UNIVERSITAT POLITÈCNICA DE VALÈNCIA

IN-RED 2016
Congreso Nacional de Innovación Educativa y de Docencia en Red

Congreso In-Red 2016
UPV, 7 y 8 de julio de 2016
Doi: http://dx.doi.org/10.4995/INRED2016.2016.4368

Un entorno para el desarrollo de proyectos en la enseñanza activa de un curso de Compiladores

José Miguel Benedí[1] **y Emilio Vivancos**[1]

[1]Universitat Politècnica de València

Abstract

This paper describes an experience in a course on Programming languages and compilers. In this course we have chosen to an active learning methodology aimed at the implementation of a compiler project. In order to minimize the huge effort that teachers must invest in the preparation of a new project every year, a development environment of compile projects is proposed. The goal of this development environment is to greatly simplify the process of developing new projects.

Keywords*: Education, project-based learning, compilers.*

Resumen

En este trabajo se describe una experiencia que se lleva a cabo en la asignatura de "Lenguajes de Programación y Procesadores del Lenguaje". En esta asignatura hemos optado por una metodología activa orientada a la realización de un proyecto. Para minimizar el enorme esfuerzo que el profesorado debe invertir en la preparación de un nuevo proyecto cada año, en este trabajo se propone un entorno de desarrollo de proyectos de compilación que simplifica enormemente el proceso de elaboración de nuevos proyectos.

Palabras clave*: Educación, aprendizaje orientado a proyectos, Compiladores.*

1 Introducción

En la Universitat Politècnica de València, el curso de compiladores se imparte actualmente en la asignatura de *"Lenguajes de Programación y Procesadores del Lenguaje"* (LPyPL). El diseño y construcción de un compilador es un perfecto ejemplo de maridaje entre teoría y práctica (Aho 2008); de echo, constituye una de las primeras disciplinas de la informática para la cual se desarrolló un importante aparato formal que se utiliza actualmente de forma rutinaria en la construcción de compiladores. Hace unos años, la mayor parte de los cursos de compiladores ponían el énfasis en la teoría del análisis sintáctico y de la traducción dirigida por la sintaxis (fase de análisis). Sin embargo, pronto se tomó conciencia de que centrarse sólo en la teoría no enseña necesariamente mejor a los estudiantes el modo en que debe construirse un compilador útil (Waite 2006). Esta observación fue positivamente reforzada por nuestra experiencia en la impartición de las sucesivas asignaturas de compiladores en la Universitat Politècnica de València (Benedí y col. 2008). A lo largo de estos años, gradualmente hemos ido modificando nuestro punto de vista poniendo el énfasis en la parte de generación y optimización de código (fase de síntesis). Como resultado de todo ello, el objetivo de la actual asignatura de LPyPL es que los alumnos tengan la base teórica necesaria que les permita construir un compilador real para un lenguaje de programación sencillo, pero no trivial.

La constante evolución de los lenguajes de programación y de las arquitecturas de las máquinas destino hace imposible cubrir, en un curso de compiladores de un semestre, la diversidad de algoritmos y técnicas utilizados en un compilador comercial moderno. Por tanto se hace necesaria una estrategia viable que nos permita tomar decisiones coherentes sobre el enfoque y la profundidad con la que se deben impartir los contenidos.

En LPyPL hemos optado por una estrategia metodológica activa que permita la superación de una enseñanza por simple "transmisión/recepción de conocimientos ya elaborados". Esto sólo puede tener lugar si quien aprende participa, de alguna manera, en la (re)construcción de los conocimientos (Gil-Pérez y Carrascosa-Alis 1994); es decir, planteando el aprendizaje como tratamiento de problemas (realización de proyectos) y no como una transmisión de conocimientos ya elaborados que olvida los problemas que condujeron a su construcción. En este tipo de planteamientos, el aprendizaje resulta de la construcción de conocimientos a través del tratamiento colectivo de problemas significativos –en nuestro caso de un proyecto de compilación– de tal forma que los alumnos se enfrentan a situaciones problemáticas que han de solucionar, en un proceso colectivo de investigación dirigida a la consecución de un objetivo común. La labor fundamental del profesor ya no es solo la de transmisión de conocimientos, sino que también es la de preparación de actividades, que permitan a los alumnos participar en la (re)construcción de los conocimientos, y la de dirigir dichas actividades (Gil-Pérez y Carrascosa-Alis 1994).

