EVOLUCIÓN Y PERSPECTIVAS EN EL DESARROLLO DE SOFTWARE: NUEVAS TENDENCIAS ORIENTADAS A OBJETOS

María Dolores Lozano
Isidro Ramos Salavert

María Dolores Lozano es Profesora ayudante del Departamento de Informática. UCLM.

Isidro Ramos Salavert es Catedrático de la Universidad Politécnica de Valencia. Dpto. de Sistemas Informáticos y Computación.

RESUMEN

En este articulo se presentan los problemas existentes entorno al desarrollo de software, se da una visión general de su evolución y perspectivas y se presenta la nueva tendencia orientada a objetos. El paradigma orientado a objetos está siendo en los últimos años objeto de estudio de muchos equipos de investigación. Con el objetivo de desarrollar Sistemas de Información más cercanos al entorno real como alternativa fiable a las técnicas tradicionales, el modelo orientado a objetos se ha convertido en un método de análisis, diseño e implementación que cada vez está cobrando más importancia en lo que respecta la Ingeniería del Software, Lenguajes de Programación, Bases de Datos y Lenguajes de Especificación. Las ventajas que aporta este modelo impulsan a informatizar los Sistemas siguiendo esta nueva metodología, todavía en fase de crecimiento.

1. INTRODUCCIÓN

En el proceso de desarrollo de software, cuando analizamos sistemas[1], creamos modelos de las áreas de aplicación que nos interesan. Se podrían crear modelos enfocados a un área específica del sistema, o cubrir su totalidad. Modelizar la empresa es importante a la hora de plantear su informatización. Por ello, en una primera fase de análisis, se empieza modelizando áreas del Sistema para luego enlazarlas

(1) Por sistema se entiende la empresa u organización objeto de estudio que se desea informatizar.

todas juntas y así formar el modelo global [4]. El modelo representa un aspecto de la realidad y se construye de tal modo que nos ayude a entender dicha realidad. Además, es mucho más simple que ella. Manipular el modelo nos ayuda a ver de modo más sencillo como modificar, ampliar o rediseñar áreas de la misma.

Al plantear la informatización de un sistema, es importante modelizar dicho Sistema [4]. Este proceso además, se debe plantear por partes, ya que el Sistema engloba diversos subsistemas que deben abordarse de forma individual para un mejor y más claro entendimiento del mismo. Los modelos de dichos subsistemas ayudan a entender más fácilmente su funcionamiento.

La siguiente figura ilustra el proceso que se sigue para construir sistemas:

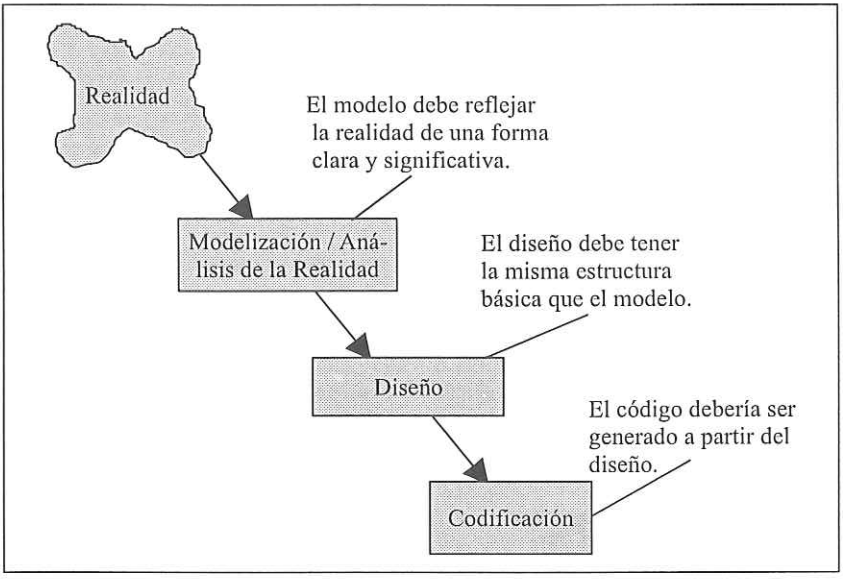

FIGURA 1. Cómo informatizar un sistema.

