LA PROGRAMACIÓN ORIENTADA A OBJETOS FRENTE A LA PROGRAMACIÓN ESTRUCTURADA

Carlos Alberto Vanegas *

1. Introducción

Cuando se habla de la crisis del software se hace énfasis en todos los inconvenientes que se presentan en el proceso de desarrollo, en la calidad de los programas que se producen y en los procesos de mantenimiento. Estos problemas han incidido directamente en el incremento de los costos totales de su desarrollo; así, por ejemplo, el costo del mantenimiento correctivo y evolutivo puede alcanzar alrededor del 70% del costo total del producto. Las posibles fuentes de estos problemas se deben buscar en el proceso mismo de construcción de software y, en especial, en las metodologías utilizadas para tal fin.

El ciclo de desarrollo del software está dividido en tres etapas: análisis, diseño e implementación. En la etapa de análisis se estudia el problema y se establecen los requerimientos que se desean satisfacer; en la etapa de diseño se implanta una solución al problema, y en la etapa de implementación se desarrolla el software. Después de que éste entra en uso se realizan cambios en los requerimientos o errores de funcionamiento. El artículo se centra en la etapa de implementación.

El factor central de la etapa de diseño es el modelaje del mundo en el que se presenta el problema, lo que se logra abstrayendo y copiando las características de sus elementos y sus relaciones, lo mismo que el problema que desea resolver. Esta es, indudablemente, la etapa en la cual se determinan las características estructurales de la solución. Por lo anterior, una metodología de diseño es una forma de lograr un modelo del mundo del problema, un lenguaje de programación es la sintaxis para implantar este modelo, y el esquema de implantación es la manera de traducir el modelo a un lenguaje de programación.

* Ingeniero de Sistemas Universidad Incca de Colombia, estudios de Especialización en Ingenieria de Software Universidad Distrital F.J.C. .Profesor adscrito a la Facultad Tecnológica de la Universidad Distrital F.J.C.

Existen varios modelos válidos para un mismo mundo, diversas maneras de lograrlo, muchas sintaxis para expresarlo y diferentes maneras para hacer la implantación, todas perfectamente utilizables. En consecuencia el programador se debe plantear, entre otras, las siguientes preguntas: ¿qué criterios inciden en la selección de una metodología de diseño?, y ¿qué características del problema hacen que una metodología sea más conveniente que otra?.

A continuación se realizarán las comparaciones pertinentes entre la Programación Orientada a Objetos (POO) y la metodología de Programación Estructurada (PE). En la PE existe uniformidad en la terminología y en los conceptos que se manejan, debido al tiempo que ha tenido para madurar y por su gran difusión. Esto no sucede con la POO; en ella hay un gran problema grave de terminología y una tendencia preocupante hacia la trivialización de todos los conceptos.

1. La Programación Orientada a Objetos (POO)

El paradigma de la POO se basa en la idea de que el mundo puede ser modelado como un conjunto de objetos que interactúan entre sí; programar se reduce entonces a identificar dichos objetos y copiar adecuadamente las características y comportamientos. El objetivo central del diseño es que todo elemento del mundo (objeto real) corresponda directamente a un objeto del mundo computacional (objeto formal) que lo modele. El proceso de programación se puede resumir de la siguiente manera:

- Identificar las clases de objetos y sus atributos
- Identificar y especificar sus operaciones
- Implantar las clases de objetos.

Cada uno de estos pasos corresponde a refinamientos progresivos, buscando cada vez el mejoramiento del modelaje que se está haciendo del mundo. También es muy usual tener que regresar a pasos anteriores para mejorar algún aspecto del diseño.

La POO se basa en cuatro conceptos básicos: *objetos, clases, métodos y herencia*, los cuales se describen a continuación:

- **Objeto**: cada elemento del mundo involucrado en el problema debe tener una adecuada representación dentro de un programa. Un objeto es la representación de un elemento del mundo real (mucho más que un dato, o que un simple conjunto de variables con unos procedimientos asociados). Para realizar esta representación deben copiarse todas las características observables e interesantes del objeto en relación con el problema, como también su comportamiento. Un objeto está compuesto por un conjunto de atributos (objetos más simples) que dan soporte a su estado actual, y por un conjunto de métodos que representa su comportamiento, es decir, la forma de reaccionar a los diversos estímulos que se le pueden presentar. Cada objeto reconoce y puede responder a determinadas acciones denominadas eventos; un evento es una actividad específica y predeterminada, iniciada por el usuario o por el sistema.

