6

Análisis automático del código en prácticas de programación en orientación a objetos

Automatic analysis of the code in object-oriented programming practices

Pedro Delgado-Pérez e Inmaculada Medina-Bulo
Departamento de Ingeniería Informática, Universidad de Cádiz, España

Resumen

La enseñanza y aprendizaje de habilidades de programación informática son procesos complejos para el profesor y para el alumno respectivamente. Como tal, haciendo uso de avances tecnológicos, en los últimos años han surgido diversas propuestas para apoyar al alumno en esta tarea, así como para ayudar al profesor en la evaluación de asignaturas en las que se imparten estas habilidades. Entre estas propuestas es popular la ejecución de conjuntos de casos de prueba, lo cual permite comprobar que el programa que escribe el alumno en las prácticas satisface la funcionalidad requerida. Sin embargo, este enfoque no permite detectar si el alumno cumple ciertos requisitos que estaban establecidos. Este capítulo se centra en el análisis directo del código para la comprobación del cumplimiento de los requisitos requeridos en los enunciados de las prácticas. Este análisis, conocido como estático, es automático y cuenta con varios beneficios tanto para el alumno como para el profesor, destacando que se acelera la retroalimentación que obtiene el estudiante en todo momento. El objetivo del capítulo es describir la experiencia en una asignatura de programación orientada a objetos durante dos cursos aplicando dichas comprobaciones sobre el código, mostrando en qué consiste y cómo se ha desarrollado tal innovación. Además, se muestran resultados orientativos de su aceptación por parte de los alumnos y su utilidad.

Palabras clave: Prácticas de programación, corrección automática, análisis estático, C++, paradigma de orientación a objetos.

Cita sugerida:
Delgado-Pérez, P., y Medina-Bulo, I. (2017). Análisis automático del código en prácticas de programación en orientación a objetos. En S. Pérez-Aldeguer, G. Castellano-Pérez, y A. Pina-Calafi (Coords.), *Propuestas de Innovación Educativa en la Sociedad de la Información* (pp. 61-71). Eindhoven, NL: Adaya Press. https://doi.org/10.58909/ad17102678

Abstract

Teaching and learning programming skills in computer science are complex processes for the lecturer and the student respectively. As such, making use of technological advances, several proposals have emerged in order to support both the student in this task and the professor in the evaluation of subjects where these skills are taught. Among these proposals, it is popular the execution of test suites, which allow checking that the program developed by the student satisfies the requested functionality. However, this option cannot help detect whether that program complies with certain established requirements or not. This chapter focuses on the direct analysis of the code to verify that the solutions provided by the students meet the conditions set in the practice statement. This analysis, known as static analysis, is automatic and beneficial both for the professor and the student, highlighting the quick *feedback* obtained by the student at all times. The aim of this chapter is to describe the experience in a subject about object-oriented programming in a two-year period applying the aforementioned checks on the code. We explain what the innovation is used for and how it was developed. In addition, we show illustrative results related to its acceptance by the student body and its utility.

Keywords: Programming practices, automatic corrections, static analysis, C++, object-oriented paradigm.

Introducción

Varias asignaturas en diversos grados universitarios inician a los alumnos en el uso de lenguajes de programación. Las prácticas de estas asignaturas son muy importantes para la adquisición de los conocimientos. En ellas, es usual proporcionar enunciados de forma que el alumnado ha de crear programas que implementen las funcionalidades solicitadas. En estos enunciados, no solo es importante la funcionalidad del programa desarrollado, sino que además se suelen incluir determinados objetivos a cumplir a fin de que el alumno se familiarice y ejercite los conocimientos que se van abordando en la teoría. Sin embargo, para que el alumno sepa si estos objetivos se han llevado a cabo correctamente, se suele necesitar de la intervención del profesor, lo cual dificulta el proceso de aprendizaje, especialmente en grupos con muchos alumnos. Este hecho ha motivado que, en los últimos años, haya surgido una tendencia hacia la búsqueda de mecanismos que guíen al alumno durante las prácticas y/o ayuden al profesor a corregir y evaluar el código de los alumnos (Higgins et al., 2005; Moltó et al., 2009; Mosqueira-Rey, 2010).