El proceso es el siguiente:

La fase de **Análisis** consiste en estudiar el problema planteado para obtener una idea clara y concisa sobre los datos de entrada de los que se parte, incluyendo la forma en que estos datos iniciales son obtenidos, así como la información que se desea obtener como resultado y de que manera debe presentarse [4].

Es decir, después de analizar el problema, se han de conocer tres cosas:

1) *Datos* de entrada de que se dispone.

2) *Tratamiento* que ha de realizarse con esos datos.
3) *Información de Salida* deseada.

Por ejemplo, para hacer las facturas de los clientes de una empresa se necesita saber:

1°. Los datos de cada uno de los clientes a facturar. Si esos datos se van a recibir impresos en papel o si se hallan en un fichero o tabla de Clientes, donde en cada ficha o registro se encuentra toda la información referente a cada uno de ellos.
2°. La formula matemática para confeccionar el Importe Total de la factura.

$$\text{Importe\_Total (Cliente, mes)} = \\ = \Sigma \text{ albaranes(Cliente, fecha)} \ \forall \text{ fecha} \in \text{mes}$$

3°. El formato de salida del informe donde se desean imprimir los datos de la factura.

Fase de **Diseño:** Una vez analizado el problema, se precisa diseñar un algoritmo que indique claramente los pasos a seguir para resolverlo. Es decir, es escribir un pseudocódigo de las instrucciones a seguir para llevar a cabo los requerimientos solicitados por el Sistema.

La última fase indicada en la figura 1 es la **Codificación o Implementación**. Para que el algoritmo diseñado en la fase anterior pueda ser introducido en el ordenador, debe ser escrito (codificado) en un lenguaje de programación, siguiendo las reglas de sintaxis del lenguaje escogido.

En definitiva, se va siguiendo un proceso lineal desde que se plantea el problema hasta que se automatiza en una máquina.

2. ¿QUÉ ES LA INGENIERÍA DE LA PROGRAMACIÓN?

El término *Ingeniería* hace referencia a la aplicación de un enfoque sistemático, basado en las Ciencias y las Matemáticas, a la producción de una estructura, máquina, producto, proceso o sistema. Así, la *Ingeniería del Software o de la programación* es el resultado de llevar la tradicional disciplina de las ingenierías al mundo de la construcción de sistemas software. Es una disciplina dependiente de diversas tecnologías: usa *técnicas de gestión* para controlar y gestionar proyectos, y *técnicas de análisis de sistemas* para captar una serie de descripciones que un sistema capaz de ser transformado, mediante la *programación*, a su forma ejecutable en un computador [1].

Surgió como disciplina a partir de la crisis de software producida en los años 70 como consecuencia de la baja calidad del software, su alto coste y un mantenimiento casi imposible.

Así, la Ingeniería del Software abarca tres elementos clave que facilitan el proceso de desarrollo de software y proporcionar a los que practiquen dicha ingeniería las bases para construir software de calidad de una forma productiva. Dichos elementos son:

- **Métodos**: Suministrar el *cómo* construir técnicamente el software.
- **Herramientas**: Suministran un soporte automático o semiautomático a los métodos.
- **Procedimientos**: Integran los métodos y herramientas y facilitan un desarrollo racional y correcto del software.

Ingeniería de la Información es la aplicación de un conjunto de técnicas interrelacionadas para la planificación, diseño, construcción y mantenimiento de sistemas de información para la totalidad de una empresa o para un sector grande de la misma.

3. EVOLUCIÓN Y PERSPECTIVAS

La Ingeniería del Software ha evolucionado rápidamente en las dos últimas décadas. Este avance ha supuesto una auténtica revolución en el mundo del software, ya que se ha pasado de la programación artesanal al desarrollo sistemático de programas que ya pueden ser considerados *productos* desde el punto de vista industrial. En este apartado damos un repaso histórico de cómo han avanzado las técnicas de desarrollo de programas hasta nuestros días y a continuación veremos las perspectivas de futuro de esta disciplina[1].

