ARTÍCULO I

EXPERIENCIA INVESTIGATIVA CON LOS PROYECTOS MIPSOO Y SISMOO

Ricardo de Jesús Botero Tabares
Profesor de Tiempo Completo, Facultad de Informática
Tecnológico de Antioquia – Institución Universitaria
rbotero@tdea.edu.co

RESUMEN

Este artículo relata la experiencia investigativa con dos proyectos de investigación desarrollados por el grupo GIISTA de la Facultad de Informática. El primero, *Método Integrado de Programación Secuencial y Programación Orientada a Objetos para el análisis, diseño y elaboración de algoritmos -MIPSOO*, propone un método de aprendizaje de la programación orientada a objetos que comprende cuatro pasos: definición de la tabla de requerimientos, diseño del diagrama de clases, definición de las responsabilidades de las clases y escritura del seudocódigo. El segundo proyecto,*Sistema para el Modelamiento por Objetos-SISMOO,* implementa un entorno integrado de desarrollo que permite editar, compilar y ejecutar seudocódigo orientado a objetos.

Palabras clave: Aprendizaje basado en problemas, Aprendizajede la programación, Programación orientada a objetos.

INTRODUCCIÓN

Los procesos de aprendizaje en programación de computadoras deben conllevar la aplicación de paradigmas y modelos acordes con los recientes avances tecnológicos en ingeniería de software, en respuesta a las necesidades del sector productivo.

Las nuevas versiones de entornos integrados y lenguajes para el desarrollo de software se apoyan en paradigmas de programación que garantizan un producto final confiable, eficiente y fácil de utilizar[1]. Estas versiones llegan a las universidades y empresas desarrolladoras de software, donde se deben adaptar y comprender de la mejor manera posible para lograr los altos niveles de competitividad que exige una sociedad globalizada e interconectada. Por esto, las instituciones de educación superior que ofrecen programas de tecnología en sistemas, ingeniería en software u otros afines, deben idear mecanismos que propicien, desde los primeros niveles de formación, espacios para fortalecerconceptos esenciales en la formación de ingenieros de software con una visión de futuro cimentada en la sociedad del conocimiento, con una alta capacidad de apropiación e innovación de la tecnología informática.

El aprendizaje de la lógica de programación para resolución de problemas, ligado al proceso pedagógico paralelo implícito, ha sido influenciado por pa-

radigmas que iniciaron con el modelo de la programación libre (con su perogrullado *go to*), pasaron por la diagramación estructurada (programación imperativa con diseño *top-down*) y han desembocado en la programación orientada a objetos (desarrollo basado en componentes con una interfaz pública que posibilita su reutilización). Se observan ahora marcadas tendencias hacia la programación orientada a aspectos y a servicios.

1. ANTECEDENTES

Desde el año 2004 algunos profesores de la Facultad de Informática del Tecnológico de Antioquia que impartían las asignaturas de Algoritmos, Estructuras de Datos, Lenguajes de Programación I e Ingeniería de Software, manifestaron su preocupación porque los contenidos de estas asignaturas giraban en torno a la programación estructurada, cuando las versiones de los productos (compiladores, intérpretes y herramientas CASE) soportaban el paradigma orientado a objetos. Esta contingencia llevó a un grupo de docentes a proponer cambios curriculares en las asignaturas citadas y a idear un método que unificara criterios para la enseñanza y el aprendizaje de los fundamentos de programación con orientación a objetos, mediante la aplicación de un seudolenguaje y estrategias pedagógicas propias del aprendizaje significativo y basado en problemas. Dado que el cambio curricular se hacía lento por las discusiones académicas que la propuesta suscitaba, se presentó al Comité para el Desarrollo de la Investigación del Tecnológico de Antioquia-CODEI,el proyecto de investigación MIPSOO, propuesta realizada por los profesores Eucario Parra, Carlos Castro y Ricardo Botero. Finalizado este proyecto, se unió al grupo el profesor Gabriel Taborda, para fortalecer el grupo de trabajo que consolidó otro proyecto: SISMOO. En ambos proyectos fue fundamental el apoyo de los egresados de la Facultad de Informática Miguel Valencia y Juan David Maya.

