Reflexiones y experiencias sobre la enseñanza de POO como único paradigma

Daniel Gayo Avello, Agustín Cernuda del Río, Juan Manuel Cueva Lovelle,
Marián Díaz Fondón, Mª Pilar Almudena García Fuente, José Manuel Redondo López
Departamento de Informática
Universidad de Oviedo
C/Calvo Sotelo s/n 33007 Oviedo
e-mail: {dani, guti, cueva, fondon, almu, redondo}@lsi.uniovi.es

Resumen

En la actualidad el paradigma de orientación a objetos (en adelante OO) está ampliamente aceptado y todos los Planes de Estudios modernos consideran necesario impartir una o más asignaturas relacionadas con el análisis, el diseño y la programación orientados a objetos en las titulaciones universitarias en informática.

Sin embargo, existe un candente debate sobre el momento de presentar a los alumnos la programación orientada a objetos (en adelante POO). Algunos docentes consideran problemático empezar directamente con POO y creen necesario enseñar previamente programación estructurada. Otros profesores opinan de manera totalmente opuesta.

En este artículo expondremos nuestros argumentos a favor de una enseñanza temprana de la POO, así como los contenidos teóricos y prácticos impartidos en una asignatura de Introducción a la Programación (OO) de primer curso, además de los resultados y las conclusiones alcanzadas en este primer año de andadura.

1. Introducción

Desde hace algún tiempo se viene debatiendo la forma de introducir la POO en las enseñanzas universitarias en informática. Se trata de un debate fuertemente polarizado. En un lado nos encontramos aquellos que consideramos que es necesario comenzar a explicar a los alumnos OO desde el primer día. En el otro se sitúan los que creen que se debe empezar por la programación estructurada para, después, enseñar OO.

Con este artículo no pretendemos echar más leña al fuego; somos conocedores de los inconvenientes de ambas opciones, pero el hecho es que forzosamente hay que decantarse por una y asumir dichos inconvenientes. Aquí pretendemos simplemente describir por qué y cómo estamos enseñando a alumnos de primer curso de Ingeniería Técnica en Informática POO desde el primer día sin encontrar más dificultades de las que se encontraban al enseñar programación estructurada y con unos resultados provisionales que juzgamos bastante satisfactorios.

En primer lugar partiremos de la hipótesis de que el paradigma OO, sin ser la panacea, es la mejor opción disponible en este momento para el desarrollo de software. Sobre esa base, argumentaremos que si la OO es un objetivo deseable es conveniente afrontarlo desde el primer momento y de la forma más directa posible. A continuación definiremos lo que entendemos por "introducción a la programación" y la forma en que semejante definición encaja en el programa de una asignatura de primer cuatrimestre. Posteriormente, describiremos la manera en que se conducen las clases, se expondrán algunos ejercicios llevados a cabo con los alumnos y se comentarán brevemente los resultados parciales que se han obtenido, así como las impresiones de alumnos que han experimentado en cursos anteriores la enseñanza de programación estructurada como paso previo a la POO.

2. ¿Es útil la orientación a objetos?

Muchos conceptos básicos de la OO se remontan a finales de los años 60 (lenguaje Simula) y 70 (Smalltalk) y es a mediados de los años 80 cuando la OO comienza a despuntar. Hoy en día la OO es

considerada por la inmensa mayoría de las personas involucradas en el desarrollo de software como el mejor paradigma disponible para construir sistemas informáticos.

Sin embargo, el hecho de que un grupo numeroso afirme que algo es cierto no implica que necesariamente lo sea. En el OOPSLA 2002 se celebró un debate en torno al fracaso de los objetos. Apoyaba esta tesis R.P. Gabriel y la refutaba G.L. Steele Jr., ambos de *Sun Microsystems*.

Entrar en detalles sobre tan interesante discusión no es el objeto de este escrito pero puede resultar ilustrativo mencionar algunos puntos allí tratados.