No entraremos en detalle pero si mencionaremos los principales hitos que han marcado las diversas facetas del desarrollo de programas, fundamentalmente relacionadas con el diseño y la programación.

- **La Programación Estructurada**: Surge a principios de los 70. Hasta entonces los programadores se habían preocupado más de la eficiencia de los programas en términos de velocidad y uso de la memoria que de su legibilidad o mantenimiento, debido a la baja calidad del hardware, pero a medida que este fue mejorando y los programas aumentaban en complejidad, se empezó a ver la importancia del mantenimiento del software y así surge este estilo de programación que dio lugar a la metodología estructurada.
- **Las Metodologías de Diseño**: Surgen a mediados de los 70. Las compañías de desarrollo de software se dieron cuenta de que los sistemas construidos directamente a partir de requisitos informa-

les eran poco eficaces y se necesitaba de una especificación y un diseño del sistema previo a su implementación.

- **Las Metodologías de Análisis**: Fuertemente relacionadas con las ideas de la Programación Estructurada y las Metodologías de Diseño Estructurado, surgen a finales de los 70.
- **La Automatización de los Métodos de la Ingeniería del Software**: Dado que las actividades de desarrollo de software muchas veces consisten en procesos repetitivos o rutinarios, la Ingeniería del Software siempre ha sido una buena candidata a la automatización. Es por ello que surge a principios de los 80. Inicialmente consistió en llevar a una máquina las herramientas gráficas asociadas a determinadas metodologías de análisis o diseño. Posteriormente se han ido perfeccionando, dando lugar a unas mas completas herramientas CASE (Computer-Aided Software Engineering).
- **La Introducción de la Programación Orientada a Objetos**: Surge a mediados de los 80 y continua siendo la tendencia de los 90. Su influencia se compara a la de la programación estructurada de los años 70 [5]. En los siguientes apartados se comenta de forma mas detallada.

En cuanto a las perspectivas de futuro podemos señalar 4 puntos principales [1]: La programación orientada a objetos que acabamos de mencionar, el refinamiento de requerimientos y prototipado rápido, que ayuda enormemente a identificar correctamente los requisitos de los sistemas, los avances en la tecnología CASE y por último la reusabilidad como solución alternativa al desarrollo constante, es decir, poder disponer de librerías de componentes software fácilmente ensamblables puede ahorrar mucho trabajo a los equipos de desarrollo.

4. PARADIGMA ORIENTADO A OBJETOS COMO INTENTO DE SOLUCIÓN

El desarrollo Orientado a Objetos está jugando un papel cada vez más importante en el diseño e implementación de Sistemas Software, hasta el punto de que su influencia está siendo comparada a la de la programación estructurada en los años 70. Se considera como pionero de la aproximación Orientada a Objetos el lenguaje Smalltalk. Lo que en sus inicios fue tan solo un lenguaje de programación dio lugar posteriormente a toda una corriente en el mundo del desarrollo del Software, a cuyo amparo se han desarrollado metodologías de diseño y lenguajes de programación [2].

El *Diseño Orientado a Objetos* es una aproximación diferente a las tradicionales metodologías estructuradas. Estas se basan en la descom-

posición de un sistema en módulos atendiendo a consideraciones procedimentales y/o de datos. En la Orientación a Objetos, en lugar de descomponer los sistemas en módulos que denotan operaciones, los sistemas se estructuran alrededor de los objetos que existen en el modelo del sistema [3]. Si se consigue identificar todos los Objetos componentes de un cierto sistema, así como las relaciones de cada uno con los demás, se genera una visión del mismo mucho más cercana a la realidad, ya que no se separa entre datos y operaciones, sino que ambos se recogen en la definición de los Objetos.

