Análisis de las propuestas de la enseñanza de la programación orientada a objetos en los primeros cursos

L. Fernández Muñoz (a), R. Peña(b), F. Nava (c), Á. Velázquez Iturbide(c)
(a) Dept. LPSI. Universidad Politécnica. 28071 Madrid e-mail: setillo@eui.upm.es
(b) Facultad de Documentación. Universidad de Alcalá. 28871 Madrid e-mail: rpr@uah.es
(c) Escuela Superior de Ciencias Experimentales. Universidad Rey Juan Carlos. 28933 Madrid
e-mail: {f.j.nava,a.velazquez}@escet.urjc.es

Resumen

Existe un fuerte debate sobre qué paradigma conviene impartir en el primer curso de los planes de estudio de Informática: orientación a objetos (OO) o procedimental, así como sobre el lenguaje de soporte de las sesiones prácticas. Muchos artículos presentan sus experiencias o proponen nuevas metodologías de forma aislada, pero no encontramos un estudio sistemático de los distintos enfoques. En este trabajo se presenta una visión crítica de las propuestas de diferentes autores, se evalúan sus ventajas e inconvenientes. Se concluye la necesidad de reconducir el debate de la planificación de los primeros cursos de programación, hoy día centrado en el orden en que se enseñan los paradigmas y el lenguaje para implementarlo, hacia la búsqueda de los conceptos fundamentales y herramientas pedagógicas que permitan una exposición gradual de los conceptos de la programación.

1. Introducción

En la década de los noventa, la mayoría de los centros de enseñanza superior de Informática ha ido integrando conceptos de OO en sus planes de estudio, como se refleja en los datos aportados por [11,18,27,37,42,49]. La última propuesta curricular del grupo conjunto de ACM e IEEE [2] incluye la OO como una de las materias claves en las titulaciones universitarias de Informática.

Inicialmente, la programación OO se ubicó en segundo curso o superiores. Actualmente hay un debate abierto sobre cuál es el primer paradigma al que debe enfrentarse al alumno. Unos abogan por presentar antes un paradigma procedimental [4-6,9,10,15,23,26,36,39,45,53,56], basándose en la necesidad de no aumentar el gran número de conceptos que es necesario presentar al alumno que se acerca por primera vez a la programación; mientras otros abogan por la OO como primer paradigma, basándose en su carácter intuitivo, potencia del lenguaje, y la fuerte motivación que provoca en los alumnos [3,7,16,18,21,25,28,33-37,47,48,51,55,58,59,61-63].

Esta discusión es relevante también desde el punto de vista administrativo, ya que la disparidad de criterios en la planificación de los primeros cursos provoca dificultades en la movilidad de los alumnos [11], que se pretende fomentar en Europa [17], resultando importante homogeneizar los enfoques adoptados por los centros.

Este artículo presenta una panorámica de las diferentes propuestas, las clasifica y sistematiza, resaltando sus ventajas e inconvenientes. La figura 1 sintetiza esta clasificación.

La estructura del artículo es la siguiente: el apartado 2 resume las recomendaciones curriculares de ACM/IEEE para la docencia de la programación de los primeros cursos. En los apartados 3 y 4 estudiamos, respectivamente, las propuestas que presentan el paradigma procedimental en primero y el OO en segundo, y las que abordan el paradigma OO desde el principio.