- **Clase**: si programar utilizando el enfoque orientado por objetos consistiera en modelar individualmente cada elemento del mundo real involucrado en un problema, la tarea sería realmente larga y pesada. Entonces se debe tratar de no describir elementos individuales sino de encontrar patrones de objetos que, por sus características comunes, puedan ser representados de una manera similar. Cada uno de

estos patrones se denomina una clase, y en conjunto se constituyen en los paquetes principales de la programación orientada a objetos. Las clases y los objetos están estrechamente relacionados, pero no son lo mismo. Una clase contiene información sobre cuál debe ser la apariencia y el comportamiento de un objeto; es el plano o esquema de un objeto; por ejemplo, el esquema eléctrico y de diseño de un teléfono serían similares a una clase. El objeto, o una instancia de la clase, sería el teléfono.

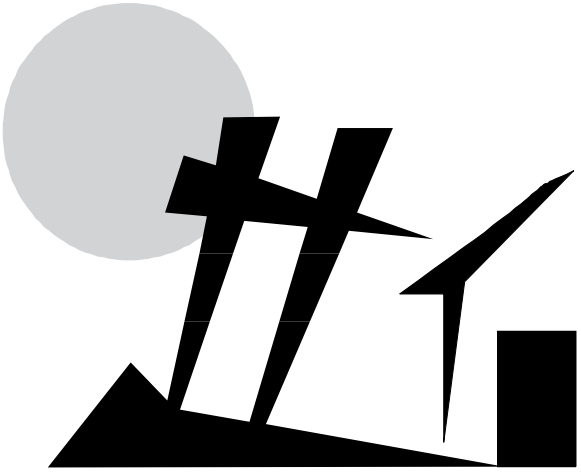
La estructuración de una clase en un lenguaje de programación orientada a objetos sería:

```
class Nombre_de_la_clase
{
    atributos; // encapsulamiento de los datos
    metodos(); // comportamientos del objeto
}
```

- **Método**: el comportamiento de los objetos de una clase se representa a través de métodos. Estos le indican la manera de reaccionar ante los diversos estímulos que le pueden llegar. La interacción entre los objetos del sistema se hace por medio de mensajes: un objeto envía a otro un mensaje a manera de estímulo para obtener de éste una respuesta. El receptor del mensaje decide cuál de sus métodos debe activar para responder. Un método es una secuencia de mensajes a otros objetos más simples (en la mayoría de los casos a sus atributos) para simular la reacción a un mensaje (encontrar la forma de resolverlo).

- **Herencia**: es un mecanismo por el cual se puede aprovechar el diseño de clases ya existente para definir nuevas clases, ya sea especializando sus características y comportamiento (herencia sencilla), o haciendo una composición de varias de ellas (herencia múltiple), en la cual la clase resultante corresponde a la suma de todas las clases. La reutilización de código ahorra tiempo en el desarrollo de software.

La metodología sugerida por la POO se puede resumir brevemente de la siguiente manera:



a. **Identificar las clases de objetos y sus atributos**: para realizar este proceso el programador debe identificar los objetos directamente involucrados en el problema. Para cada uno de ellos debe decidir el conjunto de características importantes para el problema, es decir, sus atributos. Esto produce un conjunto de objetos del mundo real, con todos sus atributos modelados, en términos de objetos más simples.
b. **Identificar y especificar sus operaciones**: cada objeto formal debe tener suficientes métodos para que cualquier transformación imaginable en el mundo real sobre el elemento que se modela sea expresable en términos de las operaciones incluidas en el diseño y, por lo tanto, sea simulable en el mundo formal (un conjunto completo de operaciones). Mientras no se llegue a este estado es necesario incluir nuevas operaciones en la clase del objeto.
c. **Implantar las clases de objetos**: existen tres opciones básicas para hacer la implantación. La primera es utilizar *lenguajes orientados por objetos puros*, como Smalltalk o Eiffel, en los cuales no es necesario tener un esquema de implantación. La segunda opción es utilizar *lenguajes orientados por objetos híbridos* como Object Pascal o C++, para los cuales es necesario establecer un esquema de implantación que no resulta muy complicado y que depende de la calidad de la extensión hecha a los lenguajes originales para manejo de objetos. La tercera opción es utilizar lenguajes imperativos tradicionales como C y Pascal.

