muestran varios diagramas sectoriales sobre la frecuencia relativa de aparición de las diversas comprobaciones soportadas en las distintas asignaturas donde se ha utilizado la herramienta. El aviso `performance-avoid-endl` es frecuente en todas las asignaturas porque se mostraba siempre en caso de ejecución demasiado lenta, aunque no siempre es significativo y se debería limitar su aparición a ciertos problemas.

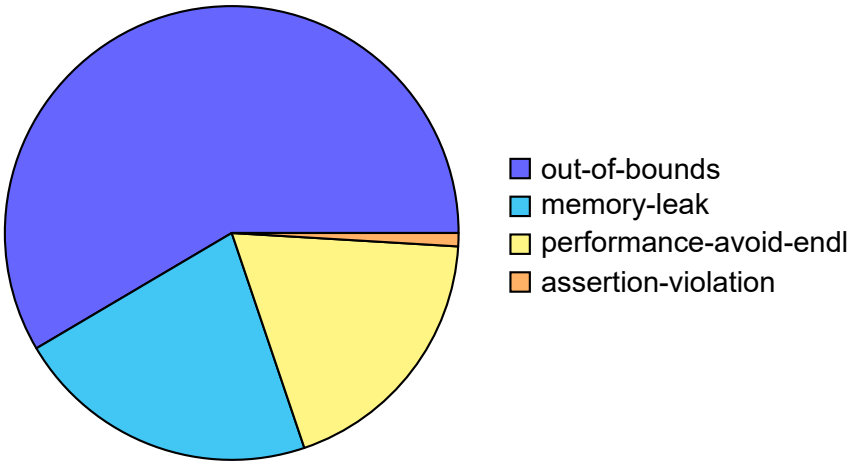
6.3.1 Estructuras de Datos

El fallo más habitual en Estructura de Datos son las fugas de memoria (`memory-leak`), el acceso a arrays o semejantes fuera de rango (`out-of-bounds`), el uso indebido de las estructuras de datos, representado en los avisos de excepción no capturada (`uncaught-exception`) y violación de un aserto (`assertion-violation`), y el olvido de la sentencia `return` en funciones que devuelven valores (`missing-return-static`). Salvo `memory-leak`, que puede aparecer en soluciones correctas funcionalmente, todos los avisos anteriores explican el veredicto *runtime error*. Los avisos `nonpod-by-value` y `nonpod-by-value-rec` indican que se ha pasado por valor una estructura de datos con copia no trivial (en el segundo caso en un contexto recursivo), lo que suele ser causa del veredicto *timelimit*. La herramienta también ha detectado unos pocos bucles no terminantes triviales (`bugprone-infinite-loop`) y desbordamiento en las operaciones aritméticas (`integer-overflow`). Este último aviso es habitualmente responsable del veredicto *wrong answer*.



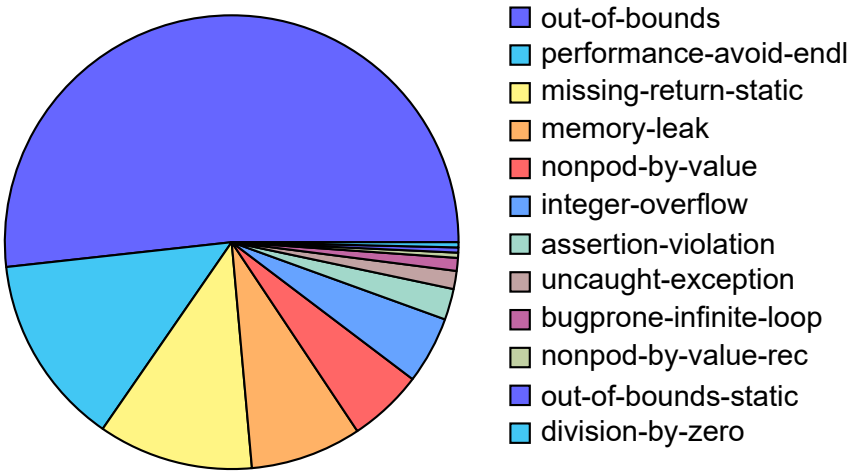
6.3.2 Examen de Estructuras de Datos

Los avisos obtenidos en el examen (no mostrados a los estudiantes) son un subconjunto de los producidos durante las clases. Como se ve en el cuadro 1, el número de envíos con mensajes de la herramienta es sustancialmente mayor.



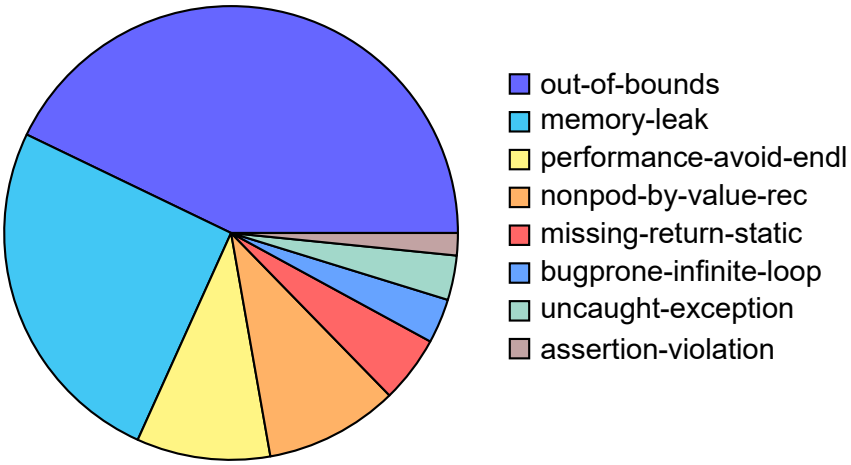
6.3.3 Fundamentos de la Programación 2

En la asignatura Fundamentos de la Programación 2 el problema más frecuente con diferencia es el acceso a arrays o semejantes fuera de rango (`out-of-bounds` y `out-of-bounds-static`, dependiendo de si ha sido detectado dinámica o estáticamente). Aparece también con baja frecuencia (2 apariciones) la división entre cero (`division-by-zero`), que es causa de *runtime error*.



6.3.4 Examen de Fundamentos de la Programación 2

Los resultados son semejantes a los obtenidos durante el curso.



6.3.5 Técnicas Algorítmicas en Ingeniería del Software

En la asignatura Técnicas Algorítmicas en Ingeniería del Software (de tercer curso) no aparecen errores por fugas de memoria, pues no se reserva o libera memoria explícitamente, sino a través de estructuras de datos predefinidas y correctas. El aviso más frecuente es el desbordamiento aritmético (`integer-overflow`), pues unos cuantos problemas requieren enteros de mayor longitud para poder realizar sus operaciones. El acceso fuera de rango, el paso de estructuras de datos por valor y el uso indebido de las estructuras de datos son otros de los avisos frecuentes.