Por todo ello, la metodología propuesta para LPyPL está orientada a la realización de un proyecto. En la aplicación de esta metodología se han encontrado varias ventajas:

- que es una forma adecuada de conseguir que el alumno adquiera una visión global mucho más real del funcionamiento de un compilador;
- que el alumno se siente más motivado al saber que terminará el curso habiendo participado en la elaboración de un compilador completo que funciona "realmente";
- que facilita la comprensión de algunos conceptos que difícilmente se entenderían solo en las sesiones teóricas;
- y finalmente, que esta alternativa es la más cercana al tipo de trabajos que el ingeniero en informática se va a encontrar en sus labores profesionales.

La puesta en marcha de un planteamiento de este tipo se enfrenta, sin duda, a grandes obstáculos. El primero de ellos consiste en una posible sobrecarga inaceptable de trabajo para los alumnos. Este planteamiento obliga a medir cuidadosamente el esfuerzo exigido a los alumnos y a estructurar adecuadamente el proyecto. En la Sección 2, mostraremos las acciones que en concreto se proponen para LPyPL.

El segundo problema está relacionado con el enorme esfuerzo y tiempo de dedicación que el profesorado debe invertir en el diseño y la preparación del proyecto de compiladores para cada curso académico. La propuesta presentada en este trabajo tiene como principal motivación la de mitigar este problema y ayudar al profesorado en la elaboración del proyecto de compiladores. Para darnos una idea de este trabajo, recordemos que un compilador acepta como entrada un programa escrito en un cierto *lenguaje fuente* (usualmente un lenguaje de alto nivel), y genera un programa escrito en un *lenguaje objeto* (usualmente un lenguaje máquina) (Aho y col. 2008).

La preparación, por parte del profesorado, de un nuevo proyecto de compilación lleva asociadas varias e importantes tareas. La primera de ellas está relacionada con la definición del lenguaje fuente. Esta es una tarea compleja ya que incorpora todos los problemas de diseño e implementación de un lenguaje de programación. El uso de un lenguaje fuente "real" no simplifica significativamente el problema dado que estos lenguajes son demasiado grandes y complejos para ser implementados en su totalidad por los estudiantes en un solo semestre. En la práctica se suele utilizar un subconjunto de un lenguaje de programación real, lo que termina siendo un problema de diseño en sí mismo.

Diseñar el lenguaje fuente del proyecto solo es la primera tarea que debe abordar el profesor. El segundo paso consiste en crear y suministrar el material necesario para la elaboración del proyecto, en concreto: definir las especificaciones formales (léxicas, sintácticas y semánticas) del proyecto; diseñar, implementar, probar y documentar todo el material de ayuda (librerías); y por último, elaborar toda la documentación necesaria. En la Sección 3, daremos una descripción más detallada de este material para la asignatura de LPyPL.

Finalmente, el profesorado debe implementar el proyecto completo antes del comienzo de la asignatura. Dado que solo una implementación preliminar completa nos permitirá garantizar que el proyecto es auto-consistente, completo y tratable por parte de los alumnos.

Cualquiera que tenga experiencia en impartir algún curso de compiladores reconocerá que la descripción idealizada de la preparación de un proyecto que se ha mencionado tiene poco parecido con la realidad. Las presiones de tiempo pueden llevarnos a tomar atajos o a imponer modificaciones en el diseño del proyecto, o que incluso el proyecto se diseñe en "tiempo real" mientras que el curso está en marcha. Por otro lado, una vez que un profesor ha creado un proyecto, la gran inversión realizada proporciona un fuerte incentivo para reutilizar el proyecto una y otra vez, incluso más allá del punto en que el proyecto se convierte en obsoleto.

Muchos profesores crean sus propios proyectos de compiladores desde cero, repitiendo mucho trabajo que se ha realizado otras muchas veces antes. La actual situación mejoraría si los profesores que diseñan proyectos de compiladores compartiesen los frutos de su trabajo.