Gabriel argumenta el fracaso de los objetos apoyándose en la incapacidad de la OO para modelar de manera adecuada futuros requisitos (funcionamiento distribuido, ubicuidad y cooperación entre componentes), en la pérdida de la simplicidad de los lenguajes originales, en el escaso índice de reutilización así como en la falsa sensación de facilidad de aprendizaje cuando "en realidad (programar) es tan difícil como siempre" [7]. Sin embargo, la principal crítica que hace a la OO es que "asfixia" la investigación y desarrollo de nuevos paradigmas de programación.

Por su parte, Steele [13] afirma que la OO es un éxito: la mayor parte de los desarrolladores emplean lenguajes OO y los objetos son un buen modelo para la mayoría de entidades del mundo real. Señala que el énfasis en procesar entradas de datos para transformarlas en salidas es una debilidad de la programación procedimental que ha hecho que la OO fuese la respuesta a muchos problemas que la primera difícilmente podía resolver. Naturalmente, acepta que la OO no puede resolver todos los problemas de la programación, que la evolución del paradigma no se ha completado (y tal vez nunca se haga) y que los lenguajes actuales no son los mejores lenguajes OO posibles.

Por nuestra parte, estamos de acuerdo en que es necesario desarrollar más el paradigma OO y otros nuevos; sin embargo, estamos con Steele cuando afirma que "la POO es como el dinero: no lo es todo, pero es mucho mejor que las otras opciones". Por otro lado, también coincidimos con Gabriel en que la POO no es más sencilla, sólo igual de difícil que la programación procedimental.

3. Orientación a objetos en los planes de estudio

En resumen, la OO y los lenguajes, métodos y herramientas asociados no son una solución óptima, pero es lo mejor que hay disponible y el mercado demanda profesionales formados en estas técnicas. Esta necesidad ha afectado finalmente a la forma en que se plantean los planes de estudio de las titulaciones en informática.

El *Computing Curricula 2001* de *ACM* e *IEEE Computer Society* [1] señala la OO como un cambio tecnológico relevante en la enseñanza universitaria de la informática. Considera que la programación, el análisis y el diseño OO deben formar parte de cualquier plan de estudios en informática y plantea varios enfoques para su enseñanza, interesándonos especialmente los denominados "Primero imperativo" y "Primero objetos". En el siguiente apartado argumentaremos por qué consideramos conveniente aplicar el segundo planteamiento.

4. ¿Por qué empezar por objetos?

Hemos establecido que la OO es un paradigma extendido, con un amplio soporte por parte de la industria y que debe ser conocido por cualquier estudiante universitario de informática. Así pues, la cuestión no es si se debe enseñar OO sino en qué momento se debe presentar; básicamente, se trata de elegir entre las dos opciones planteadas en el mencionado informe de *ACM/IEEE*.

Este el centro del debate que mencionábamos en la introducción: "¿Se debe enseñar desde el primer día POO?" Sin embargo, creemos que la pregunta que se formula en semejante debate debería replantearse como: "¿Se debe enseñar desde el primer día POO sabiendo que la OO es un objetivo fundamental de una titulación universitaria en informática?"

Entendemos que la primera pregunta, al estar planteada incorrectamente, ofrece muchas posibilidades de polémica. Sin embargo, si aceptamos que la OO es un paradigma útil y que los alumnos deben comprenderlo y manejarlo con soltura es necesario, en nuestra opinión, comenzar desde el primer momento con la POO (en especial, en titulaciones de ciclo corto como la impartida en nuestra Universidad).

4.1. Posturas opuestas a un enfoque temprano

Muchos autores consideran un error iniciar la enseñanza de la programación comenzando con la OO. Tan sólo en JENUI 2002 se presentaron cuatro ponencias que desaconsejaban tal opción.

Así, Fernández Muñoz et al ([5] y [6]) señalan los siguientes problemas: sobrecarga de conceptos teóricos o sobresimplificación, utilización de bibliotecas de clases, utilización de interfaces gráficos o entornos de desarrollo artificiosos. García Molina et al [9] proponen retrasar las asignaturas de POO hasta el tercer curso de la titulación. Por su parte, Gómez Albarrán [10] propone una asignatura de introducción a la programación que comienza con la descomposición funcional para pasar, posteriormente, a la OO. Todas estas propuestas pueden encuadrarse en la línea de "objetos más tarde".

4.2. ¿Qué paradigma enseñar si la OO es el objetivo?