Una de las técnicas que más se emplea para tal fin es el desarrollo de conjuntos de casos de prueba, tal y como realizan diversas plataformas desarrolladas por otros autores para la enseñanza de prácticas de programación (Surrell et al., 2011). Así se comprueba si la ejecución de los casos de prueba sobre el programa creado por el alumno devuelve las salidas esperadas. Sin embargo, el uso de casos de prueba no permite asegurar que

el alumno ha realizado la implementación de la práctica siguiendo las pautas indicadas, tal y como se explica en un artículo anterior (Delgado-Pérez y Medina-Bulo, 2015). Este problema sí se solventa con el análisis de ciertas comprobaciones directamente sobre el código (análisis estático), lo cual ha sido puesto en práctica por varios autores en el pasado (Rodríguez et al. 2007; Romero et al., 2010).

En este capítulo (el cual es una extensión del trabajo de Delgado-Pérez y Medina-Bulo, 2016) se quiere dar a conocer la aplicación de la corrección automática de código y los resultados de nuestra experiencia en su aplicación en la asignatura de Programación Orientada a Objetos del Grado de Ingeniería Informática de la Universidad de Cádiz (España). La innovación, que se aplica desde el curso 2014-2015, consiste en ofrecer al alumno un programa que, al ejecutarlo, le permite conocer al instante los problemas que presenta la solución que ha programado a través de mensajes informativos. Estos mensajes podrán ser utilizados entonces por el alumno para corregir la práctica. El profesor es quien desarrolla previamente un programa por cada práctica en base al enunciado de la misma. De esta manera, el programa contendrá comprobaciones de acuerdo a los requisitos que se espera que la solución de la práctica deba cumplir (sobre todo, respecto a la orientación a objetos). El análisis estático del código que subyace en estos programas se logra a través de las bibliotecas proporcionadas por Clang (n.d.), un compilador para C++, que es el lenguaje de programación que se emplea en la asignatura. A fin de ofrecer indicios de la utilidad de la innovación, se comentan los resultados de encuestas pasadas a los alumnos, así como los resultados a nivel académico en estos últimos cursos.

La estructura del capítulo es la que sigue. La Sección 2 se centra en describir el cometido de la corrección automática del código, qué lo diferencia de la ejecución de casos de prueba y cuáles son los beneficios y limitaciones en su aplicación a la corrección de prácticas. La Sección 3 explica el enfoque para la aplicación del análisis estático en las prácticas de programación (ilustrado con un ejemplo) y muestra los resultados observados a partir de la implantación de esta innovación. Finalmente, se comentarán las conclusiones extraídas.

Corrección automática de prácticas

Descripción

En esta sección se describe en qué se basa el enfoque que aplicamos para automatizar la corrección de prácticas de programación. La innovación que se presenta busca realizar comprobaciones sobre el código de los alumnos sin necesidad de ejecutar sus programas. Esta técnica se conoce como análisis estático de software. Al contrario que el proceso de revisión de código que puede realizar manualmente una o un grupo de

personas, este análisis estático se acomete de forma totalmente automática. En el ámbito docente que nos concierne, la revisión de código en asignaturas de enseñanza de habilidades de programación la efectúa uno o varios profesores, normalmente tras la entrega de las soluciones de los alumnos en fechas establecidas para la finalización de cada práctica. Al automatizar este proceso que habitualmente se realiza de manera totalmente manual, se consigue reducir el tiempo y esfuerzo dedicados a la corrección de las prácticas.

Ejemplo

Pongamos el supuesto de que el alumno tiene que plasmar a través de una jerarquía de clases una cierta taxonomía relacionada con animales, en concreto, una como la que se muestra en la Figura 1. Desde objetos de cada una de las clases (excepto “Animal”), debe poder llamarse a un método “familia”, que devuelve la familia a la que pertenece cada animal particular. Es por ello que, tanto “Cánido” como “Félido” tendrán que proporcionar una definición para este método para que devuelva el mensaje “cánido” o “félido” respectivamente. Sin embargo, las clases “Perro”, “Lobo” y “Tigre” no tienen que sobrescribir ese método, ya que lo heredan de la clase superior.

Dicho esto, supongamos por ejemplo que el alumno replicase el método “familia” de “Cánido” en la clase “Perro” (representado en rojo en la Figura 1). Si tuviésemos un caso de prueba que llamara al método “familia” desde un objeto de la clase “Perro” y comprobara que lo que devuelve el método es “cánido”, el caso de prueba pasaría con éxito, pues efectivamente la llamada ha cumplido con lo esperado. No obstante, el caso de prueba soslaya el hecho de que el alumno ha incluido código innecesario.

