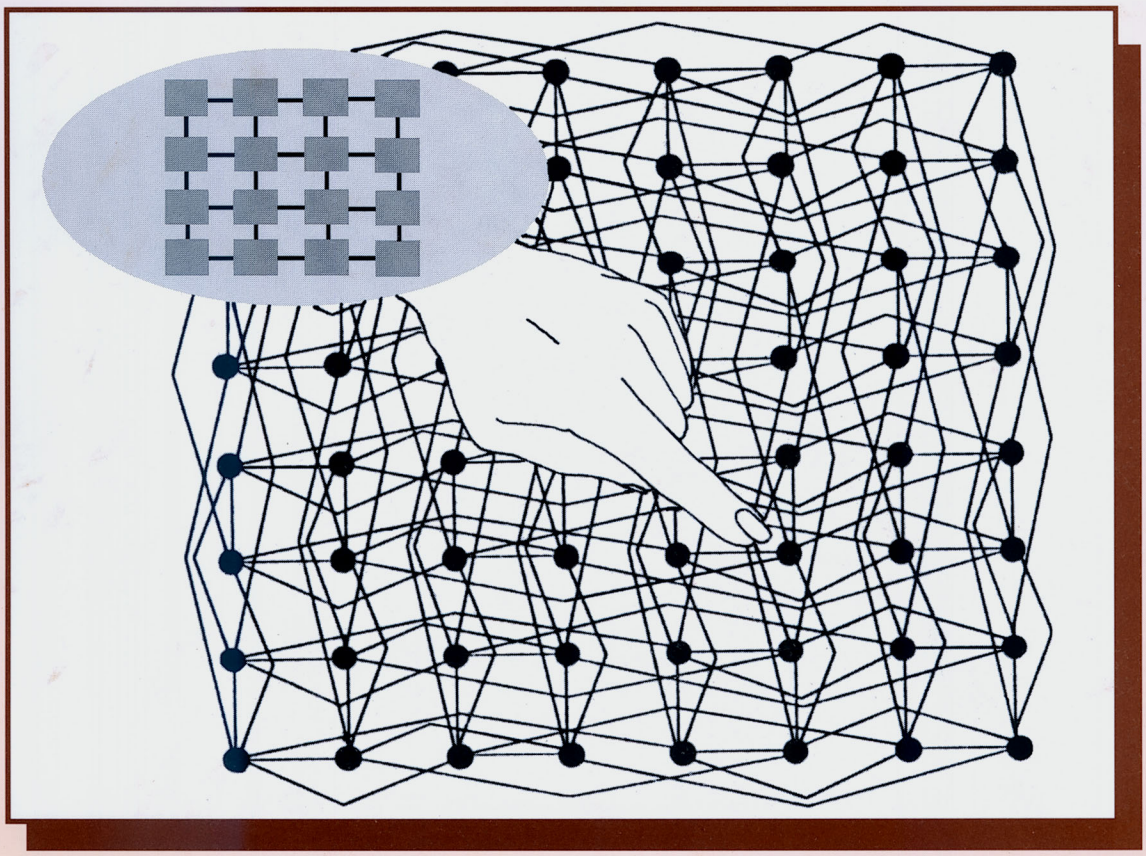
Ingeniería

Revista de la Universidad de Costa Rica
Julio/Diciembre 1996 VOLUMEN 6 Nº 2



INGENIERIA

Revista Semestral de la Universidad de Costa Rica
Volumen 6, Julio/Diciembre 1996 Número 2

CORRESPONDENCIA Y SUSCRIPCIONES

Editorial de la Universidad de Costa Rica
Apartado Postal 75
2060 Ciudad Universitaria Rodrigo Facio
San José, Costa Rica

CANJES

Universidad de Costa Rica
Sistema de Bibliotecas, Documentación e Información
Unidad de Selección y Aquisiciones-CANJE
Ciudad Universitaria Rodrigo Facio
San José, Costa Rica

Suscripción anual:
Costa Rica: ¢ 1 000,00
Otros países: US $ 25,00

Número suelto:
Costa Rica: ¢ 750,00
Otros países: $ 15,00

Edición aprobada por la Comisión Editorial de la Universidad de Costa Rica

Ciudad Universitaria Rodrigo Facio
San José, Costa Rica.

Revisión Filológica: *Lorena Rodríguez*

Diagramación:
José R. Argüello V.