Las propuestas anteriores plantean algo en apariencia muy sencillo: primero se enseña a los alumnos el paradigma procedimental así como TAD's. Una vez han asimilado los conceptos deberían aprender fácilmente POO puesto que "puede considerarse un paradigma construido sobre la base del paradigma imperativo" [8].

Sin embargo, en la práctica es frecuente que los alumnos se encuentren con el problema del "cambio de paradigma". Stroustrup [14] asegura que un programador experimentado en el paradigma procedimental necesita entre 6 y 18 meses para adaptarse al paradigma OO. Probablemente sea pedir demasiado que nuestros alumnos se adapten mejor que un profesional.

Hay otras razones de peso que aconsejan comenzar por la OO si es el objetivo perseguido en la formación de los estudiantes. Todas ellas se derivan de una premisa muy simple: a pesar de las similitudes no puede decirse que la OO sea una mera extensión del paradigma procedimental.

Bergin [2] señala que el problema de enseñar el paradigma procedimental como paso previo a la POO radica en que las estrategias de resolución de problemas de ambos paradigmas son radicalmente distintas. Así, la programación procedimental fomenta la división de problemas en subproblemas que deben ser resolubles a partir de una serie de datos que serán procesados y transformados, a su vez, en nuevos datos. Así pues, en la programación procedimental el algoritmo es lo que cuenta y los datos son algo secundario, circunstancia poco frecuente en el mundo real.

En parte, esta debilidad de la programación procedimental ha facilitado el camino de la POO. Ésta no busca en primer lugar soluciones a los problemas sino que trata de modelar el sistema como objetos que tendrán un comportamiento, un estado y unas relaciones de colaboración entre los mismos.

Por ello, aun cuando los lenguajes OO sean imperativos y se puedan establecer ciertos símiles entre métodos y subrutinas o entre objetos y TAD, lo cierto es que la forma en que se desarrolla un sistema informático para resolver un problema del mundo real es muy diferente según el paradigma que se emplee; de ahí el problema del cambio de paradigma y la afirmación tajante de Bergin de que el paradigma procedimental es la opción equivocada si lo que se quiere es enseñar OO. No podemos por menos que estar de acuerdo con su postura.

5. Una propuesta para la introducción a la programación

Un aspecto interesante de todas las propuestas mencionadas en el apartado 4.1 es que el citado problema del "cambio de paradigma" no parece producirse. Sería muy interesante estudiar qué aspectos de la actividad docente de los respectivos autores permiten evitar tal problema, pero en nuestro caso lo hemos sufrido de forma clara y reiterada, hasta el punto de que lo consideramos una premisa en las decisiones que tomamos.

Debido en parte a nuestra experiencia y convencidos de la importancia de la OO para los alumnos se decidió aprovechar la entrada del nuevo Plan de Estudios[1] para construir una asignatura de introducción a la programación donde el único paradigma que estudiaría el alumno fuese la OO.

Desde nuestro punto de vista el objetivo primordial de tal asignatura sería proporcionar a los alumnos conocimientos y estrategias básicos para poder enfrentarse a un problema del mundo real susceptible de ser solucionado informáticamente,

[1] http://www.euitio.uniovi.es/nuevoplan/spanish

determinar los objetos que podrían modelar tal sistema, describir con una notación precisa el modelo a construir e implementar dicho modelo mediante un lenguaje de programación OO.

En el caso que nos ocupa, dicha asignatura, denominada Introducción a la Programación, consta de 3 créditos de teoría, 1 crédito de prácticas de tablero y 2 créditos de prácticas de laboratorio y se imparte en el primer cuatrimestre.

A la hora de elaborar el programa y seleccionar las herramientas a emplear se trataron de alcanzar los siguientes objetivos:

1. Seguir, hasta donde fuera posible, las propuestas "objetos primero" de *ACM/IEEE* [1].
2. Lograr que el alumno aprendiese POO y no las peculiaridades del lenguaje de programación elegido.
3. Presentar los contenidos teóricos de manera que se facilitase su comprensión y aprendizaje progresivo y, a la vez, permitiese el desarrollo de ejercicios prácticos desde el primer momento.
4. Minimizar el impacto de la herramienta de desarrollo en la realización de los ejercicios.
5. Presentar a los alumnos técnicas básicas de análisis y diseño orientado a objetos así como los elementos mínimos de la notación UML.
6. Evitar, en la medida de lo posible, problemas habituales en este tipo de iniciativas como la utilización de entornos artificiosos, el énfasis en el desarrollo de aplicaciones gráficas o la presencia excesiva de bibliotecas de clases.