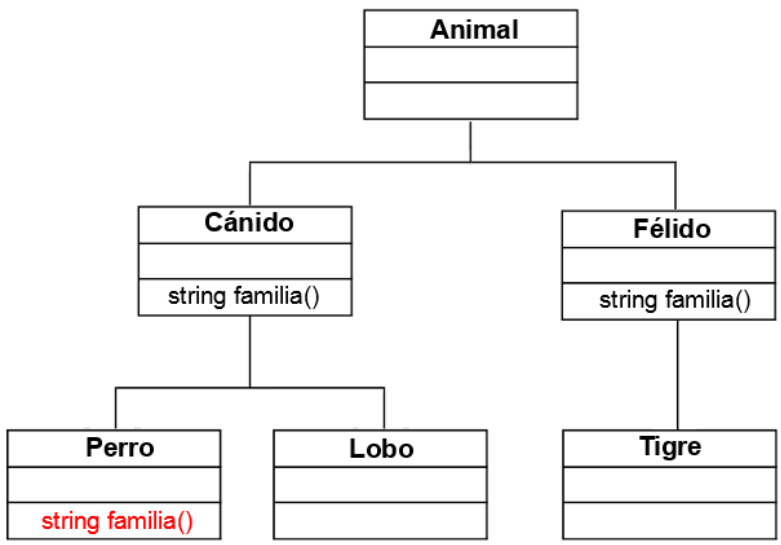

Figura 1. Jerarquía de clases a implementar en una práctica

Aquí se observa la necesidad de realizar un análisis directo sobre el código. Con este análisis podríamos recorrer el código y tratar de localizar la existencia de un método "familia" en la clase "Perro". Si así fuera, se podría informar al alumno del error que está cometiendo. Este análisis, complementado con la ejecución del caso de prueba, permitiría verificar no solo que el programa funciona, sino que está construido adecuadamente.

Ventajas y limitaciones

La aplicación del análisis estático de código tiene varias ventajas, aunque también se presentan algunas dificultades en su puesta en práctica. El mayor beneficio de su aplicación radica en que la adquisición de información sobre los errores cometidos por parte de los alumnos deja de depender de forma directa de la revisión por parte del profesor, permitiendo que el aprendizaje sea más fluido y adaptado al ritmo del alumno. De esta manera, se pueden mencionar tres ventajas principales:

- El alumno puede recibir retroalimentación de los errores cometidos en el mismo momento, disponiendo de una ayuda incluso si trabaja fuera de las horas de laboratorio de prácticas.
- Se produce una mejora en la interacción profesor-alumno. El alumno, en lugar de realizar preguntas muy generales y sin mucho trasfondo al profesor, puede realizar preguntas más razonadas y centradas en los errores cometidos al disponer de unos indicios previos sobre los problemas con los que se encuentra.
- Gracias a lo anterior, el profesor está más liberado de la corrección (la cual además será menos propensa a errores al estar automatizada) y puede dedicar mayor tiempo a otras tareas, lo cual supone una ventaja sobre todo en clases en las que hay un número elevado de alumnos.

Como se ha comentado anteriormente, también el análisis estático presenta algunas limitaciones:

- El profesor debe dedicar un tiempo inicial a la implementación de las comprobaciones para que estén disponibles para las prácticas.
- En ocasiones, y por la propia complejidad de llevar a cabo un análisis del código, no es fácil lograr una implementación robusta de las diferentes comprobaciones.
- Por último, existe una limitación inherente en el análisis estático a la hora de realizar comprobaciones específicas. Cuando la comprobación se centra en un elemento concreto del código, como puede ser un método de una clase, se hace necesario conocer el nombre de ese elemento para poder buscarlo en el código. Siguiendo con el mismo ejemplo, conocer el nombre del método nos permite delimitar el método sobre el que realizar la comprobación, evitando aplicarla al resto de métodos también. Esta situación obliga en ocasiones a especificar en el enunciado qué nombres hay que usar en la práctica para ciertos elementos.

Experiencia usando análisis estático en programación orientada a objetos

Desarrollo y aplicación de programas de análisis estático

En este apartado se explica cómo se ha empleado análisis estático en la asignatura de Programación Orientada a Objetos del Grado de Ingeniería Informática de la Universidad de Cádiz para realizar comprobaciones sobre el código de los alumnos en las prácticas de la asignatura. Esta asignatura se imparte en el segundo semestre del segundo curso, siendo obligatoria y con una carga de 6 créditos ECTS. El porcentaje de dedicación a la parte práctica es del 60%, por un 40% de la parte teórica. En los cursos en los que se aplicó la innovación, se proponía la realización de cinco prácticas en las que el alumno debía implementar diversas clases para ejercitar los conceptos de la orientación a objetos.