2. Programación Estructurada

La metodología de la programación estructurada está basada en la idea que se requiere para resolver un problema y no el mundo en el cual éste ocurre. El modelo representa una solución al problema y corresponde a un conjunto de procesos jerárquicos conectados por medio de flujos de datos, el cual se logra utilizando la técnica de refinamiento a pasos para descomponer las funciones del sistema. Un proceso o módulo es una serie de instrucciones que puede ser invocada por su nombre, encargada de transformar los datos de entrada en datos de salida. La jerarquía entre los procesos significa que la función que realiza un módulo, o el problema que se resuelva, puede ser descompuesto en varios subprocesos; cada uno de ellos resuelve un subproblema del problema final. La definición de los flujos de datos que intervienen en el modelo es lo que se llama diccionario de datos.

Se puede esquematizar este proceso de construcción del modelo en los siguientes pasos:

- Formalizar los resultados del análisis
- Identificar los módulos
- Elaborar el diccionario de datos
- Especificar los módulos
- Implantar los módulos.

A continuación se presentará de manera general cada uno de ellos:

a. **Formalizar los resultados del análisis**: para obtener mejores resultados y facilitar la aplicación de la metodología se presupone que para identificar y analizar el problema se ha usado análisis estructurado. En este paso se elaboran los diagramas de flujo de datos que muestren como fluye la información entre los diferentes procesos; también se crea un diccionario de datos preliminar y se describen las funciones que realizan los procesos.

b. **Identificar los módulos**: los diagramas de flujo de datos se definen por descomposición, esto es, se parte de un diagrama de contexto en donde se define el sistema como una caja negra comunicada con el medio exterior por medio de datos de entrada y de salida. A medida que avanza el diseño este diagrama de contexto se descompone en niveles en los cuales la gran caja negra se subdivide en subprocesos conectados por datos. A partir de estos diagramas se define una estructura arborescente de módulos en la cual existe uno de entrada, uno de transformación y uno de salida.

c. **Elaborar el diccionario de datos**: este se debe actualizar para el proceso de descomposición modular. Cada uno de los flujos de datos se hace por medio de una expresión regular donde las cadenas terminales son los elementos de información.

d. **Especificar los módulos**: para cada módulo identificado se deben realizar especificaciones en términos de los datos de entrada que recibe y de los datos de salida que produce. Existen varias maneras para hacer estas especificaciones de módulos. Algunos se deben hacer utilizando el español estructurado o seudocódigo, en donde se origina un macroalgoritmo del proceso que soluciona el módulo. Otra manera es utilizar el Ada o Prolog como lenguaje de especificación.

e. **Implantar los módulos**: esta etapa se reduce a hacer la codificación en un lenguaje de programación (de preferencia procedimental y estructurada), en la cual a cada módulo corresponde un procedimiento escrito en tal lenguaje. Se trata de un proceso muy simple que suministra la visión de datos pasivos y procesos de modificación sobre ellos; es natural usar lenguajes imperativos. También se definen aquí, de manera formal, las bases de datos a partir del diccionario de datos.

3. Comparación de la POO con la PE

Se realizará una comparación de dos niveles. Un primer nivel en el cual se muestran de manera macroscópica las ventajas y desventajas de la POO con respecto a la programación estructurada, y un segundo nivel en el que se discuten aspectos puntuales de la POO que a primera vista pueden parecer especialmente fuertes o débiles con respecto a la PE.

3.1 Comparación Macroscópica

Las características propias de las dos metodologías permiten destacar que la selección de una de las dos opciones representa un compromiso: facilidad en el diseño o facilidad de mantenimiento.

- El proceso de diseño en POO requiere un mayor esfuerzo que el que se requiere en la PE, ya que es mucho más difuso y frágil debido a la ausencia de una guía salida como es el mismo problema que se requiere resolver para el caso de la PE. Además, el hecho de buscar tanta generalidad (clases completas) exige indudablemente un trabajo adicional.