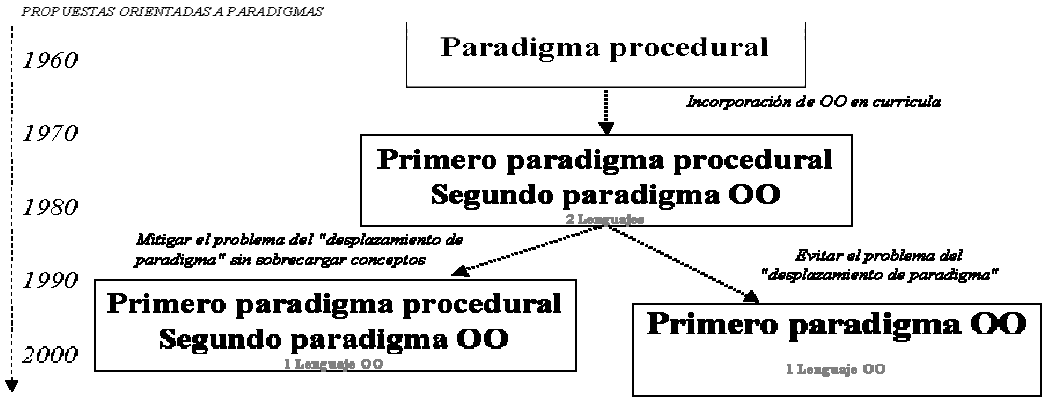

Figura 1 Evolución de los enfoques de la enseñanza de los paradigmas

2. Recomendaciones ACM/IEEE2001

La última propuesta curricular ACM/IEEE avala la seriedad de la polémica acerca del momento adecuado de introducir al alumno en la OO. Reconoce el debate y lo amplía proponiendo, no dos, sino seis diseños alternativos para los cursos de introducción a la programación:

- comenzando con el paradigma procedimental: Incluso usando un lenguaje OO, el primer curso se centra sobre los aspectos imperativos del lenguaje.
- comenzando con el paradigma OO: Empieza directamente con nociones de objetos y herencia.
- comenzando con el paradigma funcional: Se caracteriza por usar un lenguaje funcional sencillo.
- presentación en amplitud: El primer curso, además de atender la programación, algoritmos y estructuras de datos integra otras subdisciplinas, como las matemáticas.
- enfoque algorítmico: Los conceptos básicos se introducen usando pseudocódigo en vez de un lenguaje ejecutable.
- enfoque arquitectónico: Comienza con circuitos, con los que construye una máquina de von Neumann simple. Establecidos los fundamentos del *hardware*, presenta un lenguaje de alto nivel.

Las dos primeras propuestas son las que afectan a nuestra discusión. Las experiencias publicadas que tratan la polémica se pueden clasificar como: procedimental en primero y OO en segundo versus OO en primero.

3. Paradigma procedimental en primero y OO en segundo

Esta línea aborda primero la programación procedimental y, una vez asentados los conceptos de control de flujo de ejecución y de tipos abstractos de datos, se introducen los conceptos de OO.

La OO viene a resolver la complejidad de la gestión del software. No es adecuada para los ejercicios correspondientes a un primer acercamiento al paradigma procedimental. La OO no es una novedad, se emplea desde el año 1967 [16,18,37,40] en áreas de simulación y gráficos. En los sistemas de gestión, comunicaciones, etc., se introdujo 20 años después, como una vía para aumentar la eficiencia en la gestión del software, cuando éstos elevaron su complejidad, antes no tenía justificación.

La enseñanza de OO en segundo curso permite abordar problemas más complejos, en los que se pueda tomar mayor ventaja, y por tanto presentar mejor sus principales aportaciones: (reutilización de código, jerarquización, encapsulación y facilidad de la gestión del *software*).

Las ventajas de este enfoque son la exposición escalonada y paulatina de los conceptos de la programación. Pero muchas

experiencias relatan la dificultad que presentan los alumnos al cambiar de paradigma, lo que ha venido denominándose el "problema del desplazamiento" [18,40]. Otros, consideran como desventaja de esta ordenación temporal el retraso en el desarrollo de aplicaciones motivadoras.

Este enfoque se articula de dos formas distintas, que evaluamos a continuación.

3.1. Dos lenguajes para dos paradigmas

Estas propuestas defienden la exposición tradicional [15,23,26]: primero programación estructurada con un lenguaje y luego programación OO con otro lenguaje.

La ventaja es que los conceptos se pueden introducir de forma gradual e independiente, pero el problema del "desplazamiento de paradigma" se muestra en toda su amplitud.