Control de Calidad:
Unidad Diseño Revistas. Oficina de Publicaciones

Impreso en la Oficina de Publicaciones
de la Universidad de Costa Rica

Revista
620.005
I-46i Ingeniería / Universidad de Costa Rica. ——
Vol. I, no. 1 (ene./jun. 1991)— . —— San José, C. R. : Editorial de la Universidad de Costa Rica, 1991— (Oficina de Publicaciones de la Universidad de Costa Rica)
v. : il

Semestral.

1. Ingeniería - Publicaciones periódicas.

CCC/BUCR—250

Ingeniería 6 (2): 71-78 1996 San José, Costa Rica

¿ES UN AMBIENTE DE PROGRAMACIÓN ORIENTADO A OBJETOS?

Vladimir Lara V.[1]
Maureen Murillo R.[2]

Resumen

Cuando se plantea el tema de si determinada herramienta de desarrollo o lenguaje de programación es un ambiente orientado a objetos es necesario aclarar a qué se hace referencia con el concepto *ambiente orientado a objetos*. Este artículo es producto de la investigación hecha sobre el alcance del término *ambiente orientado a objetos* en el contexto del ciclo de desarrollo de una aplicación. La visión personal presentada es un aporte a la clarificación de un tema que, por su popularidad, amerita ser tratado de manera consistente.

Summary

When you inquire if a programming tool or language is an object oriented environment, it is imperative to clarify the meaning of that concept. This paper is the result of investigating the meaning of the term *object oriented programming environment*, in the context of the software developmental process. The personal approach presented in this paper is a contribution to the clarification of a subject that, due to its popularity, needs to be treated consistently.

1.- AMBIENTE DE PROGRAMACION EN EL CICLO DE DESARROLLO

El desarrollo de una aplicación está inmerso en un ciclo compuesto por varias etapas. En cada una de estas fases se utilizan diferentes técnicas y herramientas, las cuales pueden ser orientadas a objetos o de otro tipo de paradigma de desarrollo, determinando así la orientación a objetos de cada fase. Con base en la relación que tengan las etapas y en su orientación a objetos, un ciclo de desarrollo puede ser parcial, totalmente o en ningún grado orientado a objetos.

Para ubicar dentro de este ciclo de desarrollo de un proyecto la utilización del ambiente de programación, se tomará como base el proceso de desarrollo propuesto por Grady Booch dirigido a proyectos orientados a objetos.

Este ciclo propuesto por Booch está compuesto por cinco fases [Booch, 1996], tal como se muestra en la figura 1:

Conceptualización: establecimiento de los requerimientos principales.

Análisis: desarrollo de un modelo del comportamiento deseado del sistema.

Diseño: creación de una arquitectura para la solución.

Evolución: desarrollo de la solución a través de refinamiento sucesivo.

Mantenimiento: administración de la evolución posterior a la entrega del producto.

[1] Ph.D, prof. Esc. Ciencias de la Computación e Informática, Fac. Ingeniería, UCR
[2] Licda., prof. Esc. Ciencias de la Computación e Informática, Fac. Ingeniería, UCR.

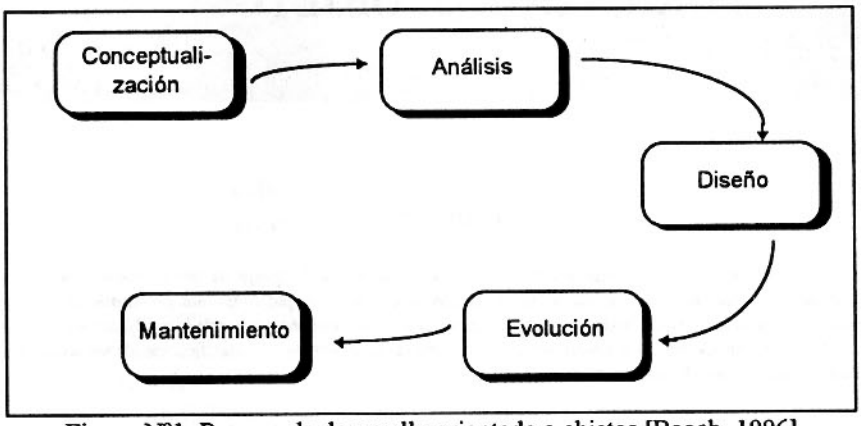


Figura N°1: Proceso de desarrollo orientado a objetos [Booch, 1996]

