Experiencia en la corrección automática de prácticas de programación en orientación a objetos

Pedro Delgado-Pérez
Departamento de Ingeniería Informática,
Universidad de Cádiz, España.

Inmaculada Medina-Bulo
Departamento de Ingeniería Informática,
Universidad de Cádiz, España.

Resumen

El aprendizaje de habilidades de programación informática es un proceso complejo y es por ello que se han realizado diferentes propuestas para ayudar al alumno y potenciar la enseñanza haciendo uso de los avances tecnológicos. Entre ellos se encuentra la corrección automática de prácticas, lo cual principalmente acelera la retroalimentación que obtiene el estudiante. El enfoque que se presenta en este documento se basa en el análisis directo del código para la comprobación del cumplimiento de los requisitos requeridos en los enunciados de las prácticas, contando con varios beneficios tanto para el alumno como para el profesor. El objetivo de este artículo 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. En este artículo también se muestran resultados orientativos de su aceptación por parte de los alumnos y su utilidad.

Palabras clave: *prácticas de programación; corrección automática; C++; paradigma de orientación a objetos.*

1. Introducción

Las prácticas de programación informática suelen incluir determinados objetivos a cumplir por parte del alumno a fin de que ejercite los conocimientos adquiridos en la teoría. Sin embargo, para que el alumno sepa si estos objetivos se han llevado a cabo correctamente, se suele necesitar de la revisión del profesor, lo cual dificulta el proceso de aprendizaje. Este hecho ha propiciado en los últimos años una tendencia hacia la búsqueda de mecanismos que guíen al alumno durante las prácticas y a la corrección automática del código (Higgins et al., 2005; Moltó et al., 2009).

Nuestro enfoque va más allá de comprobar automáticamente que la ejecución de casos de prueba sobre el programa devuelve las salidas esperadas, como en el caso de plataformas desarrolladas por otros autores para la enseñanza de prácticas de programación (Surrell et al., 2011). Nuestro objetivo busca realizar ciertas comprobaciones directamente sobre el código, de forma que se pueda validar que el alumno ha realizado la implementación de la práctica siguiendo las pautas indicadas, tal y como fue sugerido en un artículo anterior (Delgado-Pérez y Medina-Bulo, 2015) y como otros autores han realizado en el pasado (Rodríguez et al. 2007; Romero et al., 2010). Con este trabajo se quiere dar a conocer la aplicación del enfoque mencionado y los resultados en el desarrollo de esta experiencia.

En concreto, esta innovación se ha aplicado durante los cursos 2014-15 y 2015-16 en la asignatura de Programación Orientada a Objetos del Grado de Ingeniería Informática de la Universidad de Cádiz (España), en la cual las prácticas se implementan usando el lenguaje de orientación a objetos C++. A partir de las bibliotecas de Clang (n.d.), un compilador maduro para este lenguaje, se han desarrollado programas de comprobación acorde a los enunciados de cada una de las prácticas. Estos programas están disponibles para el alumno a la hora de implementar el código, guiándole a través de mensajes informativos de los errores que van cometiendo en su elaboración. Los resultados de una encuesta a los alumnos así como los resultados a nivel académico revelan la utilidad de los esfuerzos puestos en la ayuda al alumno para su aprendizaje.

En este artículo se explicará en la Sección 2 en qué consiste la corrección automática y cuáles son los beneficios y limitaciones detectadas en este campo de aplicación. En la Sección 3 describiremos cómo se ha llevado a cabo la experiencia y en la Sección 4 se expondrán los resultados observados a partir de la implantación de esta innovación. Finalmente, se comentarán las conclusiones extraídas.