El cambio sintáctico/semántico entre ambos lenguajes, distrae al profesor en la exposición y al alumno en el aprendizaje de nuevos conceptos que se apoyan en otros asumidos, pero presentados con una nueva cara. Hay que parar constantemente para explicar unos nuevos tipos primitivos, una nueva tabla de precedencias de operadores, parecidas estructuras de control de flujo de ejecución, etc.

3.2. Un lenguaje OO para los dos paradigmas

Para mitigar el problema del desplazamiento, esta línea comienza con el paradigma convencional, empleando un lenguaje OO [39,48,53].

La ventaja de usar el mismo lenguaje para las dos partes es mantener una uniformidad y evitar la exposición de los aspectos léxicos, sintácticos y semánticos de dos lenguajes diferentes.

Sin embargo, presenta graves inconvenientes: el tratamiento de tipos, variables, expresiones y sentencias tiene que ubicarse en algún sitio; sin duda en una clase. Es imprescindible empezar con ejemplos sencillos (quizás la gestión de un contador, o dos alternativas anidadas para determinar el tipo de triángulo formado por tres vértices fijados) ¿cómo se llaman las clases?, ¿dónde están los objetos?, ¿quién lanza los mensajes? La resolución de este tipo de ejercicios desvirtúa conceptos muy complejos que no tienen cabida en los ejemplos iniciales, necesariamente sencillos.

Emplear un lenguaje que no está orientado al paradigma explicado genera confusión y hábitos inadecuados en el alumno. Algunos autores advierten a sus alumnos que "esto, realmente, no se resuelve como lo estamos haciendo" [48]. Nos parece una pauta pedagógica inadecuada.

El otro grave inconveniente radica en cuestiones psicológicas y sociológicas. El cambio de las palabras reservadas que organizan los módulos del software no perjudica, sino que facilita la transición de la programación estructurada a la OO. Si se desea cambiar la percepción social, por ejemplo, sobre el oficio de "barrendero", es adecuado cambiar su denominación, por ejemplo a "especialista en sanidad urbana".

Merecen una atención especial las propuestas docentes que emplean lenguajes híbridos (Ada/Ada9x, C/C++ o Pascal/Delphi) que podrían mitigar el impacto en la transición [3,39,59]. Nuestra experiencia demuestra que generan confusión en el alumno a la hora de ubicarse en uno u otro paradigma ya que el programa "compila" y "funciona" aunque no se respete el diseño adecuado al paradigma. En [26,29,31,34] se concluye que "el uso de un lenguaje OO puro para enseñar los conceptos OO es mejor e involucra menos conceptos que el empleo de un lenguaje híbrido". En lo tocante a procedimientos, declaración de variables y estructuras de control no hay inconveniente; por el contrario, beneficia, mantener la sintaxis.

En resumen, consideramos adecuado abordar los conceptos de OO cuando están consolidados los correspondientes al paradigma procedimental y abordar la docencia de cada paradigma con un lenguaje que se adecue a los conceptos que se están trasmitiendo.

4. Objetos en primero

La motivación general de estas propuestas viene de la mano del carácter "intuitivo" [49] y la facilidad del alumno para descubrir las clases del dominio [47].

Sorprendente la importancia dada al carácter intuitivo de la OO. Esto no es nuevo. Cada

lenguaje surge para hacer más intuitiva la expresión de determinados mecanismos mentales.

Negroponte afirma que "los sistemas informáticos son el mayor reto de complejidad al que se ha enfrentado el hombre en la historia" [46]. Esta aseveración resulta paradójica con la intuitiva sencillez prometida.

La falacia de la paradoja es sencilla: hablan de cosas diferentes. Una cosa es aprender a manejar un mundo de objetos simulados y otra es diseñar y programar una aplicación de calidad a través del enfoque OO. Son tareas de muy diferente índole, a realizar por un usuario y un ingeniero, respectivamente.