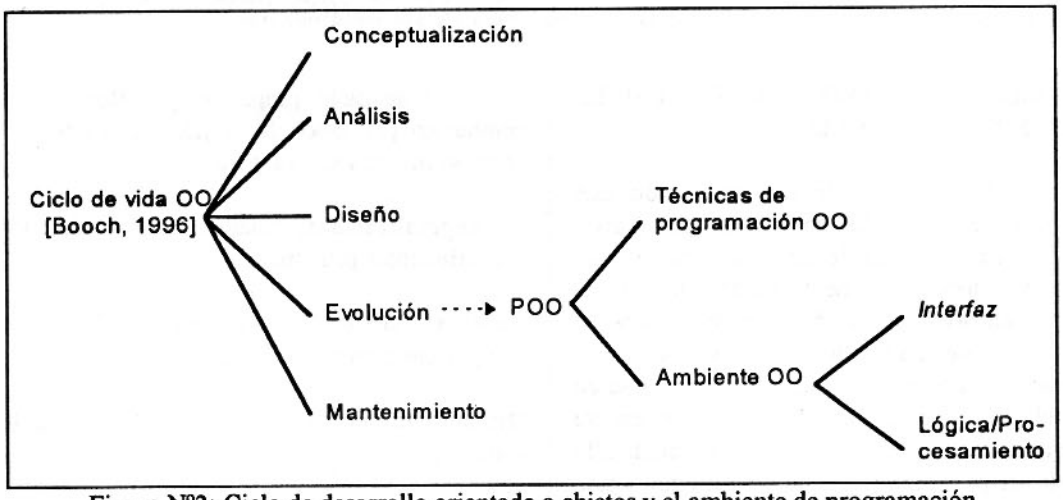


Figura N°2: Ciclo de desarrollo orientado a objetos y el ambiente de programación

En cada etapa[3] se pueden emplear técnicas y herramientas orientadas a objetos. Particularmente, en la fase de evolución se aplican los conceptos de la programación orientada a objetos (POO). La POO requiere la utilización de dos tipos de herramientas: 1) técnicas o buenas prácticas de programación orientadas a objetos tales como la reusabilidad de código y el ocultamiento de información, y 2) un ambiente orientado a objetos que permita aplicar esas técnicas.

El ambiente utilizado es de especial interés en este artículo, el cual puede ofrecer facilidades de orientación a objetos en la

[3] Las etapas básicas de este proceso propuesto por Booch son similares a las de otros ciclos de desarrollo y se toman sólo como un ejemplo, razón por la cual lo expuesto en este documento puede ser generalizado a otros ciclos de vida.

construcción de dos de los componentes principales de una aplicación: su *interfaz* y su lógica de procesamiento. Un ambiente de programación puede permitir el desarrollo orientado a objetos para sólo uno de los componentes o para ambos, constituyéndose así en un ambiente orientado a objetos sólo en cuanto a *interfaz*, o sólo en relación con la construcción de procesos, o en ambos aspectos. En la figura 2 se muestra esquemáticamente la ubicación del ambiente de programación dentro del ciclo de desarrollo de una aplicación.

Habiendo identificado la fase dentro del ciclo de desarrollo donde cobra importancia la utilización de un ambiente de programación, se puede evaluar con más claridad una herramienta de programación particular en cuanto a los aspectos de orientación a objetos que atañen al ambiente, evitando confundirlos con los que son propios de otras fases (por ejemplo, un buen diseño orientado a objetos) o que dependen del desarrollador (tal como procurar producir código reusable).

En la siguiente sección se interpreta la orientación a objetos y los términos asociados al concepto, sobre los cuales está basada la evaluación de un ambiente.

2.- VISION DE ORIENTACION A OBJETOS

La definición de un lenguaje o ambiente orientado a objetos varía de un autor a otro y de una tendencia a otra (refiérase a [Nelson, 1991] para un análisis más detallado). En este artículo se hace una definición particular de los elementos que están asociados al concepto de orientación a objetos y se clasifican, según su función, como características determinantes de un ambiente. Esta clasificación comprende las características básicas de un lenguaje para que sea considerado como orientado a objetos, las características asociadas a tal lenguaje y las facilidades que ofrece un ambiente para la manipulación de los objetos.

2.1.- Características básicas de un lenguaje orientado a objetos

Alrededor del concepto de orientación a objetos giran una variedad de términos relacionados tales como abstracción de datos, encapsulación, clases, objetos, herencia, polimorfismo, chequeo de tipos, manejo de memoria, etc.