Un Objeto es visto como una unidad que encapsula estructura y comportamiento. Dicho comportamiento es descrito fundamentalmente a través de las respuestas por parte del Objeto a acciones externas y a través también de las acciones que un Objeto puede requerir a otros. La comunicación entre ellos se realiza a través de mensajes, que no son más que peticiones de servicio de unos Objetos a otros [7]. De esta manera, una primera propuesta metodológica para el Diseño Orientado a Objetos [6] podría resumirse en las siguientes:

1. Examinar los requerimientos del sistema e identificar los Objetos en el mismo.
2. Para cada Objeto, identificar las acciones que sufre y/o requiere de otros Objetos. En otras palabras, identificar los mensajes que puede recibir y los que puede enviar cada Objeto.
3. Diseñar las Estructuras de Datos apropiadas para representar los datos dentro de cada Objeto, y elegir algoritmos apropiados para implementar las operaciones a realizar por cada Objeto (sus Métodos).

Las Metodologías Orientadas a Objetos (MOO), basan sus principios en una concepción del mundo compuesto de Objetos que se agrupan en Clases en función de sus características particulares y la función que desempeñen dentro del Sistema. Es decir, las MOO identifican tipos de Objetos [9]. Así, entendemos por Clase la implementación de un tipo de Objeto. La Clase especifica la estructura de los datos y los métodos para implementar cada una de las operaciones concernientes a esos objetos.

Es decir, dividimos el entorno en Clases que a su vez están compuestas de Objetos y especificamos los métodos y operaciones características de cada Clase.

Los Sistemas diseñados siguiendo una Metodología Orientada a Objetos suelen a continuación ser implementados a través de un Lenguaje de Programación Orientado a Objetos. Estos lenguajes recogen toda la terminología de la aproximación Orientada a Objetos, y añaden nuevos mecanismos que son interesantes desde el punto de vista de la programación. Entre ellos, el más destacado es el de la *herencia entre clases*.

La idea básica, es definir nuevos elementos Software (es decir, nuevos Objetos) a partir de otros ya existentes, refinando la definición de estos a base de añadir y/o redefinir nuevos atributos o métodos, con la ventaja de heredar características de la Clase madre [7].

Gran cantidad de lenguajes de programación se han desarrollado de cara a ser usados siguiendo esta metodología.

Tres son las líneas en las que ese desarrollo se ha llevado a cabo [1]:

1. Creación de lenguajes Orientados a objetos. Es la aproximación más pura, y consiste en diseñar nuevos lenguajes que recojan toda la terminología y características de la aproximación Orientada a Objetos. Ejemplos de este tipo de lenguajes son Smalltalk y Eiffel.
2. Desarrollo de extensiones Orientadas a Objetos de lenguajes clásicos. Consiste en el enriquecimiento de lenguajes ya existentes con nuevas estructuras que den cabida al paradigma objetual. Destacan C++ como la extensión más aceptada del lenguaje C, y algunas de las extensiones del lenguaje LISP, como CLOS.
3. Uso de lenguajes no orientados a Objetos como tales: existen autores que defienden el uso de las capacidades en cuanto a modularidad, abstracción, etc., para realizar implementaciones de sistemas diseñados con la metodología orientada a objetos.

5. CARACTERÍSTICAS DEL MODELO ORIENTADO A OBJETOS

Una de las características más destacables del paradigma objetual es que aporta una notación apropiada para el desarrollo de cualquier tipo de Software. Esto no ocurría con las metodologías anteriores, que en cierto modo estaban orientadas demasiado a las aplicaciones de proceso de datos, dejando un tanto de lado las aplicaciones de control.

A continuación se enumeran las principales características ventajosas que las tecnologías con orientación a objetos aportan a la modelización de los sistemas [9]. Son las siguientes:

1. **REUSABILIDAD**: Las clases son diseñadas para que puedan ser reutilizadas en muchos sistemas. Para maximizar la reusabilidad, las clases pueden ser construidas para poder ser personalizadas. El sistema debería contener una colección de clases reutilizables con posibilidad de crecer. Las librerías de clases tienen posibilidad de crecer rápidamente. Un objetivo primordial de las técnicas OO es facilitar la reusabilidad masiva en la construcción del Software.