En base a tales objetivos y teniendo en cuenta el tiempo disponible se elaboró el programa de la asignatura cuyos bloques teóricos y prácticos principales se muestran en Figura 1 y Figura 2. En los apartados siguientes se explicará de forma detallada la forma en que se están impartiendo estos contenidos.

5.1. Contenidos teóricos

Los contenidos teóricos de la asignatura se reparten en siete bloques. El primero permite presentar a los alumnos una serie de conceptos fundamentales (en especial el de algoritmo) así como la forma en que la informática ha evolucionado para resolver la antigua necesidad de realizar cálculos y procesar datos de manera automatizada. Además, se explica a los estudiantes el funcionamiento de la arquitectura Von Neumann, la forma en que se representa la información en un ordenador y los distintos tipos de lenguajes de programación existentes.

1. ANTECEDENTES
   - 1.1. Historia de la Informática
   - 1.2. Lenguajes de programación
2. INTRODUCCIÓN A LA PROGRAMACIÓN ORIENTADA A OBJETOS
   - 2.1. Paradigma OO.
   - 2.2. Encapsulación y ocultación de información.
   - 2.3. Abstracción y genericidad.
   - 2.4. Comportamiento de los objetos.
   - 2.5. Constructores y destructores.
   - 2.6. Recolección de basura.
3. ESTADO DE LOS OBJETOS
   - 3.1. Tipos de datos. Tipos primitivos.
   - 3.2. Declaración de atributos
   - 3.3. Expresiones y asignación.
   - 3.4. Comprobación de tipos.
4. COMPORTAMIENTO DE LOS OBJETOS
   - 4.1. Métodos.
   - 4.2. Paso de parámetros.
   - 4.3. Declaración de variables
   - 4.4. Sobrecarga de métodos.
5. ESTRUCTURAS DE CONTROL
   - 5.1. Propiedades de las estructuras de control.
   - 5.2. Estructura secuencial.
   - 5.3. Estructuras alternativas.
   - 5.4. Estructuras repetitivas.
6. CLASIFICACIÓN DE OBJETOS
   - 6.1. Clases y subclases.
   - 6.2. Interfaces.
   - 6.3. Herencia de clases e interfaces. Redefinición de métodos.
   - 6.4. Jerarquías de clases.
7. ESTRUCTURAS DE DATOS FUNDAMENTALES
   - 7.1. Arrays.
   - 7.2. Cadenas de caracteres.

Figura 1. Temario teórico de "Introducción a la Programación"

Prácticas de tablero

1. INTRODUCCIÓN A LA PROGRAMACIÓN ORIENTADA A OBJETOS
2. ESTADO DE LOS OBJETOS
3. COMPORTAMIENTO DE LOS OBJETOS
4. ESTRUCTURAS DE CONTROL
5. CLASES DE OBJETOS
6. ESTRUCTURAS DE DATOS FUNDAMENTALES

Prácticas de laboratorio

1. PRESENTACIÓN DEL ENTORNO DE DESARROLLO.
2. COMPORTAMIENTO DE LOS OBJETOS (I).
3. DOTANDO DE ESTADO A LOS OBJETOS.
4. COMPORTAMIENTO DE LOS OBJETOS (II).
5. JERARQUÍAS DE CLASES.
6. DESARROLLO DE UNA APLICACIÓN COMPLETA.

Figura 2. Temario práctico de "Introducción a la Programación"

Una vez los alumnos han asimilado el concepto de algoritmo es posible plantearles sencillos problemas para los que se puede encontrar un algoritmo sin demasiadas dificultades. Sin embargo, la finalidad de esos ejemplos no es explicar el paradigma procedimental sino exponer sus debilidades.