En una primera fase de desarrollo previa, por cada una de las prácticas de esta asignatura se estudió su enunciado para crear comprobaciones dedicadas a validar los requisitos de la práctica a través del análisis estático. Estas comprobaciones fueron recogidas en un programa por cada práctica particular, al cual llamamos de ahora en adelante como programa-solución. El programa-solución efectúa las comprobaciones oportunas sobre el código del alumno al ser ejecutado sobre el mismo. En esa ejecución, recopila aquellas comprobaciones que se cumplen y aquellas que no. Al finalizar, utiliza esa información para mostrar al alumno los mensajes informativos asociados a los errores cometidos a fin de que este pueda emplearlos para corregir la práctica y aprender de sus errores. Esos mensajes son establecidos de antemano por el profesor, que es quien decide el nivel de detalle que se ofrece sobre el error, sobre todo en base al conocimiento del alumno o dependiendo de la fase del curso en el que se encuentre (en las prácticas iniciales podría darse más información que en las finales). De esta forma, se consigue que cada programa-solución esté personalizado por completo a la práctica.

El análisis estático se lleva a cabo mediante el uso de las bibliotecas de Clang (n.d.), que nos permiten realizar la validación de las comprobaciones deseadas. Este compilador nos facilita las mismas bibliotecas que se utilizan para compilar el código pero a fin de que desarrollemos nuevas herramientas de análisis. En este capítulo se omite el modo de implementación mediante la reutilización de las bibliotecas del compilador al contener detalles de muy bajo nivel.

Una vez implementados los programas-solución, estos programas se les proporcionaban a los alumnos para que pudieran apoyarse en ellos a la hora de realizar las prácticas. Esto es así porque les permite tener retroalimentación directa de los errores que les lleven a emprender acciones correctivas en el mismo momento. Estos programas-solución vienen a sumarse al conjunto de casos de prueba que el código de la práctica ha de superar, de forma que cada solución debe pasar por el filtro del análisis estático y dinámico.

Ejemplo de programa-solución

En la Figura 2 se muestra un ejemplo de un fragmento de programa-solución para la primera práctica, en la cual se pide implementar una clase para trabajar con fechas. En esta imagen, en primer lugar se ejecutan todas las comprobaciones (primera línea) sobre el código que suministra el alumno, e internamente se almacena si las comprobaciones se han cumplido o no. Después, ya de forma individualizada, se van procesando cada una de las comprobaciones para mostrar o no el mensaje asociado:

```
Tool.run(newFrontendActionFactory(&Finder).get());

if(!FC.valida_constantes()){
        sin_fallos = false;
        llvm::outs() << "Revisa el enunciado respecto a las constantes AnnoMinimo y AnnoMaximo.\n";
}

if(!FC.tiene_default_constructor()){
        sin_fallos = false;
        llvm::outs() << "Revisa el enunciado respecto a la construcción de objetos.\n";
}

if(sin_fallos && !FC.tiene_initialization_list()){
        sin_fallos = false;
        llvm::outs() << "Revisa la inicialización de los atributos.\n";
}
```

Figura 2. Ejemplo de un fragmento con tres comprobaciones en un programa solución

- Es un requisito de la práctica emplear las constantes *AnnoMinimo* y *AnnoMaximo*. Sin embargo, cabe la posibilidad de no definirlas y aun así la funcionalidad del programa no cambiaría, por ejemplo si se inserta directamente el valor de esas constantes en los puntos en los que se deberían referenciar esas constantes. En un caso como ese, se mostraría el mensaje “Revisa el enunciado respecto a las constantes AnnoMinimo y AnnoMaximo” en pantalla.
- La segunda condición comprueba que el alumno provee una definición para el constructor por defecto, tal y como indica la práctica.
- En tercer lugar, la comprobación “tiene_initialization_list” se encarga de analizar si en el constructor por defecto se han inicializado los atributos de la clase mediante listas de inicialización, una de las características que se pretende ejercitar en esta práctica. Hay que hacer notar que el mensaje de esta comprobación no se mostrará en caso de fallar si la comprobación anterior falla también (se evita mostrar el mensaje asociado al uso de listas de inicialización si el alumno ni siquiera ha proporcionado el constructor).