2. **ESTABILIDAD**: Las clases diseñadas para uso repetido se vuelven estables para facilitar esa reusabilidad.
3. **ENCAPSULACIÓN**: Al diseñar se piensa en términos de comportamiento de objetos, no de detalles de bajo nivel. La encapsulación oculta los detalles y hace las clases complejas fáciles de usar. Las clases son como cajas negras, se usa esa caja negra y no se mira dentro de ella. Sólo se tiene que comprender el comportamiento de esa caja negra y como comunicarse con ella.
4. **CONSTRUCCIÓN DE OBJETOS DE COMPLEJIDAD CRECIENTE**: Los objetos se construyen a partir de otros objetos que a su vez han sido construidos a partir de otros. Esto permite construir componentes Software complejas que se convertirán en nuevos bloques a partir de los cuales se podrá construir más Software complejo.
5. **FIABILIDAD**: El Software construido a partir de clases estables y ya verificadas, es menos probable que tenga defectos que el Software inventado partiendo de cero, sin usar nada ya creado. Cada método en una clase es, en sí mismo, relativamente simple y puede ser diseñado para ser fiable.
6. **NUEVOS MERCADOS DE SOFTWARE**: Las compañías de Software podrán proporcionar librerías de clases para áreas específicas, fácilmente adaptables a las necesidades de las organizaciones que las usen. Serán usadas como lo son las componentes Hardware.
7. **DISEÑO MAS RÁPIDO**: Las aplicaciones son creadas a partir de componentes ya existentes. Muchas de las componentes se construyen de forma que puedan ser personalizadas para un diseño particular.
8. **DISEÑOS DE MAYOR CALIDAD**: Los diseños son a menudo de mayor calidad, ya que se construyen a partir de componentes bien verificados que han sido probados y refinados repetidamente.
9. **INTEGRIDAD**: Las estructuras de datos sólo pueden ser usadas con métodos específicos. Es decir, los objetos de una clase solo responden a los métodos definidos explícitamente para ella. Esto ayuda a crear Sistemas seguros.
10. **PROGRAMACIÓN MAS FÁCIL**: Los programas se construyen con pequeñas piezas, que son generalmente fáciles de crear.
11. **MANTENIMIENTO MAS FÁCIL**: Esto se consigue ya que cada clase maneja sus operaciones independientemente de otras clases.
12. **CICLO DE VIDA DINÁMICO**: El objetivo del desarrollo de sistemas a menudo cambia durante la implementación, pero

cabe la posibilidad de adaptarse a los cambios del sistema, refinar los objetivos si así lo requiere la organización y mejorar constantemente el diseño durante la implementación.

13. **REFINAMIENTOS DURANTE LA CONSTRUCCIÓN**: A menudo es necesario cambiar el diseño del trabajo durante la implementación. Esto conduce a la obtención de mejores resultados finales.
14. **MODELIZACIÓN MAS REALISTA**: El AOO modeliza los sistemas de una forma más cercana a la realidad que el Análisis convencional. El Análisis se traduce directamente en Diseño e Implementación. En técnicas convencionales, el modelo cambia conforme pasamos del análisis al diseño y del diseño a la implementación. Con las técnicas OO, el Análisis, Diseño e Implementación usan el mismo paradigma y sucesivamente lo refinan.
15. **MEJOR COMUNICACIÓN:** Existe una mejor comunicación entre los profesionales del Sistema de Información y sus clientes. Los clientes entienden más fácilmente el paradigma OO. Ellos piensan en términos de eventos, objetos y comportamiento de los objetos. Las metodologías OO proporcionan un mejor entendimiento entre los programadores y los usuarios finales, al compartir un modelo común.
16. **ESPECIFICACIÓN Y DISEÑOS DECLARATIVOS**: La especificación y el diseño debería ser declarativo siempre que sea posible. Esto permite al diseñador pensar más como el usuario final que como la computadora.
17. **INDEPENDENCIA DE DISEÑO**: Las clases se diseñan para ser independientes de los entornos del Software o del Hardware. Emplean peticiones y respuestas de formatos estándar. Esto permite que sean usados con múltiples Sistemas Operativos, manejadores de Bases de Datos, etc. No hay que preocuparse del entorno o esperar hasta que este sea especificado.
18. **AUTOMATIZACIÓN DE BASES DE DATOS DE MAYOR NIVEL**: Las estructuras de datos en BDOO (Bases de Datos Orientadas a Objetos) como por ejemplo la presentada en [8], contienen métodos que toman acciones automáticas. Una BDOO incluye «inteligencia» en su contenido en forma de métodos, mientras que las BDR (Bases de Datos Relacionales) no.
19. **MEJORES HERRAMIENTAS CASE**: Las herramientas CASE usan técnicas gráficas para diseñar clases y sus interacciones. Estas herramientas facilitan la modelización en términos de eventos, desencadenamiento de acciones, estados de los objetos, etc. Y además generan código tan pronto como se definen las clases.