Meyer apunta [43]: "hay que huir de dar mucha importancia a la idea de que los sistemas OO se deducen directamente del *mundo real*... Aunque una vez que se entienden pueden parecer tan reales como cualquier otro, para un recién llegado pueden parecer menos *naturales* que los conceptos utilizados en las soluciones orientadas a procesos".

El programa OO resultante no es tan intuitivo como se pretende. Pensemos en la clase *Fecha* con su método *esNavidad*, o *Tablero* con el método *ponerFicha* en una *Coordenada*... ¿desde cuando una fecha informa sí es el día de Navidad?, ¿es "intuitivo" un tablero con manos que coloca fichas donde le dice un objeto coordenada?, ¿coordenada es un objeto? En "el mundo real" las acciones (*esNavidad* o *ponerFicha*), las controla el hombre, no los objetos (de la clase *Fecha* o *Tablero*).

Cuando se hace referencia carácter "cercano al pensamiento cotidiano" de la OO, se está hablando de localizar entes y asociarlos con sus operaciones plausibles, advirtiendo, por ejemplo que no se botan vasos, ni se beben pelotas. De forma similar, en OO se determina qué operaciones tienen cabida para cierto conjunto de datos y, para organizar el software, se usan los mecanismos habituales para estructurar el conocimiento: los modelos, la abstracción, los enlaces, la ordenación en el tiempo/espacio... Solo en este sentido es intuitiva.

En la programación OO hay que encontrar un modelo eficiente, tanto en ejecución e implementación, como en la gestión y posterior mantenimiento del software, que puede no corresponder a la visión informal del mundo.

"Es útil remarcar que los primeros libros sobre OO enfatizaban lo fácil que era identificar objetos mientras que los últimos, a menudo de los mismos autores, enfatizan su dificultad" [19]. "Experiencias con principiantes en OO muestran que tienen dificultades en crear clases adecuadas" [14,19].

En conclusión, un principiante no está en mejor disposición para aprender el paradigma OO que otro paradigma [60].

Existen diversas concreciones para el enfoque de OO en primero, que hemos clasificado como: posponiendo o anteponiendo la programación.

4.1 Posponiendo la programación

Suelen implementarse mediante entornos gráficos de universos simulados de objetos, o bien mediante análisis, diseño y, posteriormente, programación OO.

Las del primer grupo proponen entornos interactivos (la charca virtual, la cesta de la compra...) repletos de objetos susceptibles de recibir mensajes [30,49,61,62], en los que el alumno describirá la secuencia de mensajes lanzados sobre objetos para conseguir cierto objetivo propuesto, aprovechando el supuesto carácter "intuitivo" del paradigma. Ya hemos rebatido la validez de esta hipótesis.

El empleo de herramientas interactivas para la presentación motiva al alumno, pero conlleva una sobresimplificación que genera expectativas irreales. Puede que le enseñe a comprender sus propios mecanismos de adquisición de conocimiento, pero no a programar mejor. No es un enfoque válido para abordar la docencia de primer curso.

Las propuestas basadas en análisis y diseño, mantienen que lo fundamental es entender el análisis y diseño de la parcela del mundo real a programar, no siendo necesario, en principio, recurrir a un lenguaje concreto. Las propuestas giran en torno al uso de UML o parecidos [32,47]. Algunos abordan este enfoque para la enseñanza de la OO, aunque la ubican en segundo curso [36,45].

La ventaja es la aproximación paulatina a los conceptos. Pero el argumento en contra es rotundo: el lenguaje determina el pensamiento. Si no se conoce el lenguaje no se puede ni diseñar ni analizar. Hay citas categóricas contrarias al enfoque de presentar la filosofía de un paradigma de forma disociada del desarrollo de código: "enseñar a programar a través de buenos diseños,

enseñar a diseñar a través de buenos patrones y, por último, enseñar a analizar" [T.Love]; "para dominar el análisis OO, es preciso haber aprendido los conceptos fundamentales desde el nivel de la implantación [...] Únicamente después de un encuentro, con las *manos en la masa*, con el uso operativo del método, estará uno preparado para comprender los conceptos del análisis OO" [43].