- El proceso de mantenimiento en POO resulta, en el caso típico, más simple que en la PE. Si la evolución corresponde a cambios en el problema (como ocurre en la mayoría de los casos, ya que las funciones son el componente más variable dentro de un sistema) es inconveniente que la estructura del modelo dependa directamente del problema, dado que su mantenimiento puede implicar cambios en la arquitectura misma de la solución, lo que resulta difícil y costoso. Si la evolución es debida a alteraciones en el mundo, en POO los cambios son fácilmente localizables en el modelo, puesto que existe una clara correspondencia entre éste y el mundo. La dificultad para hacer los cambios depende básicamente de su tamaño. En la PE el principal problema es localizar los puntos del modelo en los cuales se deben hacer las modificaciones, y los posibles efectos laterales que esto puede implicar (no existe encapsulamiento); así, una característica del mundo puede encontrarse repartida por todo el modelo y comunicada por medio de los flujos de datos por múltiples módulos, todos los cuales es necesario actualizar.

En general la POO tiene dos grandes ventajas sobre la PE. La primera es que el proceso de diseño es mucho más fácil de apoyar mediante herramientas y guías que el proceso de mantenimiento, y solo es cuestión de tiempo tenerlas en el mercado. En la PE se han hecho todos los esfuerzos imaginables para apoyar la labor de mantenimiento sin lograr resultados satisfactorios. Esto no debería de sorprender a nadie, ya que por la misma arquitectura del modelo y por la imposibilidad de prever en la fase de diseño su posible evolución, es muy difícil trabajar en metodologías generales de soporte para el proceso de mantenimiento.

La segunda ventaja es que, debido a las facilidades de reutilización de diseños y de software que permite la POO, el costo en tiempo y recursos no resulta, en el caso típico, más alto que el logrado utilizando PE. Así, para muchas aplicaciones resulta más económico utilizar POO que PE.

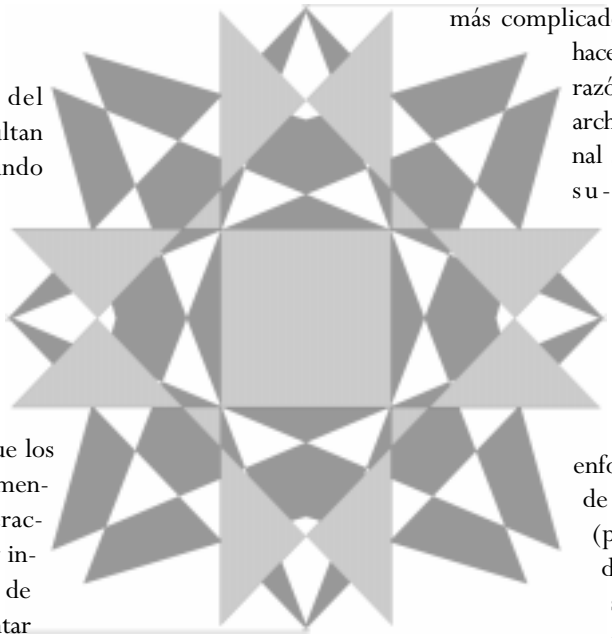
Pero la decisión, que a primera vista puede resultar favorable a la POO, no es tan clara si se piensa con más detenimiento: la teoría y la práctica no siempre están de acuerdo. No existen suficientes herramientas ni son tan poderosas como las que tiene la PE; no hay personal capacitado en POO y esta labor de aprendizaje no parece nada trivial; los *lenguajes orientados por objetos* no se encuentran tan difundidos como los *lenguajes imperativos*, lo cual podría dificultar la portabilidad. De esta forma nadie garantiza, a pesar de la gran cantidad de profetas, que algún día la POO sea una metodología sólida, apoyada y completa.

3.2 Comparación Microscópica

Se compararán ahora las dos metodologías a un nivel más detallado, teniendo en cuenta algunas características de su utilización y la forma de integrar ciertos aspectos al modelo que manejan.

- **Software interactivo**: la interfaz es la parte de un programa encargada de la comunicación con el usuario. Para facilitar su diseño y posterior mantenimiento debe existir una clara división entre la interfaz y el resto del programa. Si se tiene en cuenta que dentro del código de un programa la interfaz puede llegar a constituir cerca de la mitad de él, puede concluirse que una buena metodología debe incluir una forma de diseñar la interfaz y contemplar una manera de integrarla (sin mezclarla) con el resto del programa. Ninguna de las dos metodologías considera este problema; solamente en algunas extensiones recientes

la PE incluye un método adicional para diseñar sistemas interactivos. De las dos metodologías, la PE es la que presenta menos problemas al respecto, puesto que la interfaz se puede incluir sin dificultades en el modelo ya que hace parte del problema que se quiere resolver, aunque no es clara la forma de integrarla sin mezclar. El problema con la POO es que el mundo del usuario (la pantalla, el teclado, el ratón, las ventanas, etc.) no hace parte del mundo del problema (no corresponde a elementos del mundo real que se está modelando) y, por esta razón, no es claro como debe incluirse la interfaz dentro del software, ni cómo debe modelarse la comunicación con el usuario.