20. **LIBRERÍAS DE CLASES**: Muchos LPOO incorporan librerías de clases que facilitan enormemente la implementación de las aplicaciones.

6. CONCLUSIONES

Como conclusión se resumen las diferencias mas destacables entre las metodologías orientadas a objetos y las tradicionales.

Con **métodos Orientados a Objetos**, la forma en que se modela la realidad difiere de los métodos de Análisis Convencional. Con los métodos Orientados a Objetos, se modela el mundo en términos de tipos de Objetos y lo que le ocurre a esos tipos de Objetos [3]. El mismo modelo y la forma de concebir el sistema se mantiene durante las distintas etapas del proceso de modelización del sistema, es decir, durante las etapas de análisis, diseño y codificación. Es decir, se sigue un proceso mucho mas uniforme.

Con las **metodologías tradicionales** se establece una diferencia más notable en las diferentes etapas del desarrollo. Es decir, durante el análisis se elaboran una serie de esquemas y diagramas y una especificación del sistema que sirven de base para el diseño, pero este se lleva a cabo de forma diferente a como se realizó el análisis. Y posteriormente la implementación desarrollada de forma tradicional siguiendo las técnicas de la programación estructurada [4], sigue un modelo diferente.

7. BIBLIOGRAFÍA

[1] BALZER, R., CHEATMAN, T. E. and GREEN, C., Nov. 1983: «Software Technology in the 1990's: Using a New Paradigm». *IEEE Computer*, pp. 39-45.

[2] BOOCH, G., 1986: «*Object-Oriented Development*». IEEE Transactions on Software Engineering.

[3] MEYER, B., 1988: «*Object-Oriented Software Construction*». Prentice-Hall.

[4] PRESSMAN, R. S., 1988: «*Ingeniería del Software. Un enfoque práctico*», segunda edición. Mc.Graw-Hill.

[5] KIM, W. And LOCHOWSKY, F. H. (eds.), 1989: «*Object-Oriented Concepts, Databases and Applications*». ACM Press, Addison-Wesley.

[6] BOOCH, G., 1991: «*Object-Oriented Design with Applications*». Benjamin-Cummings.

[7] PELECHANO, V., PASTOR, O., 1995: «*Case OO-METHOD: Un Entorno de Producción automática de Software Orientado a Objetos*». Proc. Of CIL-95, Barcelona.

[8] CANÓS, J. H., PENADÉS, M. C., RAMOS, I., PASTOR, O., 1995: «*A Knowledge-Base Architecture for Objects Societies*». Proceedings of DEXA-95, London.

[9] LOZANO, M. D., 1995: «Aplicación del Paradigma Orientado a Objetos: OO-METHOD y Especificación OASIS Comparativo con la Metodología Tradicional en la Informatización de una Estación de Servicio». Proyecto Fin de Carrera dirigido por Isidro Ramos, Facultad de Informática. Universidad Politécnica de Valencia.