2. EL PROYECTO MIPSOO

El método para el aprendizaje y la enseñanza de la programación orientada a objetos del proyecto MIPSOO incluye elementos didácticos del aprendizaje basado en problemas y principios pedagógicos constructivistas; propone un seudolenguaje que toma características asociadas a lenguajes de programación de la familia C/ C++/ Java y al Lenguaje de Modelado Unificado (UML); contiene elementos sintácticos que posibilitan la integración entre los paradigmas de programación estructurada y orientada a objetos, y apoya en gran medida la enseñanza temprana de este último. Además,el proyecto tiene una justificación real basada en los resultados de encuestas a profesores, estudiantes y administrativos de diversas instituciones de educación superior del departamento de Antioquia - Colombia; y una justificación conceptual basada en componentes pedagógicos del modelo constructivista.

El proyecto MIPSOO armoniza con otras experiencias investigativas nacionales [2], [3] y extranjeras [4]; su didáctica implícita propone el seguimiento de cuatro fases para la solución de problemas:

- Definición de la tabla de requerimientos.
- Diseño del diagrama de clases.
- Definición de las responsabilidades de las clases.
- Escritura del seudocódigo (tiene una gran similitud con los lenguajes de programación Java, C# y Visual Basic.Net).

Cada etapa conlleva una serie de formalismos que se explican a continuación.

a) Definición de la tabla de requerimientos

La construcción de la tabla de requerimientos [5] forma parte del análisis del problema. Los requerimientos hacen referencia a las necesidades de los usuarios, es decir, identifican los aspectos que los usuarios del programa desean resolver mediante software. Estos requerimientos se denominan *funcionales* al sostener una relación directa con la funcionalidad del sistema.

La tabla de requerimientos está compuesta por cuatro columnas (ver tabla 1):

- **Identificación del requerimiento**: es un código que identifica al requerimiento, generalmente compuesto por una letra y un dígito. Identificadores comunes para los requerimientos son R1, R2, R3, etc.
- **Descripción**: consiste en una descripción concisa y clara, en lenguaje natural, del requerimiento.

- **Entradas**: son los insumos o datos necesarios para que el requerimiento se pueda suplir con éxito.
- **Resultados o salidas**: constituyen el cumplimiento del requerimiento, es decir, son los resultados que dan solución a un requerimiento funcional definido por el usuario.

Tabla 1. Identificación de requerimientos.

Identificación del requerimiento	Descripción	Entradas	Resultados (Salidas)

Es común que la salida de un requerimiento se convierta en la entrada de otro; los requerimientos funcionales son *casos de uso* que describen de una manera detallada el comportamiento del sistema con los distintos actores que interactúan con él. Un caso para citar es la descripción de un conjunto de secuencias de acciones que un sistema ejecuta y que produce un resultado observable de interés para un actor particular [6]. Un actor es un usuario del sistema.

b) Diagrama de clases

La abstracción de clases también forma parte del análisis del problema y es el primer momento para diseñar su solución. Consiste en una representación gráfica del problema, plano de software, donde se dibujan abstracciones de la realidad relacionadas con el mundo del problema, modelables con software. El plano construido se puede presentar en dos fases, que comprenden un *diagrama conceptual* y un *diagrama de clases*. Es de aclarar que el primero de ellos es opcional en tanto no se requiere cuando los problemas a tratar son pequeños o con cierto margen de trivialidad: mostrar un mensaje, comparar dos cadenas, hallar una sumatoria, entre otros. Un diagrama de clases típico se presenta en la figura 1.

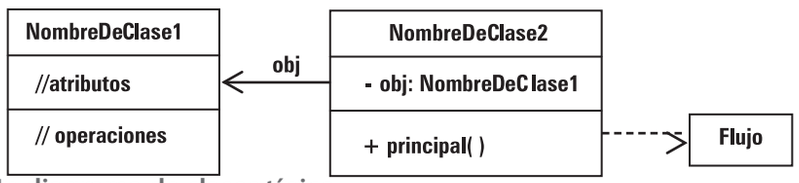

Figura 1. Un diagrama de clases típico.

c) Definición de responsabilidades de las clases

El análisis de responsabilidades de las clases conlleva la descripción de los métodos de cada clase mediante contratos que incluyen los requerimientos asociados, la precondición o estado del objeto antes de ejecutar el método, la postcondición que aclara el estado del objeto luego de ejecutar el método, y el modelo verbal que consiste en una descripción en lenguaje natural de la solución planteada, algo similar al denominado algoritmo cualitativo.