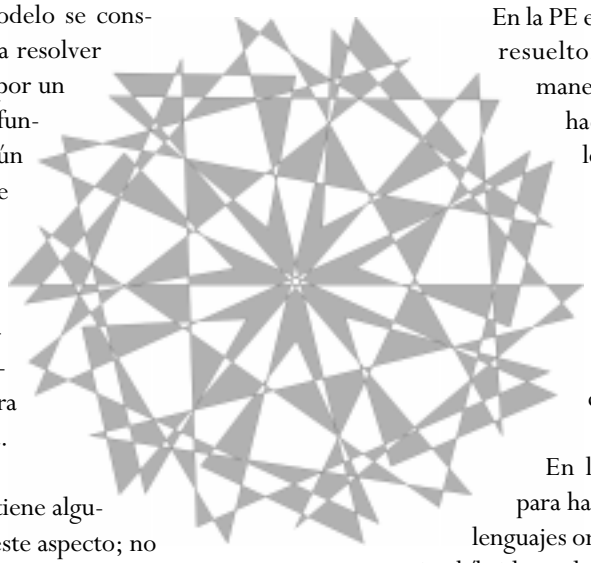
Todos los elementos del mundo del usuario resultan extraños dentro del mundo formal, luego no hay manera de establecer una relación entre ellos y los objetos formales (por lo menos no de manera natural al modelo). En la POO la solución puede ser de dos tipos: dejar que los objetos que modelan elementos del mundo real interactúen con el usuario (muy inconveniente para efectos de mantenimiento) o aumentar el modelo de tal manera que representen el mundo del usuario comunicándose con los objetos del mundo real mediante mecanismos estándares de mensajes. De esta forma, es una lástima que la parte de la interfaz esté tan descuidada en las dos metodologías, porque en la actualidad la mayoría de sistemas se basan en una buena y amigable comunicación con el usuario, y esto implica costos adicionales en el diseño, en la implantación y en el mantenimiento.

- **Memoria secundaria**: el diseño de un programa debe incluir la manera de almacenar y mantener la información mediante el uso de archivos en memoria secundaria o de un sistema de base de datos. Una metodología debe guiar este proceso de diseño, lo mismo que incluir dentro del modelo el concepto de persistencia como una característica adicional de sus elementos. En la PE existen métodos claros y bien estructurados para aprovechar la información contenida en el diccionario de datos y diseñar a partir de él los archivos en memoria secundaria, y las bases de datos que dan soporte al sistema.

  Si el problema consiste en una sencilla administración de información soportada por un sistema de archivos, la PE es, sin ninguna duda, la metodología adecuada. En POO el problema es un poco más complicado. El concepto de dato no hace parte del modelo, y por esta razón al utilizar un sistema de archivos de la manera tradicional se está perdiendo el encapsulamiento de la información que, mediante un conjunto de métodos, asegura que la interpretación que se hace de estos es la adecuada. Por esta razón las mal llamadas *bases de datos orientadas por objetos* son un enfoque orientados por objetos de una base de datos corriente (perfectamente utilizables desde la PE, por ejemplo) o son bases de objetos en las cuales no existen datos como tal sino modelos completos (con su semántica) de elementos del mundo real. Actualmente se utilizan en POO sistemas corrientes de archivos y de bases de datos, sin que haya mucha claridad con respecto a su integración con el modelo y las consecuentes dificultades de mantenimiento.

- **Diseño de funciones**: las funciones del sistema son el conjunto de servicios que puede prestar el programa al usuario (en el fondo corresponden al problema que se quiere resolver); su diseño y posterior inclusión en el modelo debe estar considerado en toda metodología.

En la PE, donde el modelo se construye precisamente para resolver el problema planteado por un conjunto específico de funciones, no existe ningún problema. La fase de análisis debe identificar claramente este conjunto y la metodología lo utiliza desde un comienzo para construir el modelo, basándose en él para obtener su arquitectura.