El informe de ACM/IEEE2001 previene sobre los efectos de trabajar sin un lenguaje de programación y llama la atención sobre el mayor esfuerzo requerido por parte del profesor.

Por tanto, pensamos que no se puede empezar con modelos, ni con análisis primero, para evitar la complejidad de la programación, sino, que conviene introducirla escalonadamente, reforzando cada concepto teórico con su correspondiente implantación concreta sobre un lenguaje ejecutable.

4.2 Anteponiendo la programación

Este enfoque se apoya en la siguiente tesis: ¿conoce usted algo "mejor" que la OO?, ¿no?, ¿por qué enseña algo que le parece "peor"? [16,18,35,40,54].

Habría otras alternativas, por ejemplo empezar con programación concurrente OO con interfaces gráficos, o usar lenguajes multiparadigmáticos como Leda [13] (lógico/funcionales/procedimentales orientados a procesos y a objetos) y resolver cada parte del dominio del problema con el paradigma más adecuado. Se podría empezar con XML y, desechar definitivamente el lastre de los datos formateados. Pero, ¿Tengo que enseñar "el mejor" paradigma?... ¿cuál será el mejor mañana? El objetivo de la docencia es prepararles para que puedan evolucionar en consonancia con la evolución del software y según las necesidades de su propio ejercicio profesional.

Algunos de estos trabajos plantean el aprendizaje a partir del estudio y adaptación de buenos programas diseñados por otros, que servirán como *patrone*s o modelos de nuevos programas [48,58]. En el entorno OO, muchos patrones interesantes conllevan complejas relaciones entre clases, que producen una sobrecarga, no sólo de conceptos, sino de diseño.

Otras propuestas conjugan la exposición de los conceptos OO con el uso de bibliotecas (*API* y *frameworks*) [41], que fomentan la abstracción, encapsulación, modularización, y jerarquización. No parece viable utilizar las clases de la biblioteca sin conocer el paso de parámetros por valor o referencia de los mensajes, sin conocer las estructuras de control de flujo de ejecución que ordenen los mensajes en el tiempo y aprender todo a la vez, es en una sobrecarga excesiva para el alumno.

Otras propuestas que anteponen la programación presentan un acercamiento a la OO desarrollando aplicaciones con interfaces gráficos de usuario [26,33,50,51,55,63,64], o basando el curso en la creación de interfaces coloridas y juegos. La solución se articula a través de *applets* o con *frameworks* simplificados e irreales.

Sus desventajas son la suma de las de otras propuestas. Respecto al temario, es preciso introducir simultáneamente: tipos, variables, expresiones, sentencias de control añadiendo clase, herencia, método y mensaje. Si es duro aprender programación estructurada en un curso, la introducción de estos nuevos conceptos produce una sobrecarga inabordable [26].

Respecto a mecanismos de diseño: también sobrecargan con relaciones de herencia, composición y asociación, impuestas desde el primer momento, para terminar construyendo un *applet,* que es el peor ejemplo de clase, del estilo *Conversor de Celsius a Fahrenheit* o similares; ¿no será la clase *GradoTemperatura* quién asume la responsabilidad de la conversión, donde el interfaz no calcula, sino que requiere y presenta los datos?

Respecto a la estructura de la interfaz, manejan un modelo de eventos globalizado o una arquitectura sin eventos, con comunicaciones por cadenas de caracteres entre todos los componentes del modelo/vista/controlador. Estos modelos no sirven como antesala de las bibliotecas industriales y no tienen envergadura para resolver problemas reales.

Por tanto, compartimos que "adoptar el paradigma OO en primero implica (al menos, tal como se ha llevado a cabo en muchos libros) exponer a los estudiantes a conceptos muy complejos y, a menudo delicados, antes de que dispongan de conocimientos adecuados para comprenderlos" [12]. Los, sin duda, loables objetivos de fomentar la reusabilidad y de desarrollar potentes aplicaciones desde el

principio no son justificación para empezar con OO; ni la reusabilidad, ni las aplicaciones motivadoras, con o sin interfaz gráfico de usuario, son patrimonio de la OO.