Para ello, se pide a los estudiantes que traten de imaginar un algoritmo que resuelva un problema muy complejo (por ejemplo, gestionar un sistema de control de tráfico aéreo). Naturalmente, coinciden en que desarrollar semejante algoritmo puede resultar muy arduo.

A continuación se pide a los alumnos que tra-

ten de identificar objetos que sean relevantes en el ámbito de dicho problema, así como el comportamiento que mejor los defina. Esta tarea no les resulta demasiado complicada y sí bastante motivadora.

Tras este contacto intuitivo con el paradigma se dedican al menos tres sesiones a explicar los conceptos básicos del mismo. Aquí se tocan los temas fundamentales de manera breve y tratando de mantener una visión global: diferencias entre clase y objeto, comportamiento y estado, métodos y atributos, constructores, herencia, etc.

La mayoría de los alumnos no comprenden de una forma clara muchos de los conceptos más difíciles; es necesario insistir en que lo fundamental en este momento no es comprenderlo todo sino saber de su existencia y conocer, a grandes rasgos, las relaciones entre los distintos conceptos.

Una vez se ha presentado una panorámica de la POO se puede comenzar a entrar en detalles referentes a estado y comportamiento. Así, al explicar el concepto de estado es posible introducir los conceptos de tipos de datos, tipos primitivos, declaración de variables miembro, etc. Al explicar el comportamiento de los objetos deberían tocarse temas especialmente relevantes como definición de métodos, variables locales, paso de parámetros, sobrecarga, etc. junto con las estructuras de control básicas.

Finalmente se explicarían de manera muy sencilla herencia, interfaces, redefinición de métodos y se introduciría brevemente el polimorfismo.

Por último, se explicarían los principales métodos de la clase `String` y *arrays*. La razón para retrasar hasta el final de la asignatura el concepto de *array* es simple. Mientras los alumnos no manejen adecuadamente los conceptos de clase y objeto además del mecanismo de herencia no es posible desarrollar ejercicios relativamente complejos; y será en dichos ejercicios donde los *arrays* sean verdaderamente útiles para poder implementar de una forma básica relaciones 1..n.

Respecto a la forma de representar los conceptos de cara a su explicación hemos optado por emplear la notación UML. Sin embargo, no hemos hecho ningún esfuerzo específico por enseñar la notación a los alumnos; simplemente hemos ido introduciendo símbolos a medida que eran necesarios. En estos momentos en los que la asignatura está prácticamente finalizada los alumnos son capaces de desarrollar por sí solos modelos con varias clases, empleando adecuadamente relaciones de herencia y asociaciones con multiplicidad.

5.2. Lenguaje de programación elegido

Para el desarrollo de las prácticas era posible elegir entre muchos lenguajes: Smalltalk, Eiffel, C++, Java, etc. Todos presentan inconvenientes a la hora de ser empleados como un lenguaje para enseñanza, resultando la opción "menos mala" Java [11].

Como argumentos a favor del uso de Java podemos citar los siguientes:

1. Es un lenguaje OO prácticamente puro.
2. La sintaxis es sencilla y legible, y permite expresar la mayor parte de conceptos OO de una manera simple.
3. Es un lenguaje relativamente pequeño.
4. Los alumnos pueden trabajar con él sin tener que conocer un número excesivo de clases de la biblioteca.
5. Sólo permite herencia simple.
6. El alumno no tiene que preocuparse de la reserva y liberación de memoria.
7. Es ampliamente utilizado por la industria y facilita la transición a otros lenguajes OO.

Obviamente, Java presenta una serie de inconvenientes, entre los que se pueden mencionar:

1. La existencia de tipos simples puede confundir al principio a los estudiantes.
2. La entrada/salida requiere la utilización de clases de la biblioteca del lenguaje y resulta, cuando menos, engorrosa.
3. Hasta que los alumnos se acostumbran a que es un lenguaje sensible a mayúsculas las clases contenedoras de tipos simples como `Boolean`, `Integer` o `Float` pueden causar más de un problema.
4. El método `main` es totalmente ajeno a los conceptos OO y precisa entre una y dos clases para ser correctamente explicado obligando, además, a introducir el concepto de métodos estáticos muy pronto.