Sin embargo, los elementos básicos que un lenguaje debe poseer para ser orientado a objetos son: *clases*, *herencia* y *polimorfismo*. Esta es una definición particular, pues existen otros autores que hacen una definición diferente. Por ejemplo, Nierstrasz opina que cualquier lenguaje de programación que provee mecanismos utilizables para explotar la encapsulación es (al menos en algún grado) orientado a objetos [Nierstrasz]. Por otro lado, Wegner define un lenguaje orientado a objetos como aquel que soporta objetos, clases y herencia [Wegner, 1987].

A continuación se definen los elementos básicos mencionados (clases, herencia y polimorfismo) que sirven como base para determinar si un lenguaje de programación es un ambiente orientado a objetos.

2.1.1.- Clases

Una clase es una descripción de objetos similares, donde un objeto es un conjunto de variables o propiedades con un comportamiento asociado representado mediante un conjunto de métodos o procedimientos que manipulan las variables.

Al hablar de clases se hace referencia a dos conceptos fundamentales en el paradigma de orientación a objetos: *abstracción de datos* y *encapsulación*. La abstracción de datos es la capacidad de aislar los detalles de implantación de un objeto, permitiendo que el objeto sea manipulado sólo a través de sus operaciones o métodos definidos. Por otra parte, la

encapsulación es la capacidad de ligar las propiedades y los métodos formando una sola entidad, específicamente, una clase. La definición de clases permite tanto la abstracción de datos como la encapsulación.

2.1.2.- Herencia

La herencia es un mecanismo para reutilizar código. Esto se logra creando una clase basada en otra, en donde la nueva clase (subclase) hereda automáticamente las propiedades y los métodos definidos en la clase de la cual se está derivando (superclase).

Existen dos tipos principales de herencia: simple y múltiple. La herencia simple se lleva a cabo cuando se crea una clase basada en una sola superclase, mientras que la herencia múltiple es aquella clase basada en más de una superclase. Se considera que la herencia múltiple no es necesaria en un lenguaje de programación para que éste sea considerado como orientado a objetos.

2.1.3.- Polimorfismo

Polimorfismo es la capacidad de llamar a un método, o a una propiedad de una clase, de igual forma que un método o propiedad de otra. Existen dos tipos principales de polimorfismo: de inclusión y de operación [Blair, Gallagher y Malik, 1989].

El polimorfismo de inclusión es el que se realiza entre clases que están relacionadas entre sí. El polimorfismo de operación es el que se realiza entre clases que no están relacionadas y puede tomar dos formas: por sobrecarga y en forma paramétrica. La sobrecarga se da cuando el nombre de un método representa diferentes comportamientos para clases distintas. El polimorfismo paramétrico se realiza cuando el nombre de un método representa un mismo comportamiento pero para clases distintas.

Un lenguaje orientado a objetos debe proveer tanto el polimorfismo de inclusión como el polimorfismo de operación, ya sea por sobrecarga o en forma paramétrica.

2.2.- Características asociadas a un lenguaje orientado a objetos

Existen otros aspectos de un lenguaje que están relacionados con la orientación a objetos, especialmente con el tipo de manipulación que hace de ellos el lenguaje de programación, tales como el chequeo de tipos, el tiempo de liga de métodos (*binding*), el manejo de memoria y la uniformidad del lenguaje. Los lenguajes enfocan en forma diferente estas funciones, las cuales no son determinantes para la definición de un lenguaje orientado a objetos.

2.2.1.- Chequeo de tipos

El chequeo de tipos se refiere a la asociación entre identificadores (nombres), tipos (clases) e instancias (objetos). Un lenguaje puede tener chequeo de tipos estático o chequeo de tipos dinámico.

En el chequeo de tipos estático los identificadores se asocian a los tipos o clases mediante declaraciones explícitas. Los lenguajes que garantizan (en tiempo de compilación) que todas las expresiones tienen el tipo correcto se les conoce como poseedores de tipos fuertes.

En el chequeo de tipos dinámico los identificadores no se asocian a clases, sino que se asocian a los objetos. Esto quiere decir que, en el tiempo de ejecución (que es cuando se hacen las asignaciones a los identificadores), es cuando un identificador asume la clase del objeto que se le está asignando.

2.2.2.- Liga de métodos

La liga de métodos se refiere al momento y a la forma en que un mensaje es asociado con el método respectivo. Existe la liga de métodos estática y la dinámica.

La liga de métodos estática se realiza en tiempo de compilación. La asociación entre el mensaje y el método se hace con base en las características estáticas del objeto al que se le envía el mensaje.

La liga de métodos dinámica se lleva a cabo en tiempo de ejecución, determinándose el método que le corresponde al mensaje con base en la clase actual del objeto al que representa el identificador y no con base en el tipo con que se declaró el objeto.