Además ACM/IEEE-2001 no considera la interfaz gráfica y las librerías, ni como objetivos, ni como unidades temáticas a considerar en los cursos de introducción en ninguno de sus 4 diseños curriculares (primero con o sin objetos, distribuido en 2 o 3 cursos). Ubicando, específicamente, estos contenidos en cursos intermedios de desarrollo del *software* o en computación gráfica. La *figura 2* presenta la dedicación porcentual en la propuesta curricular para API y GUI en los cursos introductorios y en los intermedios específicos de la materia.

		Horas Totales	APIs %	GUIs %
Primero Imperativa	2 cursos	80	0	0
	3cursos	120	2	2
Primero Objetos	2 cursos	80	5	2
	3 cursos	120	2	1
CS260T. Software Development	INTERMEDIOS	40	8	53
CS262. Software Development and Professional Para.		40	8	23
CS261S. Software Development and Systems Prog.		40	8	28
CS255. Computer Graphics and Multimedia		40	8	80

Figura 2.Ubicación de la docencia de API y GUI en la propuesta de ACM/IEEE.

5. Conclusiones

Hemos presentado una clasificación de los enfoques de la enseñanza de la OO propuestas por diferentes autores. Atendiendo a la viabilidad pedagógica, además de a la coherencia técnica, todas las propuestas estudiadas presentan algún inconveniente. Desafortunadamente, se han desarrollado pocos experimentos para medir la eficiencia pedagógica de los distintos enfoques.

Entendemos que la discusión sobre la planificación de los primeros cursos de programación no debe centrarse en el paradigma a presentar, ni en el lenguaje más adecuado para la implementación de los ejercicios, sino en los objetivos perseguidos y el modo de conseguirlos.

En concreto, el "problema del desplazamiento" surge como consecuencia del modo en que se han impartido los conceptos del paradigma procedimental, no por haber comprendido el concepto de *TAD* antes de abordar el de *clase*.

Urge disponer de una propuesta que resuelva los problemas planteados por las actuales, que permita la unificación de los planes de estudio, facilitando la movilidad de los estudiantes.

Por otro lado, nosotros compartimos que: "las teorías científicas son entidades que se extienden y perduran en el tiempo" [20]; "la OO no supone una revolución, sino, una evolución" [8,57] y que "la OO potencia y fortalece los conceptos que admiramos en la programación procedimental" [37]. De hecho, ACM/IEEE91 [1] detecta una serie de conceptos recurrentes "que ayudan a unificar una disciplina, se presentan reiteradamente en diversos contextos, son independientes de la tecnología concreta y se extienden a través de la historia".

En base a estas consideraciones hemos desarrollado una alternativa en la que los conceptos recurrentes son la clave que conduce la transición a través de los conceptos de la programación, "independientemente del paradigma concreto y a través de su evolución histórica", evitando la sobrecarga inicial que conlleva un enfoque OO en primero y eliminando el problema del desplazamiento de paradigma que aparece en las propuestas que comienzan por procedimental, al aportar la necesaria continuidad.

La descripción de dicha propuesta excede los objetivos del artículo y se desarrolla en otra ponencia de estas Jornadas [22].

Agradecimientos

Este trabajo se ha financiado con el proyecto TIC2000-1413 de la CICYT.