En la Figura 3 se muestra un ejemplo de ejecución del programa-solución de la Figura 2. En esta imagen se ilustra tanto la ejecución sobre un código de un alumno que cumple todos los requisitos (imagen superior), como uno en el que no se ha encontrado la existencia de un constructor por defecto (imagen inferior). En ese último caso, el alumno procesará el mensaje, tratará de buscar el origen del error del que se le informa y deberá revisar su código hasta obtener una versión correcta que se ajuste al cometido de la práctica.

```
./fecha_check fecha.cpp -- -std=c++11

Ejecución del programa de comprobaciones para la clase Fecha
************************************************************

Verificación correcta de la clase Fecha.
```

```
./fecha_check fecha.cpp -- -std=c++11

Ejecución del programa de comprobaciones para la clase Fecha
************************************************************

Revisa el enunciado respecto a la construcción de objetos.
```

Figura 3. Arriba: Ejecución correcta de todas las comprobaciones. Abajo: Ejemplo de error de la segunda condición de la Figura 2

Resultados experiencia con programas-solución

Los programas-solución, una vez implementados, pueden tener un uso diverso, principalmente:

- Ser entregados a los alumnos para su uso particular a la hora de realizar las prácticas.
- Ser empleados por el profesor a modo de apoyo para la evaluación.

En nuestro caso, decidimos optar por la primera ya que resulta muy interesante que los alumnos cuenten con el apoyo directo de una herramienta que les ayuda a desarrollar las prácticas y a focalizar su atención en los detalles de la implementación.

De forma general, la experiencia ha resultado muy positiva en los dos años de aplicación. Esto es así sobre todo por parte de los alumnos, quienes ahora cuentan con mecanismos para que la realización de las prácticas no esté sujeta por completo a las horas de laboratorio (en las que se puede consultar con el profesor), sino que pueden disponer de esta ayuda para su trabajo en casa o con los compañeros.

Al final del curso se entregó una encuesta de satisfacción opcional a los alumnos con diferentes preguntas relacionadas con la innovación que aquí se presenta. La nota general que otorgan los alumnos a la innovación es de un 3.98 sobre 5 puntos, por lo que la misma parece haber tenido una buena acogida. Las preguntas que recibieron una mayor puntuación son la relativa a la valoración sobre el empleo conjunto del análisis estático y dinámico y la relativa al incremento de la importancia que le daban a la prueba de software tras la innovación. Aparte de ello, el uso de estos programas:

- Ayuda a que los alumnos no vuelvan a cometer los mismos errores. Esto es debido a que las medidas correctivas las llevan a cabo de manera instantánea, y no tiempo después cuando seguramente ya no recuerdan el motivo por el que se produjo el error.
- Fomenta que los alumnos se centren en los detalles de implementación desde el comienzo y, de forma general, sean más cuidados y traten de perfeccionar la práctica ya que disponen de un modelo correcto al que deben ajustarse.

En cuanto a las notas, por lo general no se puede asociar la tasa de éxito (alumnos aprobados / alumnos presentados) y la tasa de rendimiento (alumnos aprobados / alumnos matriculados) con el uso de estos programas. Esto es así porque:

- Existen multitud de aspectos que pueden variar el rendimiento académico de un curso para otro: número de alumnos, alumnos repetidores, asistencia a clase, dificultad de los exámenes...
- Las prácticas no tienen una influencia directa en la calificación obtenida.

No obstante, tal y como se muestra en la Tabla 1, se puede observar un aumento en la tasa de éxito y la tasa de rendimiento en los dos últimos cursos (en los que se usó la innovación) con respecto a la media en los años anteriores desde la implantación del Grado de Ingeniería Informática en el curso 2011-12.

Tabla 1. Comparación de tasa de éxito y tasa de rendimiento en los cursos 2014-15 y 2015-16 (con innovación) y del promedio de los cursos previos a la innovación

	Promedio cursos previos	2014-15	2015-16
Tasa de éxito	52.6%	57.1%	58.4%
Tasa de rendimiento	31.6%	32.0%	33.9%

Conclusión y trabajo futuro

Tanto el aprendizaje (labor del alumno) como la detección y corrección de errores (labor del docente) se tornan en labores complejas cuando hablamos de la enseñanza de lenguajes de programación. La innovación que se presenta viene a solventar en cierta medida las dificultades que se encuentran ambas partes. Por el lado del alumno, el uso de los programas-solución desarrollados por el profesor le permite conocer de forma rápida (sin necesitar de la ayuda directa del profesor) sobre sus errores y razonar sobre las posibles implementaciones e identificar casos correctos y erróneos. Del lado del profesor, este puede delegar en parte la revisión que debía realizar visualmente para conseguir que cada uno de los alumnos llegase a conocer si su práctica era o no correcta. Esta

situación parece reflejarse en la encuesta de satisfacción y los resultados académicos obtenidos hasta el momento.