De los problemas anteriores los más graves son el segundo y el cuarto; a continuación se describirá la forma en que hemos tratado de solucionarlos en la asignatura que impartimos.

Para llevar a cabo entrada y salida en Java sería necesario, en principio, explicar a los alumnos en un momento muy temprano la existencia de la biblioteca de clases de Java para

poder emplear los métodos de las clases `System.out` y `System.in`. Para evitar esto optamos por construir un paquete `Consola` que implementa las clases `Pantalla` y `Teclado` y explicamos los métodos para dichas clases (por ejemplo, `escribir` o `leerFlotante`).

Las diferencias pueden parecer triviales pero son importantes para los alumnos:

1. El alumno percibe perfectamente que `Pantalla` y `Teclado` son elementos ajenos al lenguaje; después de todo fueron escritas por los profesores y tienen "nombres en español". Esta tarea no es tan sencilla al emplear clases aportadas por la biblioteca de Java.
2. La sentencia `import` es relativamente sencilla de entender si se explica en términos de directorios y subdirectorios.
3. Es posible presentar las clases `Pantalla` y `Teclado` en una lección de teoría para que los alumnos vean qué es exactamente un objeto, la forma en que proporciona un comportamiento para cumplir con sus responsabilidades así como la forma en que se invocan mensajes.

Explicar el método `main` de una manera razonable es algo más complicado. En primer lugar, requiere que el alumno comprenda perfectamente el concepto de algoritmo, la arquitectura Von Neumann y la forma en que ésta permite ejecutar algoritmos.

Una vez estos conceptos están claros es necesario explicar las diferencias entre lenguajes de bajo y de alto nivel así como el papel jugado por los distintos tipos de traductores.

En ese momento se puede explicar de un modo muy sencillo la forma en que Java depende de una máquina virtual (JVM) y cómo el código fuente Java es traducido a *bytecode* que será ejecutado por la JVM que, a su vez, se ejecutará sobre una arquitectura Von Neumann.

Si todo esto queda claro a los alumnos (lo cual puede llevar toda una sesión de teoría) pueden entender que, de algún modo, aunque Java sea un lenguaje OO tiene que acabar siendo traducido a código de bajo nivel que se ejecutará en una arquitectura Von Neumann. Puesto que dicha arquitectura requiere que un algoritmo tenga un punto de inicio y las instrucciones se ejecutan secuencialmente no les resulta difícil aceptar que de alguna forma es necesario indicar en el propio código Java qué método de qué clase será el primero en ejecutarse.

Por supuesto, es necesario indicar a los alumnos que el método `main` es ajeno al paradigma OO y que el código que se coloque en su interior debería ser siempre el menor posible.

5.3. Herramientas de desarrollo

Un problema común a la hora de impartir una asignatura de programación se plantea en el momento de seleccionar el entorno de desarrollo, especialmente si se trata de un lenguaje no orientado al ámbito académico.

En nuestro caso debíamos seleccionar un entorno[2] que permitiese el desarrollo de aplicaciones OO en Java y satisficiese, en la medida de lo posible, los siguientes criterios [12]: facilidad de uso, entorno integrado (editor, compilador y depurador), IGU con soporte para manipulación de objetos y libre disponibilidad para los estudiantes.

Las opciones que se plantearon fueron las siguientes: Kawa, BlueJ, Borland JBuilder Personal y Visual J++.

Kawa[3] era un sencillo IDE proporcionado por *Macromedia*; sin embargo, actualmente carece de soporte y no tenía sentido animar a los estudiantes a utilizar una herramienta sin futuro.

BlueJ[4] es la única de las herramientas anteriores específicamente pensada para su utilización académica. Es gratuita, muy fácil de usar y permite trabajar en un modo "visual" empleando UML. Además, el alumno puede instanciar objetos y enviarles mensajes directamente, con lo cual dos problemas de Java se solucionan de una forma sencilla: la entrada/salida y el método `main`.