La POO, por su parte, tiene algunos inconvenientes en este aspecto; no es claro cómo debe enfrentar el proceso de diseño de funciones, ni en qué parte del modelo se deben colocar. Algunos autores proponen colocar como parte del modelo algo llamado la *función de control,* que sería la encargada de administrar el modelo de acuerdo con las funciones que debe tener; sin embargo esta solución tiene dos problemas: el primero es que no existen guías acerca de cómo diseñar esta función (puede resultar un problema tan complicado como el mismo modelaje del mundo), y el segundo es que no corresponde a ninguno de los elementos planteados dentro del modelo de objetos (no es un método), luego habría que aumentar el alcance del modelo para incluir la comunicación entre la función de control y los objetos formales.

- **Implantación del sistema**: uno de los atributos fundamentales de una metodología es que el modelo que se logre en la fase de diseño sea fácilmente implantado. Para esto lo ideal es que el modelo y el lenguaje de programación se muevan dentro del mismo marco conceptual, de manera que sea mínima la traducción. Si esto no ocurre es necesario incluir como parte del diseño un esquema de implantación que indique la manera de traducir cada uno de los elementos del modelo al lenguaje de programación escogido; esto va a ocasionar, en mayor o menor grado, problemas de mantenimiento.

En la PE el problema de implantación esta resuelto. Los lenguajes estructurados manejan los mismos elementos que hacen parte del modelo (módulos-procedimientos, flujo de datos-parámetros, etc.), además de que la metodología sugiere que las especificaciones de los módulos sean tan completas que la etapa de implantación se reduzca a una simple labor de codificación.

En la POO existen tres opciones para hacer la implantación del modelo: lenguajes orientados por objetos puros, lenguajes híbridos o lenguajes imperativos tradicionales; las dificultades dependen del tipo de lenguaje escogido. Al utilizar un lenguaje puro la traducción es natural y no es necesaria la separación entre las fases de diseño e implementación, puesto que la línea divisoria no existe; esto permite que el mantenimiento posterior se vuelva más sencillo. Si se escoge un lenguaje híbrido, todo depende de la calidad de la extensión del lenguaje original. C++ es uno de los más sólidos que existen; se debe definir con cuidado un esquema de implantación, pero esto no representa mayores dificultades. En el caso de usar lenguajes tradicionales se queda sin patrones ni garantías, luego no es lo ideal.

- **Reutilización del software**: la mejor manera de disminuir los costos de producción de software y su tiempo de desarrollo es reutilizar programas anteriores. La programación debería reducirse a conectar adecuadamente un conjunto de elementos ya hechos y aprobados, en lugar de reinventar cada vez algoritmos que se han utilizado siempre. En la PE cada módulo se desarrolla para satisfacer un problema muy concreto, con una especificación muy precisa dentro del contexto definido por el problema global; esto hace que su reutilización no sea posible (por lo menos de una manera natural), ya que sería necesario que ocurriera exactamente el mismo problema en dos módulos distintos. En la POO la reutili-

zación resulta de alguna manera natural; *los lenguajes orientados a objetos* manejan polimorfismos, generecidad, alcance dinámico, etc., lo cual permite el intercambio de piezas completas entre programas con gran facilidad.

- **Extendibilidad**: contar actualmente con software fácil de mantener (extendible) es una necesidad, pues dados los altos costos de los programas es necesario asegurarles una larga vida útil. Si se tiene en cuenta que en la evolución de un sistema las acciones realizadas tienden a ser la parte más variable (volátil), parece mucho más conveniente guiar la arquitectura del modelo por la estructura del mundo y no por las funciones que realiza. Además, aunque se piense en software poco dinámico, son más frecuentes los errores en la fase de análisis y el mismo ajuste inicial puede ser más complicado. Lo ideal es que pequeños cambios en la especificación signifiquen pequeños cambios en el modelo y no una reestructuración del mismo.

- **Facilidades para desarrollo de prototipos**: para crear prototipos de programas es ideal tener ambientes y lenguajes cercanos al modelo; esto permite hacer evaluación a nivel de diseño sin necesidad de esperar hasta terminar la fase de implantación. En PE y POO existen herramientas que lo permiten (ambientes como Eiffel y aun Smalltalk); pero si a la POO se suman las facilidades de desarrollo de prototipos y de extendibilidad se tiene una metodología que permite la creación incremental de software, algo muy deseado en la actualidad.