ISBN: 978-84-617-7628-3

2. Corrección automática

Como se ha adelantado en la introducción, la innovación que se presenta en este texto se basa en la realización de *comprobaciones* sobre el código, lo cual se conoce como *análisis estático*. Este análisis de software se realiza sin ejecutar el programa ya que el análisis se produce sobre el propio código. Al contrario que el proceso de *revisión de código* realizado por una o un grupo de personas, este análisis estático se ejecuta de forma automática. En el contexto que aquí se trata, la revisión de código en asignaturas de enseñanza de habilidades de programación es acometida por uno o varios profesores, normalmente tras la entrega de las soluciones de los alumnos en fechas determinadas. La automatización de este proceso manual permite pues acortar los tiempos y reducir los esfuerzos de corrección. Este hecho evita que la adquisición de información sobre los errores cometidos por parte de los alumnos dependa de la revisión por parte del profesorado, y en esto radica el beneficio que ofrece este enfoque de corrección automática. De esta manera, se pueden mencionar tres ventajas principales:

- El profesor puede dedicar menos tiempo a la corrección ya que parte de la misma se puede realizar de forma automática. Además, la corrección será menos propensa a errores al no estar sujeta a un factor humano.
- El alumno puede recibir retroalimentación de sus errores de una manera más rápida.
- El profesor, más liberado de sus obligaciones de corrección en clases de un buen número de alumnos, y el alumno, conocedor de los errores que surgen en en el momento de la elaboración de la práctica, encuentran en el análisis estático un mecanismo que mejora la interacción profesor-alumno.

También el análisis estático cuenta con algunas limitaciones:

- Tiempo de dedicación inicial a la implementación de las comprobaciones por parte del profesor.
- Dificultad de implementación de las comprobaciones por la propia complejidad del análisis del código.
- Por lo general, no todas las comprobaciones se pueden llevar a cabo mediante este mecanismo. En ocasiones, es necesario precisar en el enunciado el uso de nombres específicos para ciertas declaraciones para poder validar comprobaciones concretas.

ISBN: 978-84-617-7628-3

3. Experiencia

En este apartado se explica cómo las comprobaciones sobre el código se han empleado en la asignatura de Programación Orientada a Objetos del Grado de Ingeniería Informática de la Universidad de Cádiz.

Cada una de las comprobaciones dedicadas a validar los requisitos de una práctica fueron recogidas en un programa, al cual conocemos como *programa-solución*. Estos programas realizan las comprobaciones e informan a través de mensajes cuando el código del alumno no cumple con las condiciones impuestas en la práctica. Estos mensajes pueden ser adaptados por el profesor en base al conocimiento de los alumnos, proporcionando más o menos información sobre el error concreto. En la Figura 1 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 las comprobaciones (primera línea) e internamente se almacena si las comprobaciones se han cumplido o no. Esto se lleva a cabo mediante el uso de las bibliotecas de Clang (n.d.), que nos permite realizar el análisis estático deseado (en este artículo se omite esta implementación mediante la reutilización de las bibliotecas del compilador al contener detalles de bajo nivel). Siguiendo con el código de la Figura 1, después de forma individual se van validando cada una de las comprobaciones:

- Tanto si el alumno define las constantes *AnnoMinimo* y *AnnoMaximo* como si no, el programa puede tener la misma funcionalidad simplemente empleando los valores de esas constantes en los puntos del código en los que se emplean. Como este es un requisito de la práctica, si no se han definido estas constantes, se muestra el mensaje “Revisa el enunciado respecto a las constantes AnnoMinimo y AnnoMaximo” en pantalla.
- La segunda condición comprueba que se definió un constructor por defecto, tal y como indicaba la práctica.