Referencias

[1]ACM/IEEE. *Computing curricula 1991*. http://www.computer.org/education/cc1991/eab1.html

[2]ACM/IEEE. *Computing curricula 2001*. http://www.acm.org/sigcse/cc2001/15-12-2001.html

[3]Adams, J.C. *Object centrered design a five-phase introduction to OO programming.* SIGCSE 1996, 78-82

[4]Albert, E. et al. *La enseñanza de Java en los estudios de informática.* JENUI 2000, 557-62

[5]Angster E. *A simple OO system pattern to introduce OOP with design*. OOPSALA'96 Workshop 1996

[6]Angster, E. *Our OO teaching concepts.* ECOOP 98

[7]Barr, M. et al. *An Exploration of novice programming errors in an OO environment.* SIGCSE 1999, 31(4) 42-6

[8]Bishop J. *Java gentile for engineers and scientists.* Addison-Wesley 2000

[9]Bisshop-Clark C.; Kiper J. *An undergraduate course in Object-Oriented software design.* 28 thFIE'98. Conference proceedings. IEEE, Piscataway.NJ,USA, (1) 1998pp 38-42

[10]Böszörményi, L. *Why Java is not my favorite first-course language software*. Concepts & Tools 1998, 19 141-145pp

[11]Brilliant, S.S.; Wiseman, T.R. *The first programming paradigm and language dilemma.* SIGCSE 1996, 338-42

[12]Buck, D.; Stucki, D. *Design early considered harmful: graduate exposure to complexity and structure based on levels of congnitive development.* SIGCSE 2000 3/00

[13]Budd, T. *Multiparadigm programming in Leda.* Addison-Wesley 1994

[14]Carter, J. et al. *Object Oriented students?* ITiCSE 1998, 271

[15]Casanova A. et al. *La enseñanza de la programación en los estudios de informática de la UPV* . JENUI 1998, 411-15

[16]Deckerr, R.; Hishfield, S. *The top 10 reasons why OO programming can't be taught in CS1*. SIGCSE94 (27) 51-4

[17]Declaración conjunta de los Ministros Europeos de Educación. Bolonia 19 de junio de 1999. www.universia.es/contenidos/universidades/documentos/universidades_docum_bolonia.htm

[18]DeClue, T. *OO and the principles of learning theory a new look at problems and benefits.* SIGCSE 1996, 232-6

[19]Détienne, F. Détienne, F. *Software-desing: cognitive aspects*. Springer 2002

[20]Díez, J.A.; Moulines, C.U. *Fundamentos de la filosofía de la ciencia.* Ariel, 1997

[21]Esteban A. et al. *La programación basada en Component Pascal: de la programación OO a los componentes del software.* JENUI 1998, 423-26

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

[23]Fernández, A.; Rossi, G. *An effective approach to learning object-oriented technology.* ECOOP 98.

[24]Franch, X. et al. *Transición de Modula2 a Java en un curso de iniciación a la programación.* JENUI 1998, 319-326

[25]Franch, X.; et al. *An introductory programming pilot course using Java* EATCS 1999. pp 69

[26]García Molina, J. *¿Es conveniente la OO en primer curso de programación.* Novática 154 (11/12) 2001, 64-8

[27]Hardgrave, B.C.; Douglas, D.E. *Trends in information systems curricula: object-oriented topics.* Journal of Computer Information System vol 39, nº1, 1-6.

[28]Hitz, M.; Hudec, M. *Modula2 versus C++ as a first programming language. Some empirical results*. SIGCSE 1995, 317-21

[29]Hosch F. *Java as a First language: an evaluation.* SIGCSE Bulleting 28(3) 1996 45-50

[30]Houle, M.E. *Ethics, programming, and virtual environments.* ITiCSE 1997, 91-3

[31]Joyner, I. Objects u*nencapsulated: Java, Eiffel and C++*. Prentice Hall, 1999

[32]Kelemen B. *A Newcomer's Thoughts about responsibility distribution.* ECOOP'98.

Lecture Notes in Computer Science, 1543 Springer-Verlag 1999.