En ese sentido, en esta ponencia se presenta un entorno de desarrollo, de libre disposición y portable, para facilitar el diseño y la elaboración de nuevos proyectos de compilación. Este entorno de desarrollo de proyectos ha sido utilizado en los 3 últimos cursos académicos de la asignatura de LPyPL en la Universitat Politècnica de València. Por ello consideramos que este entorno de desarrollo está lo suficientemente maduro como para que pueda ser útil a otros profesores de otras instituciones.

2 Contexto docente y modelo de implantación

La asignatura de LPyPL forma parte del plan de estudios en la nueva titulación de Grado, en Ingeniería Informática, de la Escuela Técnica Superior de Ingeniería Informática. Esta asignatura está enmarcada en el módulo de tecnología específica de *Computación* (4º curso, semestre A) y tiene asignados 6 créditos (4,5 teóricos y 1,5 prácticos). Su temario, como no podía ser de otro modo, es una aproximación al compromiso entre lo que se debería dar y lo que se puede dar, considerando el contexto académico actual, en un intento de ofrecer unas enseñanzas sobre compiladores ajustadas, razonables y actuales.

El objetivo principal de la asignatura de LPyPL es que el alumno conozca los fundamentos teóricos y prácticos, así como las técnicas y herramientas básicas, para el diseño y construcción de un compilador de un lenguaje imperativo. Al mismo tiempo se pretende que el alumno sea capaz de aplicar las ideas y técnicas propias del diseño de compiladores en otros campos de la informática. Tradicionalmente, en las asignaturas de compiladores, se ha hecho hincapié en la descripción formal de la fase de análisis de un compilador (análisis léxico, sintáctico y semántico, y traducción dirigida por la sintaxis). Las prácticas usualmente han consistido en la preparación de un conjunto de ejercicios/programas de laboratorio que refuercen los aspectos más significativos de la construcción de un compilador.

En esta asignatura hemos optado por desplazar el interés hacia la fase de síntesis, en general, y hacia la generación y optimización de código, en particular. Como ya hemos mencionado, en LPyPL hemos elegido una estrategia metodológica dirigida a la

realización de un proyecto, cuyo objetivo consiste en la construcción de un compilador completo para un lenguaje de programación seleccionado. De hecho, gran parte de la la teoría está orientada hacia el desarrollo de dicho proyecto de compilación.

Para la implantación de esta propuesta y para evitar, en lo posible, la sobrecarga de trabajo para los alumnos, que un proyecto de esta envergadura conlleva, se han tomado una serie de medidas correctoras:

1. Impulsar que el proyecto se realice en pequeños grupos –tres a cuatro alumnos como máximo. Además, con ello se consigue fomentar las habilidades del trabajo en equipo, requisito imprescindible en todo ingeniero informático.
2. Proporcionar a los alumnos un adecuado material de ayuda que les permita reducir significativamente el trabajo de codificación para centrarse en los problemas típicos de la construcción de compiladores. En la Sección 3, daremos una descripción más detallada de este material desde el punto de vista del trabajo del profesor.
3. Planificar un conjunto de seminarios, en grupos reducidos, para la descripción pormenorizada del material y herramientas específicas del proyecto.
4. Dividir el proyecto en partes, para facilitar su elaboración y evaluación, que se corresponden con las fases de análisis (léxico, sintáctico y semántico) y síntesis (generador de código intermedio) en la construcción de los compiladores.
5. Reforzar las tutorías en el laboratorio. La labor del tutor no solo debe ser la de resolver las dudas y problemas planteados sino también la de sugerir mejoras, detectar problemas, motivar hábitos de trabajo en equipo y enseñar a generar y documentar buenos programas.

La planificación del proyecto consiste en 10 sesiones de laboratorio de 90 minutos cada una. En 5 de estas sesiones se dedican los 20 minutos iniciales a impartir los seminarios específicos y el resto del tiempo se dedica al trabajo de los alumnos tutorizados por el profesor. Existen 3 puntos de control, sincronizados con las clases de teoría, para la autoevaluación del proyecto mediante programas de prueba proporcionados por los profesores: análisis léxico-sintáctico, análisis semántico y generación de código intermedio.