Borland JBuilder Personal[5] es una versión muy reducida de la herramienta Borland JBuilder

[2] ¿Por qué no utilizar simplemente el JDK? Puede resultar muy instructivo enfrentarse a compiladores y depuradores en línea de órdenes. Sin embargo, la finalidad de la asignatura es aprender POO empleando Java y no aprender Java *per se*; por ello, el uso didáctico del JDK carecía de sentido. Además, requeriría enseñar a la mayoría de los alumnos MS-DOS consumiendo un tiempo precioso. Por último, aun cuando fuesen capaces de emplear el JDK, la ausencia de entorno les distraería de lo realmente importante: la POO.

[3] http://www.macromedia.com/software/kawa

[4] http://www.bluej.org

[5] http://www.borland.com/jbuilder/personal

siendo, por tanto, muy similar a otros IDE's comerciales. La principal ventaja de esta herramienta es que cualquier usuario puede obtener una licencia de uso personal de manera gratuita. Visual J++ es similar aunque más complicado, razón por la que fue descartado.

Así pues, había dos opciones: BlueJ y JBuilder. Por motivos didácticos la opción más deseable era BlueJ. No obstante, su utilización planteaba dudas sobre posibles problemas por parte de los alumnos al tener que cambiar a un IDE "real".

Si dicho cambio se produjese en un nuevo curso habríamos elegido BlueJ; sin embargo, por diversas razones la asignatura "Interacción Persona-Ordenador" (IPO) se imparte en el segundo cuatrimestre del primer curso. Al no encontrarse ningún entorno sencillo para el diseño de IGU's que pudiese ser integrado con BlueJ nos vimos obligados a emplear Borland JBuilder Personal para no obligar a los alumnos a aprender una herramienta distinta en cada cuatrimestre. Fueron necesarias dos sesiones prácticas para presentar el entorno y muchos alumnos no comenzaron a manejarlo de manera autónoma hasta la tercera o la cuarta sesión.

5.4. Ejercicios prácticos

Para superar la asignatura los estudiantes deben aprobar un examen práctico o bien superar con éxito dos ejercicios que son desarrollados en gran parte durante las sesiones prácticas y finalizados fuera del horario docente.

El objetivo del primer ejercicio es consolidar dos conceptos básicos de la POO como son estado y comportamiento. Para ello el alumno parte de una clase básica denominada `Inichi` que implementa una sencilla mascota virtual [3].

Para superar el ejercicio el alumno debe añadir a su mascota una serie de métodos y atributos que la doten de un comportamiento más complejo. Una parte importante de la evaluación de este ejercicio obliga al alumno a realizar, bajo condiciones controladas, una pequeña ampliación que requiere algunos atributos y métodos nuevos.

En el segundo ejercicio [4] el alumno debe desarrollar una aplicación completa que requiere el uso de varias clases. Para ello puede optar por desarrollar su propio modelo de clases o bien ampliar uno de los distintos modelos propuestos por los profesores. Este ejercicio se entregará el mismo día del examen escrito.

5.5. Resultados parciales

En estos momentos la asignatura está a punto de finalizar; los alumnos ya han realizado un control teórico y se ha evaluado el primer ejercicio práctico, estando pendientes el examen escrito, la entrega del segundo ejercicio práctico y el examen práctico para quienes no sean evaluables mediante el sistema de evaluación continua.

El citado control se llevó a cabo tras apenas 6 semanas de clase y abarcó los tres primeros bloques del temario. El 85% de los matriculados se presentaron al examen y se obtuvieron los resultados que se muestran en Figura 3. En cuanto a la evaluación de la primera práctica se han obtenido los siguientes resultados: la han presentado el 81% de los matriculados, el 9% de los ejercicios eran "fraudulentos" (copias, prácticas en equipo, etc), de las prácticas válidas un 14% suspendieron, un 33% aprobaron, un 28% obtuvieron notable y un 25% sobresaliente.

Además, tras ocho semanas de clase se realizó una encuesta entre el alumnado que había cursado Metodología de la Programación I (una asignatura del anterior Plan de Estudios que comenzaba con programación estructurada y finalizaba con algunos conceptos de POO). Los resultados se muestran en Figura 4.