2.2.3.- Manejo de memoria

Los lenguajes de programación proveen formas para recuperar el almacenamiento en la memoria asignado a los objetos y a las variables. Esta recuperación puede realizarse en forma manual, en donde el programador libera explícitamente el espacio llamando alguna rutina, o puede realizarse en forma automática (llamada recolección de basura), en donde el lenguaje de programación detecta automáticamente los objetos y las variables que ya no son necesarios y recupera la memoria asignada.

2.2.4.- Uniformidad del lenguaje

Un lenguaje de programación es uniforme respecto de la orientación a objetos, si la programación debe realizarse necesariamente mediante la utilización de objetos. Los lenguajes que, a pesar de brindar las características de orientación a objetos, permiten obviar esta orientación y admiten otro tipo de programación (por ejemplo de procedimiento), son llamados lenguajes híbridos.

La conveniencia de un lenguaje uniforme o de un lenguaje híbrido depende de las necesidades y gustos del programador. Un lenguaje uniforme garantiza un código más sólido al utilizar las técnicas de orientación a objetos, mientras que un lenguaje híbrido ofrece flexibilidad y evita tener que programar en forma orientada a objetos, en aquellos casos en que el problema no amerite la formalidad de esta metodología.

2.3.- Facilidades complementarias de un ambiente de programación

Generalmente, los ambientes de programación poseen algunas características adicionales que facilitan la manipulación de las distintas herramientas y elementos del lenguaje. En esta categoría se ubican todos aquellos aspectos del ambiente que asisten al programador en la creación y en la utilización de los objetos, favoreciendo el desarrollo de una aplicación orientada a objetos.

2.3.1.- Interfaz visual

Como se mencionó anteriormente, un ambiente puede ofrecer facilidades de orientación a objetos en la construcción de *interfaces* y en la programación de los procesos de la aplicación.

En cuanto a la construcción de *interfaces*, independientemente de si el ambiente es orientado a objetos o no, la herramienta puede ser visual o no visual. Un ambiente puede ser totalmente orientado a objetos a pesar de que las herramientas para la construcción de *interfaces* no sean visuales. Igualmente, un ambiente puede no ser orientado a objetos a pesar de que posea una *interfaz* visual.

Un lenguaje visual manipula información visual, soporta interacción visual o permite programación con expresiones visuales [Baeza, 1997]. Existe toda un área de estudio en cuanto a los lenguajes visuales, los cuales deben cumplir con una serie de características y de especificaciones de diseño particulares. Sin embargo, en este artículo se hace un uso más flexible de la palabra *visual*, haciendo referencia a un lenguaje que utiliza una representación de este tipo (gráficos, íconos, dibujos) permitiéndole al programador observar la *interfaz* final de su aplicación mientras la construye.

Un lenguaje orientado a objetos puede manipular visualmente tanto la *interfaz* que se construye, como la definición de las clases que se utilizarán en una aplicación. Esta característica, al mostrar interactivamente lo que se construye, apoya la concepción o la visión orientada a objetos que debe poseer el programador para diseñar su aplicación.

2.3.2.- Sintaxis

En cuanto a la programación de los procesos y de la lógica del programa, el nivel de complejidad de la sintaxis del lenguaje es un factor determinante para que el programador realice una representación adecuada por medio del código de la concepción que posee de los objetos definidos y de su comportamiento. Además, favorece el rendimiento de los desarrolladores y la calidad de las aplicaciones resultantes.

2.3.3.- Biblioteca de clases

Una característica complementaria que debe considerarse en la evaluación de un ambiente orientado a objetos es la disponibilidad de bibliotecas de clases que el lenguaje de programación incorpora. Debe tomarse en cuenta la cantidad y la relevancia de las clases implantadas en esas bibliotecas y la posibilidad de que el programador extienda y construya nuevas bibliotecas de clases. Estas bibliotecas son la base para la reutilización de código en las aplicaciones.

2.3.4.- Depuración

Además de las facilidades propias de la etapa de construcción de los programas, deben considerarse las facilidades de un lenguaje para el apoyo de la fase de prueba de los programas. Lo mínimo que se espera que posea un ambiente es un depurador de programas, que además de permitir seguir paso por paso la ejecución del código y de examinar el contenido de la memoria, le ayude al programador a determinar qué tipo de chequeo de tipos y de liga de métodos se realizaron al ejecutar la aplicación.