La evaluación del proyecto contempla tres aspectos: 1) *Actividades de seguimiento en el laboratorio*, donde se estima el grado de implicación del alumno en el desarrollo del proyecto. 2) *Evaluación del trabajo en el laboratorio*, donde se valora el trabajo continuo en el laboratorio mediante los correspondientes entregables asociados con cada una de los tres puntos de control del proyecto. Y 3) *evaluación individual del proyecto*, donde se determina el conocimiento del proyecto mediante un examen práctico individual en el laboratorio.

3 Entorno de desarrollo de proyectos de compiladores

El objetivo de la presentación de este entorno de desarrollo de proyectos de compiladores (EDPC) es la simplificación del proceso de creación de nuevos proyectos, para así favorecer la enseñanza de los fundamentos de la construcción de compiladores a alumnos de 4º curso del grado en Ingeniería Informática. El EDPC permite a los profesores una fácil modificación de los proyectos de compilación, favoreciendo igualmente la elaboración de proyectos de distinta naturaleza y amplitud. Una característica adicional importante del EDPC es que está pensado para diseñar proyectos modulares, incrementales y portables.

Hay un problema de índole práctico bastante común en el desarrollo del proyectos que es el de la reutilización de los mismos. Este problema es similar a la reutilización de los exámenes u otros entregables. Si un proyecto se ha utilizado una vez, los estudiantes pueden tener una tendencia a desarrollar una cierta "memoria" del proyecto que puede durar varios años. Además, los posibles estudiantes deshonestos podrían presentar el trabajo de cursos anteriores como propios. De hecho, solo este problema podría explicar porqué se inventan tantos nuevos proyectos de compiladores con la consiguiente multiplicación de esfuerzos.

Sin embargo, hacer cambios relativamente modestos para proyectos antiguos reduce sustancialmente el incentivo para hacer trampa. El EDPC está diseñado para que sea fácil de modificar y ampliar y, de hecho, se ha modificado sustancialmente con mucho menos esfuerzo de lo que sería necesario en la construcción del nuevo proyecto de LPyPL para este curso. El EDPC completo se compone de muchos aspectos que presentamos en las siguientes subsecciones.

3.1 Lenguaje fuente

Hay dos preguntas evidentes relacionadas con los lenguajes de programación involucrados en un proyecto de compilación: ¿Cuál debe ser el lenguaje de programación (fuente) del proyecto? Y ¿cuál debe ser el lenguaje de programación en el que los estudiantes deben escribir su compilador? Antes de abordar dichas cuestiones es necesario tratar un par de consideraciones previas relacionadas con la elección del lenguaje fuente: 1) Que el proyecto esté bien definido; es decir, que se conozca la especificación formal de las restricciones léxicas, sintácticas y semánticas del lenguaje fuente. Y 2) que el proyecto sea tratable; es decir, que sea abordable y que el lenguaje fuente sea lo suficientemente reducido como para garantizar que los alumnos puedan implementar su compilador en un semestre.

Dado que el lenguaje fuente debe ser reducido para que sea abordable, usualmente se plantean dos estrategias para afrontar dicha selección: diseñar un lenguaje inventado, o escoger un subconjunto reducido de un lenguaje de programación existente. Un lenguaje inventado permite centrarse en la facilidad de implementación del compilador y no en la facilidad de uso del lenguaje. Además, permite más libertad a la hora de escoger las características del lenguaje fuente dentro de un amplio espectro. La elección de un subconjunto reducido de un lenguaje existente tiene como limitación el

propio lenguaje seleccionado pero nos permite reducir la curva de aprendizaje y dota al proyecto de una coherencia que hace que, en nuestra opinión, sea didácticamente más atractivo.

En esta trabajo, nos decantamos por esta segunda opción; es decir, un (subconjunto de un) lenguaje de programación de alto nivel, sencillo pero no trivial. El lenguaje elegido, al que denominaremos `MenosC`, es un lenguaje basado en el lenguaje `C`, con algunas restricciones de tipos del lenguaje `C++`.

Finalmente, la elección del lenguaje de programación en la que los estudiantes escriben sus compiladores no es obvia y depende mucho del contexto del resto de asignaturas de la titulación. Teniendo en cuenta que las herramientas de generación de compiladores elegidas (`flex` y `bison`) generan programas en `C`, hemos seleccionado el lenguaje `C` como lenguaje de desarrollo[1].