 ISBN: 978-84-617-7628-3

```
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 1. Ejemplo de un fragmento con tres comprobaciones en un programa solución.

- En tercer lugar, se comprueba que en el constructor por defecto se han inicializado los atributos de la clase utilizando listas de inicialización, una de las características que se pretendía ejercitar en esta práctica. Obsérvese que esta comprobación está sujeta a que se haya cumplido la condición anterior: si no se ha detectado un constructor por defecto no tiene sentido que se muestre un mensaje sobre la lista de inicialización en este constructor.

Con el fin de que el alumno tuviese información de primera mano y contara con una herramienta que le ayudase a adquirir los conocimientos de la mejor manera posible, estos programas-solución eran entregados a los alumnos al inicio de cada práctica. Al mismo tiempo también se les proporcionaba un conjunto de casos de prueba que el código de la práctica ha de superar, de forma que se pueda realizar un análisis tanto estático como dinámico.

En la Figura 2 se muestra un ejemplo de ejecución del programa-solución de la Figura 1. En este imagen se muestra un ejemplo tanto de una ejecución en la que el código del alumno cumple todos los requisitos, como uno en el que no se ha encontrado la existencia de un constructor por defecto. En ese caso, el alumno tendrá que revisar su código hasta obtener un código correcto acorde al enunciado.

ISBN: 978-84-617-7628-3

```
./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 2. Arriba: Ejecución correcta de todas las comprobaciones.
Abajo: Ejemplo de error de la segunda condición de la Figura 1.

En el vídeo que se puede observar en el siguiente enlace (https://www.dropbox.com/s/342knpgcr93gpc2/P1010117.MOV), se realiza una simulación de cómo se emplearía un programa-solución por parte del alumno. El alumno ejecuta el programa de comprobaciones con el fichero que contiene su código ("fecha.cpp"). El mensaje de error que muestra la ejecución lleva al alumno a la revisión del código y a realizar las medidas correctivas oportunas. Finalmente, cuando el código cumpla todas las comprobaciones, se mostrará el mensaje de que todo es correcto.

3. Resultados

De forma general, la experiencia ha resultado muy positiva, especialmente para los alumnos, que cuentan con el apoyo de una herramienta que les ayuda a desarrollar las prácticas con una mayor reflexión sobre los detalles de la implementación. Esta herramienta además no se limita a la hora de prácticas con el profesor, sino que puede ser utilizada en cualquier momento, lo cual es un beneficio añadido sobre todo cuando en clase se cuenta con un gran número de alumnos y el profesor no siempre está disponible. Todo esto se ha visto reflejado en una encuesta de satisfacción proporcionada a los alumnos. En términos generales, los alumnos valoran la posibilidad de realizar estas comprobaciones a su código con un 3.98 sobre 5 puntos. Según esta encuesta, también los alumnos valoran significativamente el uso conjunto de los programas de comprobaciones y la batería de casos de prueba, y esto también deriva en una mayor consciencia de la importancia de la prueba de software.

ISBN: 978-84-617-7628-3

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, ya que las prácticas, una vez aprobadas, no tienen influencia en la calificación numérica final y hay muchos otros factores que pueden afectar a estas medidas. Sin embargo, 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 usando los programas de comprobaciones 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%

4. Conclusiones

Los programas de comprobaciones se han mostrado como una herramienta muy interesante como complemento a la ejecución de casos de pruebas. Los alumnos agradecen que estos programas puedan ser empleados por ellos en lugar de usarse como sistema de puntuación, ya que los programas les permiten razonar sobre diferentes implementaciones, identificando situaciones correctas y erróneas. Este sistema les posibilita localizar fallos existentes en su código en el momento efectivo para el aprendizaje: cuando se realiza la implementación y no más adelante tras la revisión del profesor. En el futuro, sería bueno trabajar en la extensión de los ficheros de comprobaciones para otros lenguajes, otras asignaturas y dominios. De hecho, una de las preguntas de la encuesta con mayor puntuación así como algunos de los mensajes ofrecidos por los alumnos en la encuesta indicaban que valorarían de forma positiva el disponer de un mecanismo similar en otras materias relacionadas con la programación.

ISBN: 978-84-617-7628-3

Referencias

Clang (n.d.). URL: 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)*, 311-318.

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

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)*, 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)*, 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)*, 75-82.

 ISBN: 978-84-617-7628-3