[33]Koffman E. Wolz U. *A simple Java package for GUI-like interactivity*. SIGCSE 2001, 11-15

[34]Kölling, M. et al. *Requirements for a first year OO teaching language.* SIGCSE 1995, 173-77

[35]Kölling, M. *The problem of teaching object-oriented programming. Part I: Languages* Journal of Object-Oriented Programming 1999, 11(8) 8-15pp

[36]Kornecki A. J. *Teaching object-oriented simulation in a software engineering framework. Simulation*. (abril 2001) 233-239

[37]Lewis, J. *Myths about OO and its pedagogy.* SIGCSE 2000, 245-49

[38]Lidtke, D.K.; Zhou, H.H. *A New approach to an introduction to computer science.* ASEE/IEEE 29th FIE'99 4-23, 1999

[39]Llopis, F.; Pérez, E. *C++-OO como lenguaje introductorio a la programación*. JENUI 2001, 299-304

[40]Lucker *There's more to OOP than syntax!.* SIGCSE Bulletin 26 (1), p 56-60 (1994)

[41]McCracken, D.D. *An inductive approach to teaching object-oriented design.* SIGCSE 1999 3/24-28

[42]McDonald, C. *1st year programming languages in australian and New Zealand universities*. http:// www.cs.uwa.edu.au/~chris /java-in-cs1/anzacs.html

[43]Meyer, B. *Object-Oriented software construction, 2ªed.* Prentice-Hall, 2000 ISBN0136291554

[44]Meyer, B. *Towards an OO curriculum. Journal OO Programming* 1994, (3)76-81

[45]Moreira A. *Teaching objects: the case for modelling*. ECOOP'98 (1998), Lecture Notes in Computer Science, 1543 pp350-354 Springer-Verlag 1999

[46]Negroponte N. *Ser digital* (1995) Atlantida Publishing; ISBN: 9500814730

[47]Oliveira, C.A. *Aspects on teaching/learning with OO programming for entry level Courses of engineering.* ICEE 1998, (8)17-20

[48]Ortega, A. et al. *Programación multilenguaje con Component Pascal y Java en un primer curso de programación.* JENUI 1999, 343-48

[49]Osborne M., Johnson J. *An only undergraduate course in object-oriented technology.* SIGCSE Bulletin. 25(1) 101-106 (1993).

[50]Rasala, R. et al. *Java Power Tools: Model software for teaching OO design*. SIGCSE 2001, 297-301

[51]Rasala, R. *Toolkits in first year computer science: a pedagogical imperative.* SIGCSE 2000, 185-91

[52]Rayside, D.; Campbell, G. *Aristotle and OO programming: why modern students need traditional logic.* SIGCSE 2000, 237-44

[53]Reges, S. *Conservatively radical Java in CS1*. SIGCSE 2000 3/00

[54]Roberts, E. *An Overview of MiniJava.* SIGCSE 2001 2/01

[55]Schaub, S. *Teaching Java Graphics in CS1*. SIGCSE 2000 32(2)71-3

[56]Schoenefeld, D.A. *OO Design and Programming: an Eiffel, C++, and Java Course for C Programmers*. SIGCSE 1997, 135-9

[57]Stroustrup B. *The Design and Evolution of C++*. Addison-Wesley, 1994. ISBN 021543303

[58]Wallingford, E. *Toward a first course based on OO patterns.* SIGCSE 1996, 27-32

[59]Wick, M.R. *On Using C++ and OO in CS1: the Message is still more important than the Medium* SIGCSE 1995, 322-6

[60]Wiedenbeck, S.; Ramallingam, V. *Novice Comprhension of Small Programs Writen in the Procedural and OO Styles.* International Journal of Human Computer Studies (1999) 51 71-87

[61]Woodman, H.; Robinson, H.; Griffiths, R.; Holland, S. *An Object Oriented Approach to Computing* OOPSLA 98

[62]Woodman, M. et al. *The Objects Shop-Using CD-ROM Multimedia to Introduce Object Concepts*. SIGCSE 1997, 345-9

[63]Woodman, M.; Holland, S. *From software user to software author: an initial pedagogy for introductory OO computing.* ITiCSE 1996, 60-2

[64]Woodworth P.;Dann W. *Integrating Console and event-driven Models in CS1*. SIGCSE 1999 (3)132-135