3.2 Lenguaje objeto y máquina virtual

El EDPC nace con la voluntad de ser independiente de la plataforma, tanto desde el punto de vista de la facilidad de su instalación en diferentes máquinas con distintas arquitecturas, como desde el punto de vista del código objeto generado. Para potenciar la facilidad de instalación, el EDPC se ha diseñado para que resulte portable y fácil de instalar en cualquier máquina `Unix` con herramientas estándar de software GNU (`make`, `flex` y `bison`).

Respecto al código objeto generado, en el EDPC también se fomenta la portabilidad a través de la definición de un código intermedio independiente de la máquina, en vez de un código máquina (o código ensamblador) dependiente de una máquina (o una arquitectura) específica. La introducción de un código intermedio independiente de la máquina es un práctica común en la construcción de compiladores, ya que: mejora la fase de optimización de código (intermedio), aumenta la portabilidad y, en general, facilita la división en fases en el diseño de un compilador (Aho y col. 2008).

El siguiente paso es la elección del código intermedio. En primer lugar hay que exigirle que sea independiente de la máquina y que sea estructuralmente simple. En la literatura se han propuesto una gran variedad de códigos intermedios (Aho y col. 2008). *Códigos intermedios gráficos*: Árboles sintácticos de análisis, árboles sintácticos abstractos o grafos dirigidos acíclicos. *Códigos intermedios lineales*: Códigos máquina a pila (`bytecode` es un caso particular (Cooper y Torczon 2012)) o códigos 3-direcciones (Aho y col. 2008). La elección de un lenguaje intermedio real –`bytecode` o código para una máquina `MIPS32`– tienen la ventaja de ser generales y de que existen máquinas virtuales reales –`JVM` (Lindholm y col. 2011) o `SPIM` (Larus 2011)– que permiten ejecutar su código. Sin embargo, aprender un código intermedio real, así como la máquina virtual de propósito general asociada, supone un gran esfuerzo adicio-

[1] Esta decisión no es crítica ya que existen versiones de `flex` y `bison` para `C++` y `java`.

nal que, en nuestra opinión, no se justifica dado que para nuestro proyecto solo se emplearía una pequeña parte de los recursos y posibilidades disponibles.

En EDPC nos hemos decantado por un código intermedio 3-direcciones, muy parecido al propuesto en Aho y col. (2008). Además, este código 3-direcciones es similar al que se emplea en las clases de teoría, es simple e intuitivo y se adapta perfectamente a las necesidades del proyecto de compilación para la asignatura de LPyPL.

Como en el caso de los códigos intermedios generales, para poder evaluar el comportamiento del compilador necesitamos de una máquina virtual que nos permita ejecutar este código intermedio 3-direcciones que genere el compilador del proyecto. En EDPC esta máquina virtual se denomina `Malpas`. Tanto la definición del código intermedio 3-direcciones como la propia máquina virtual (`Malpas`) constituyen una de las partes más importantes del EDPC.

3.3 Material de apoyo

En la construcción de un compilador es necesario manejar muchas estructuras de datos conocidas (p.ej. tabla de símbolos) y generar código repetitivo de bajo nivel (p.ej. plantillas para la generación de código intermedio). Dado que los estudiantes han cursado asignaturas de algorítmica, programación básica y de estructuras de datos, sería absurdo que los alumnos tuvieran que desarrollar estos componentes básicos. En su lugar, el EDPC proporciona todo este código adicional, debidamente documentado, en forma de dos librerías:

- `libtds`. Librería con las operaciones para la correcta manipulación de la TDS.
- `libgci`. Librería con las operaciones para la gestión de memoria y la generación de código intermedio.

Dado que en el lenguaje de desarrollo se requiere una meticulosa atención a los detalles de gestión de memoria, cuando no se especifican con cuidado, los errores son a veces difíciles de encontrar. Los estudiantes pueden pasar más tiempo tratando de solucionarlos que aprendiendo como se construye un compilador. Facilitar este material de apoyo tiene otra ventaja adicional, ya que proporciona un nivel moderado de abstracción, elimina en gran medida los posibles errores y permite a los estudiantes centrarse en los aspectos más importantes del proyecto.