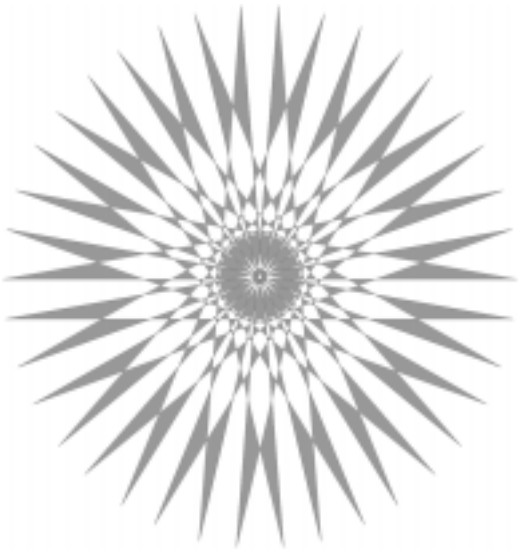
- **Compatibilidad entre los modelos**: muchas cosas se han dicho sobre la compatibilidad de los modelos obtenidos a partir de las dos metodologías, entendiendo ésta como la posibilidad de pasar del uno al otro mediante un proceso automático de traducción. Por ejemplo ya es claro que en el modelo obtenido con PE se ha perdido información sobre ciertas características del mundo que son indispensables para construir el modelo de POO.

4. Un ejemplo de aplicación

A manera de ilustración acerca de las dos metodologías de programación comparadas se creará un programa que permita la inicialización de dos y tres variables respectivamente. Así,

- **Utilizando la Programación Orientada a Objetos**

Se implementan tres clases: *figura, punto y punto 3D*. Desde la clase *figura* se inicializarán las dos y tres variables. Luego se realiza la *herencia* de Punto3D a Punto, con el fin de usar la reutilización de código que existe en la POO. En la codificación se utilizará el *constructor de punto*[1] para inicializar dos de las tres variables de punto3D. Esto nos ahorra la escritura de código y la repetición de procesos que ya existen. El siguiente es el programa resultante:



1 Un constructor es aquel que se inicializa automáticamente y lleva el mismo nombre de la clase

```
class Figura{
  int a=3,b=2, c=4;
  Punto3D p;
  Punto p1;
  public void init() {
    p=new Punto3D(a,b,c);
    p1=new Punto(a,b);
  }
}
class Punto3D extends Punto {
        public int z;
//inicializar las variables utilizando el constructor de
punto
        Punto3D(int x, int y) {
                super(x, y); //super llama al constructor
de Punto
                this.z = 0;
        }
}
public class Punto {
        public int x;
        public int y;
        Punto (int x1, int y1) {
                x = x1;
                y = y1;
        }
}
```

- **Utilizando Programación Estructurada**

  Se implementan dos funciones: *punto y punto3D;* se crea el programa principal y, por medio de las funciones, se envían las tres variables que se van a inicializar. Como se puede apreciar, deben crearse dos funciones totalmente diferentes para la inicialización de las variables, por lo cual tanto existe repetición de código, pues como se ha dicho la PE no permite reutilizaciones. El siguiente es el programa resultante:

```
void Punto3D(int x1, int y1, int z1) {
    x = x1;
    y = y1;
                z = z1;
        }
void    Punto (int x1, int y1) {
                x = x1;
                y = y1;
        }
void main(void)
 {
  int a=3, b=4, c=5;
    Punto3D(a,b,c);
    Punto(a,b);
  }
```

Aunque aparentemente el programa realizado con POO. tiene más líneas de código, supóngase que se debieran repetir mil líneas en un proceso; ahí empezarían los problemas más serios.

REFERENCIAS BIBLIOGRAFICAS

- ***AGUILAR, Joyanes Luis.*** Fundamentos de Programación, Ed. Mc. Graw Hill, España, 1994
- ***DEITEL, y Deitel.*** Cómo Programar en Java, Ed. Prentice Hall, Mexico, 1995
- ***MICROSOFT, Corporation.*** Manual del Programador Visual Foxpro, Microsoft Corporation, EE.UU. , 1995
- ***KENDALL, Kendall.*** Análisis y Diseño de sistemas, Ed. Prentice Hall, México, 1995