La identificación de responsabilidades forma parte de la documentación de la solución o del futuro sistema basado en software.

Tabla 2. Esquema de un contrato.

Nombre del método	Requerimientos asociados	Precondición	Postcondición	Modelo verbal

d) Escritura del seudocódigo orientado a objetos

El *seudocódigo orientado a objetos (OO)* especifica la solución del problema, pues da cuenta del cómo obtener una solución. Lo hace de una manera similar al proceso de codificación en un lenguaje de programación como Java o C#.

Esta fase conlleva la aplicación de pruebas manuales para cada uno de los métodos (similar a las denominadas *pruebas de escritorio*), o de manera automática con el traductor SISMOO.

Los tipos de datos estándar o predefinidos que se pueden usar en la escritura de seudocódigo, se observan en la tabla 3.

Tabla 3. Valores de inicialización por omisión para los tipos estándar de datos.

Tipo	Valor con el que se debe inicializar	Valor de inicialización por omisión (con descripción)
Numérica (**entero, real**)	Cualquier número (depende del problema)	0 (cero)
carácter	Cualquier carácter	' ' (espacio)
cadena	Cualquier cadena	" " (cadena vacía)
lógico	**falso** o **cierto**	**cierto** (verdadero)
Objeto	**nulo**	**nulo**

Otros problemas resueltos y propuestos donde se aplica el método con las cuatro fases se presentan en *Lógica y programación orientada a objetos: un enfoque basado en problemas* [7], texto que se ha constituido en la guía del módulo *Desarrollar Pensamiento Analítico Sistémico I*, dentro del plan de estudios 72 de Tecnología en Sistemas del Tecnológico de Antioquia.

3. EL PROYECTO SISMOO

Como complemento de MIPSOO, SISMOO es un proyecto de investigación aplicada, cuyo producto de desarrollo es una herramienta de software diseñada para que el usuario pueda editar, compilar y ejecutar los problemas diseñados empleando el seudolenguaje propuesto en MIPSOO. Esta herramienta, denominada en el medio informático *Entorno Integrado de Desarrollo o IDE* (*Integrated Development Environment),* fue desarrollada en lenguaje Java y su diagrama estructural se asemeja al de cualquier compilador (ver figura 2).

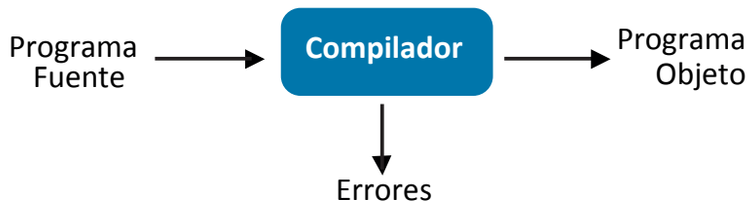

Figura 2. Esquema general de un compilador.

4. MENCIONES Y LOGROS

Los proyectos MIPSOO y SISMOO han logrado una mención nacional y un premio internacional. En Colombia los profesores Carlos Castro, Eucario Parra y Ricardo Botero, en representación de la Universidad de San Buenaventura (Medellín), la Fundación Universitaria Católica del Norte y el Tecnológico de Antioquia, respectivamente, sustentaron en el *VI Encuentro Iberoamericano de Instituciones de Enseñanza de la Ingeniería* [8], el póster de la figura 3, titulado *Método de aprendizaje en fundamentos de programación con orientación a objetos*, por lo cual les otorgaron una Mención Especial, expuesta en la figura 4.

Figura 3. Póster presentado en el VI Encuentro Iberoamericano de Instituciones de Enseñanza de la Ingeniería, organizado por ACOFI.

También, para el *Sexto Simposio Iberoamericano en Educación, Cibernética e Informática*-SIECI 2009 [9], en el contexto de la *Octava Conferencia Iberoamericana en Sistemas, Cibernética e Informática*- CISCI 2009, realizado entre el 10 y 13 de Julio de 2009 en Orlando, Florida - EE.UU, se presentó el artículo titulado M*étodo y entorno integrado de desarrollo para el aprendizaje en lógica de programación orientada por objetos* [10], por el cual los profesores Carlos Castro, Gabriel Taborda y Ricardo Botero obtuvieron el premio al Mejor Artículo Sesión (figura 4).

Figura 4. Reconocimientos en ACOFI 2007 y en SIECI 2009.