3.4 Compilador de referencia: estudio de caso

El EDPC también aporta un ejemplo o caso de estudio sencillo y derivado directamente de los ejercicios parciales realizados en las clases de teoría. Este compilador de ejemplo (referencia) desempeña varias funciones.

1. El ejemplo es sobradamente conocido por los alumnos y se ha extraído de los ejercicios propuestos y resueltos en teoría.
2. Este compilador de referencia sirve como ejemplo sencillo de uso del EDPC para resolver un problema conocido. Los alumnos pueden ver como se emplea el diverso material de apoyo y, sobre todo, el uso de las librerías.

3. Las partes en las que se descompone el compilador pueden compilarse y analizarse por separado, lo que apoya el proceso incremental del desarrollo del compilador.
4. El compilador de referencia sirve como un estudio de caso que los instructores pueden utilizar para guiar a los estudiantes en el desarrollo de su propio proyecto.
5. Los estudiantes pueden comparar sus soluciones adoptadas frente a un compilador de referencia para un ejemplo específico.

3.5 Documentación

La documentación que se aporta a los alumnos para la adecuada elaboración del proyecto de compiladores se puede resumir en:

- especificación formal completa (léxica, sintáctica y semántica) para lenguaje fuente `MenosC` del curso actual;
- descripción completa del material de apoyo, así como de las restricciones de su uso;
- guía para la elaboración del proyecto de compiladores;
- documentación adicional de todas las herramientas utilizadas en el proyecto: `flex`, `bison` y `malpas`;
- documentación y código fuente completo del compilador de referencia para el estudio de caso.

3.6 Programas de prueba para la autoevaluación

Para facilitar la tarea de autoevaluación de los alumnos, se proporciona igualmente una colección de programas de prueba en el lenguaje fuente del proyecto (`MenosC`). Estos programas de prueba permitirán a los alumnos evaluar cada una de las etapas en el desarrollo de su compilador: análisis léxico-sintáctico, análisis semántico y generación de código (intermedio).

4 Valoración de la experiencia

Uno de los aspectos más complicados en este tipo de metodologías activas es la creación de un sistema de evaluación que sea razonable y equitativo entre todos los alumnos. En la evaluación de un proyecto se debe valorar tanto el buen funcionamiento del proyecto final como el desempeño y la aportación de cada alumno individual al desarrollo del mismo. En LPyPL la evaluación se efectúa mediante la ponderación de la nota obtenida por los alumnos en tres aspectos:

- *Evaluación global del proyecto*, mediante la valoración del comportamiento del compilador para un conjunto de programas de prueba, en cada una de las tres partes en la que se descompone el proyecto.

- *Evaluación individual* del alumno, mediante una prueba práctica donde el alumno debe realizar una pequeña modificación de su compilador.
- *Actividades de seguimiento* en el laboratorio, mediante los correspondientes entregables asociados con cada una de las partes del proyecto, las encuestas finales de autoevaluación y evaluación entre iguales y la valoración de su tutor.

Esta experiencia la hemos llevado a cabo durante los 3 últimos cursos académicos (2013-14 al 2015-16) y ha afectado tanto al trabajo de los profesores como al desempeño de los alumnos. Desde el punto de vista del profesorado, la valoración de la experiencia ha sido muy positiva ya que hemos podido constatar la facilidad y la considerable reducción del trabajo necesario para la elaboración anual de un nuevo proyecto para LPyPL. Desde el punto de vista de los alumnos el EDPC también ha supuesto una considerable ayuda como lo demuestran algunos indicadores:

Porcentaje de proyectos entregados evaluados positivamente. En la Tabla 1 se puede observar un considerable aumento en el porcentaje de alumnos que han completado su proyecto, pasando de una media de 69, 7, antes de la implantación del EDPC, a una media de 91, 8, desde su implantación. Esta medida nos indica, de una manera indirecta, la dificultad del proyecto y el esfuerzo requerido para su elaboración. Si es excesivo, los alumnos tienden a abandonarlo a principio del curso. Mientras que si cuentan con la ayuda necesaria, los alumnos se implican más y se mantienen activos hasta la finalización del proyecto.