2.3.5.- Tipo de ayuda

Como apoyo en la utilización del ambiente en general, y sobre todo en el buen uso de las características de orientación a objetos que posee, debe evaluarse el tipo de ayuda que provee el ambiente. Por ejemplo, si es ayuda en línea, si es sensitiva al contexto, si además de ser descriptiva orienta al programador en la forma en que debe hacer uso de los objetos para aprovechar sus ventajas y si es pertinente en la utilización de mensajes de advertencia y de error.

2.3.6.- Herramientas de desarrollo

Otra característica importante en un ambiente orientado a objetos es la disponibilidad de un administrador de proyectos, ya que generalmente un ambiente de este tipo implica la manipulación de muchos componentes (módulos, pantallas, reportes, programas, librerías de clases, bases de datos). El administrador de proyectos facilita la integración y modificación de estos elementos y, posiblemente, permita la creación de archivos que incluyan todos los elementos necesarios para la ejecución de la aplicación.

2.3.7.- Herramientas de comunicación

Tomando en cuenta que la etapa de evolución o programación está basada en las etapas previas de análisis y diseño del proceso de desarrollo del programa, debe considerarse la posibilidad de que el ambiente de programación brinde una *interfaz* con otras herramientas orientadas a objetos que apoyen las otras fases del ciclo de desarrollo de aplicaciones. Por ejemplo, sería de gran utilidad para el programador, con base en la definición de la jerarquía de clases realizada en la etapa de diseño con alguna herramienta computacional, que el ambiente de programación defina y

construya las librerías de clases que serán utilizadas al programar de la aplicación.

2.3.8.- Paradigmas complementarios

Por último, debe analizarse si un ambiente de programación posee características complementarias propias de otros paradigmas de desarrollo que podrían servir como apoyo en la programación por objetos. Un ejemplo de esto es la programación por eventos, en donde los métodos de una clase pueden constituirse en respuestas a eventos que sucedan durante la ejecución de una aplicación particular.

3.- CONCLUSION

Las características expuestas anteriormente definen toda una gama de diferentes grados de orientación a objetos que puede poseer un ambiente de programación. Lo esencial es tener claro el concepto asociado a cada característica y las ventajas y fortalezas con las que contribuye al lenguaje de programación que se analiza.

En este artículo se han presentado los elementos que deberían considerarse en el momento en que se evalúa si un determinado ambiente de programación es orientado a objetos o no. Queda a criterio del lector la forma en que conjugue los diferentes elementos para llevar a cabo el proceso de evaluación del ambiente.

Como se mencionó al inicio, la orientación a objetos es un tema de actualidad y los ambientes de programación han seguido esta dirección. En este sentido, la acreditación de un ambiente según los términos aquí definidos debe ser flexible y tomar en cuenta el proceso de evolución por el que están pasando.

Dada la falta de consenso en el ambiente informático sobre las definiciones aquí presentadas, se espera que este artículo ayude a clarificar el tema .

4.- BIBLIOGRAFIA

Baeza, Ricardo. Visual Programming. 1997. Fuente:http://www-lsi.upc.es/~rbaeza/sem/vfp/todo.html

Blair, Gordon S; Gallagher, John J. y Malik, Javad. Genericity vs Inheritance vs Delegation vs Conformance vs ... 1989.

Booch, Grady. Object Solutions: Managing the Object-Oriented Project. Estados Unidos: Addison-Wesley Publishing Company, Inc. 1996.

Budd, Timothy. An Introduction to Object-Oriented Programming. Estados Unidos: Addison-Wesley Publishing Company. 1991.

Cargill, Tom; Cook, William; Cox, Brad; Loomis, Mary; Snyder, Alan y Yen-Ping Shan. Is Multiple Inheritance Essential to OOP? (PANEL). Año desconocido.

Nelson, Michael L. An Object-Oriented Tower of Babel. California, Estados Unidos: Department of Computer Science, Naval Postgraduate School. 1991.

Nierstrasz, Oscar. A Survey of Object-Oriented Concepts. Año desconocido.

Snyder, Alan. Encapsulation and Inheritance in Object-Oriented Programming Languages. Palo Alto CA, Estados Unidos: Software Technology Laboratory, Hewlett-Packard Laboratories. 1986.

Wegner, Peter. Dimensions of Object-Based Languaje Design. Estados Unidos: OOPSLA. 1987.

Wirfs-Brock, Rebeca y Wilkerson, Brian. Object-Oriented Design: A Responsibility-Driven Approach. Estados Unidos: OOPSLA. 1989.