De cara al futuro, en primer lugar sería bueno poder extrapolar la idea a otras asignaturas e incluso a otros lenguajes y dominios. Es destacable que los alumnos valoraran esta posibilidad positivamente en la encuesta, ya que disponer de un mecanismo parecido a este podría ayudarles en asignaturas de corte similar.

En segundo lugar, se está trabajando actualmente en la creación de una biblioteca que recoja las comprobaciones que se han ido implementando en los diferentes programas-solución, de manera que se facilite su reutilización en nuevas prácticas en el futuro. Esta biblioteca ayudará a que los cambios en una misma comprobación se vean reflejados en todos los programas-solución que emplean esa comprobación.

Referencias

Clang (n.d.). Recuperado de: http://clang.llvm.org

Delgado-Pérez, P. y Medina-Bulo, I. (2015). Automatización de la corrección de prácticas de programación a través del compilador Clang. *Actas de las XXI Jornadas de Enseñanza Universitaria de Informática (JENUI)*, pp. 311-318.

Delgado-Pérez, P., y Medina-Bulo, I. (2016). Experiencia en la corrección automática de prácticas de programación en orientación a objetos. En *EDUNOVATIC, Actas del I Congreso Virtual Internacional de Educación, Innovación y TIC* (pp. 40- 47). Madrid: REDINE. Recuperado de www.edunovatic.org/actas-2016/

Higgins C., Gray G., Symeonidis P. y Tsintsifas A. (2005). Automated assessement and experiences of teaching programming. *ACM Journal of Educational Resources in Computing, 5*(3), 1-21.

Moltó, G., Galiano M., Herrero C., Prieto N. y Sapena, O. (2009) Uso de herramientas TIC para la mejora de la interacción profesor-alumno, la evaluación continua y el aprendizaje autónomo. En las *Jornadas de Innovación UPV 2009: Metodologías Activas para la Formación de Competencias.*

Mosqueira-Rey, E. (2010). La evaluación continua y la autoevaluación en el marco de la enseñanza de la programación orientada a objetos. *Actas de las XVI Jornadas de Enseñanza Universitaria de Informática (JENUI)*, pp. 223-230.

Rodríguez del Pino, J.C., Díaz Roca, M., Hernández Figueroa, Z., y González Domínguez, J. D. (2007). Hacia la evaluación continua automática de prácticas de programación. *Actas de las XIII Jornadas de Enseñanza Universitaria de Informática (JENUI)*, pp. 179-186.

Romero, F., Serrano-Guerrero, J. y Pérez de Inestrosa, H. (2010). CUESTOR: Una nueva aproximación integral a la evaluación automática de prácticas de programación. *Actas de las XXI Jornadas de Enseñanza Universitaria de Informática (JENUI)*, pp. 493- 500.

Surrell, J., Boada I., Soler, J., Prados, F. y Poch, J. (2011). Corrección Automática de Ejercicios de Estructuras de Datos a través de una Plataforma de E-learning. *Actas de las XVII Jornadas de Enseñanza Universitaria de Informática (JENUI)*, pp. 75-82.

Pedro Delgado Pérez trabaja como profesor en el Departamento de Ingeniería Informática de la Universidad de Cádiz, en donde también desarrolla su tesis doctoral dentro del grupo UCASE de Ingeniería del Software. En el lado docente, imparte clases desde el año 2013 en asignaturas del Grado de Ingeniería Informática, como Verificación y Validación de Software, Diseño de Algoritmos e Inteligencia Artificial. En el plano investigador, su línea de investigación se centra en la prueba de software, principalmente sobre sistemas orientados a objetos, aplicando prueba de mutaciones y técnicas de búsqueda para reducir su coste.

Inmaculada Medina Bulo es profesora del Departamento de Ingeniería Informática en la Escuela Superior de Ingeniería (ESI) de la Universidad de Cádiz (España) desde 1995. Es la investigadora principal del grupo de investigación UCASE de Ingeniería del Software y de la Red de Excelencia en Ingeniería de Software Basada en Búsqueda, así como de distintos proyectos de investigación. Sus líneas principales de investigación son la aplicación de técnicas de búsquedas a problemas de Ingeniería del Software, procesamiento de eventos complejos, servicios web, y verificación y validación del software, en particular, la prueba de software.

ISBN 978-94-92805-00-3