Tabla 1: Porcentaje de proyectos entregados evaluados positivamente

2011-12	2012-13	2013-14	2014-15	2014-15
66, 7 %	72, 8 %	93, 3 %	91, 1 %	91, 1 %

Porcentaje de alumnos presentados al examen individual de prácticas. En la Figura 1 se muestra este porcentaje: antes de la implantación del EDPC (2011-12 a 202-13) y desde su implantación (2013-14 a 2015-16). Como se puede observar hay un notable aumento del porcentaje de alumnos presentados al examen individual desde la implantación del EDPC. Esta medida da cuenta del grado de comprensión personal del alumno. Solo los alumnos que tienen claro el desarrollo del proyecto se suelen presentar al este examen. Para corroborar dicha afirmación baste señalar que la nota media del examen individual desde la implantación del EDPC ha sido de 8, 3.

5 Conclusiones

En este trabajo hemos presentado un entorno de desarrollo de proyectos de compiladores que permite simplificar en gran medida el proceso de creación de nuevos proyectos por parte del profesorado. Este EDPC no solo ha permitido reducir significativamente el esfuerzo de los profesores en la creación de un nuevo proyecto de compiladores, sino que también ha influido de manera positiva en los resultados obtenidos por los alumnos.

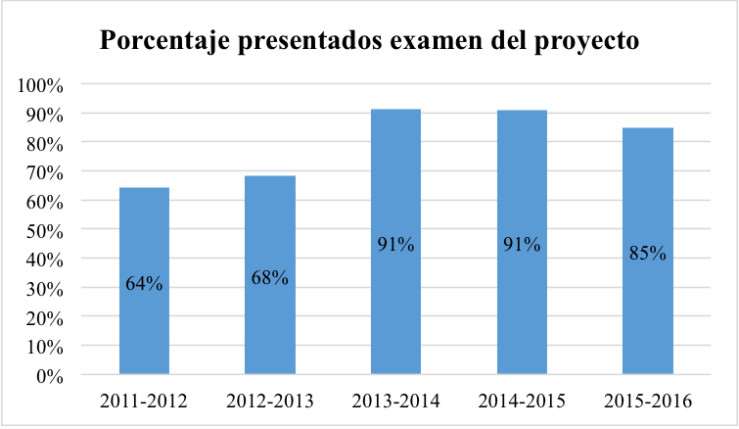

Fig. 1: Porcentaje de alumnos presentados al examen individual de prácticas

El EDPC contiene todo los materiales que los alumnos necesitan para construir su compilador y que, en resumen, incluyen: la especificación de un subconjunto del lenguaje de programación `C` (`MenosC`), como lenguaje fuente; la definición de un código intermedio 3-direcciones, como lenguaje objeto; una máquina virtual para la ejecución de ese código intermedio (`Malpas`); material de ayuda (en forma de librerías), para la simplificación del proceso de codificación; un compilador completo de referencia para un análisis de caso; documentación completa y unos programas de prueba para la autoevaluación.

Referencias bibliográficas

Aho, Alfred V (2008). "Teaching the compilers course". En: *ACM SIGCSE Bulletin* 40.4, págs. 6-8.

Aho, Alfred V. y col. (2008). *Compilers: Principles, Techniques, and Tools (2nd Edition)*. Addison Wesley.

Benedí, José Miguel y col. (2008). *Procesadores de Lenguajes: una introducción a la fase de análisis*. Universidad Politécnica de Valencia.

Cooper, Keith y Linda Torczon (2012). *Engineering a Compiler*. Morgan Kaufman.

Gil-Pérez, Daniel y Jaime Carrascosa-Alis (1994). "Bringing Pupils' Learning Closer to a Scientific Construction of Knowledge: A Permanent Feature in Innovations in Science Teaching". En: *Science Education* 78.3, págs. 301-315.

Larus, James (2011). *SPIM: A MIPS32 Simulator*. http://spimsimulator.sourceforge.net/.

Lindholm, Tim y col. (2011). *The Java Virtual Machine Specification*. 2015-06-26.

Waite, William M. (2006). "The Compiler Course in Today's Curriculum: Three Strategies". En: *SIGCSE Bull.* 38.1, págs. 87-91.