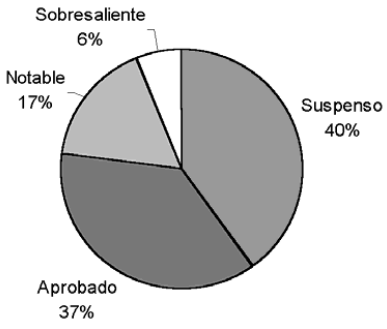

Figura 3. Resultados del control teórico

Paradigma preferido	¿Dificultad teoría POO?	¿Dificultad práctica POO?	Factores influyen percepción dificultad
78% OO	30% Más difícil	19% Más difícil	37% Cambio de Pascal a Java
6% Indiferente	22%Similar	23% Similar	49% Comenzar con POO
16% Progr. Estructurada	48% Más fácil	58% Más fácil	40% Haber estudiado antes programación

Figura 4. Resultados de la encuesta entre repetidores

6. Conclusión

Esta experiencia de enseñanza de la OO como único paradigma de programación está resultando muy positiva. Creemos que no resulta más difícil para el alumno que la programación estructurada y, además, el grado de satisfacción de los alumnos con el enfoque, el lenguaje y la herramienta de desarrollo son altos. Creemos que esto se debe a una serie de factores motivadores: el alumno aprende un lenguaje que se emplea en la industria, utiliza una versión de una herramienta profesional y se siente especialmente motivado por el propio paradigma. No pensamos que esto se deba a que la OO sea más intuitiva o sencilla que la programación estructurada; simplemente es más potente a la hora de enfrentarse a problemas reales y creemos que esta percepción de poder por parte de alumnos de primer curso es lo que les alienta.

Referencias

[1] ACM/IEEE. *Computing Curricula 2001*. http://www.computer.org/education/cc2001/report

[2] Bergin, J. *Why Procedural is the Wrong First Paradigm if OOP is the Goal.* http://csis.pace.edu/~bergin/papers/Whynotproceduralfirst.html

[3] Díaz Fondón, M., García Fuente, A., Gayo Avello, D. *Desarrollo de una mascota virtual.* http://www.euitio.uniovi.es/~ip/practica09.pdf

[4] Díaz Fondón, M., García Fuente, A., Gayo Avello, D., Redondo López, J.M. *Evaluación de conductores con dispositivos PDA*. http://www.euitio.uniovi.es/~ip/practica10.pdf

[5] Fernández Muñoz, L., Peña, R., Nava, F., Velázquez Iturbide, A. *Análisis de las propuestas de la enseñanza de la programación orientada a objetos en los primeros cursos*. JENUI 2002. Cáceres, España.

[6] Fernández Muñoz, L., Peña, R., Nava, F., Velázquez Iturbide, A. *Enfoque diacrónico para la enseñanza de la programación imperativa*. JENUI 2002. Cáceres, España.

[7] Gabriel, R.P. *Objects have failed*. http://www.dreamsongs.com/NewFiles/ObjectsHaveFailed.pdf

[8] García Molina, J. *¿Es conveniente la Orientación a Objetos en un primer curso de programación?* Novática 154, 2001.

[9] García Molina, J., Menárguez Tortosa, M., Moros Valle, B. *Una propuesta para organizar la enseñanza de la Orientación a Objetos*. JENUI 2002. Cáceres, España.

[10] Gómez Albarrán, M. *Metodología basada en descomposición funcional y orientación a objetos en la introducción a la programación*. JENUI 2002. Cáceres, España.

[11] Kölling, M. *The Problem of Teaching Object-Oriented Programming, Part 1: Languages*. Journal of Object-Oriented Programming, Vol. 11 No. 8, 8-15, 1999. http://www.mip.sdu.dk/~mik/papers/oo-languages.pdf

[12] Kölling, M. *The Problem of Teaching Object-Oriented Programming, Part 2: Environments*. Journal of Object-Oriented Programming, Vol. 11 No. 9, 6-12, 1999. http://www.mip.sdu.dk/~mik/papers/oo-environments.pdf

[13] Steele Jr., G.L. *Objects have not failed*. http://www.dreamsongs.com/ObjectsHaveNotFailedNarr.html

[14] Stroustrup, B. *The Design and Evolution of C++*. Addison-Wesley, 1994.