5. UN PROBLEMA RESUELTO CON EL MÉTODO PROPUESTO

"La famosa ecuación de Einstein para conversión de una masa *m* en energía, viene dada por la fórmula $E = mc^2$, donde *c* es la velocidad de la luz, *c* = 2.997925 x 1010 m/s. Leer la masa de un objeto en gramos y obtener la cantidad de energía producida en ergios".

La solución a este problema se presenta siguiendo las cuatro etapas planteadas en MIPSOO:

a) Identificación de requerimientos

Se identifican dos requerimientos donde se observa que la entrada del requerimiento R2 es la salida del requerimiento R1 (ver tabla 4).

Tabla 4. Requerimientos para el problema de la ecuación de Einstein.

Identificación del requerimiento	Descripción	Entradas	Salidas
R1	Conocer la masa del objeto en gramos.	Un número real digitado por el usuario.	La masa del objeto está almacenada en memoria.
R2	Calcular la cantidad de energía producida por un objeto.	La masa del objeto (en gramos).	La energía del objeto (en ergios).

b) Diseño del diagrama de clases

El diagrama de clases de la figura 5 conlleva la definición de las abstracciones *Energía* y *Proyecto*, y a la reutilización de las clases de uso común *Flujo* y *Mat*.

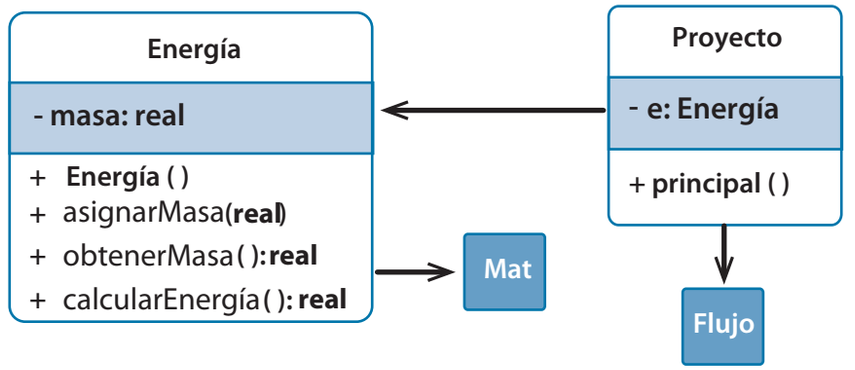

Figura 5. Diagrama de clases para el problema de la ecuación de Einstein.

c) Especificación de las responsabilidades de las clases

Las responsabilidades de las clases se expresan mediante los contratos de cada uno de sus métodos. En términos generales, la clase *Energía* es responsable de almacenar la masa del objeto y calcular su energía; la clase *Proyecto* es responsable de establecer comunicación con el usuario para la captura de la masa del objeto, crearlo, asignarle un estado y visualizar resultados para cumplir con los requerimientos. Las tablas 5 y 6 presentan los contratos de las clases de uso no común *Energía* y *Proyecto*.

Tabla 5. Contratos de la clase Energía.

Método	Requerimiento asociado	Precondición	Poscondición
Energía	R1	No existe un objeto para calcularle su energía.	Existe un objeto en memoria listo para ser procesado.
asignarMasa	R1	El objeto tiene una masa igual a 0 (cero)	El objeto tiene una masa (número real positivo) igual a la especificada por el usuario.
obtenerMasa	R2	El objeto tiene una masa distinta de cero, desconocida para el mundo exterior al objeto.	El mundo exterior al objeto conoce la masa del objeto.
calcularEnergía	R2	Se desconoce la energía producida por el objeto.	Se conoce la energía producida por el objeto.

Tabla 6. Contratos de la clase Proyecto.

Método	Requerimiento asociado	Precondición	Poscondición
principal	R1 y R2	No hay entradas de datos y no se ha efectuado proceso alguno.	Se tiene un objeto en memoria con una masa definida y una energía conocida.

Las responsabilidades de las clases de utilidad *Flujo* y *Mat*, conocidas también como clases de uso común, no se definen de forma explícita porque se asumen comprendidas por el analista; *Flujo* se responsabiliza de las operaciones de entrada y salida estándar y *Mat* de las operaciones con funciones matemáticas.

d) Escritura de seudocódigo

El seudocódigo guarda similitud con la sintaxis de lenguajes de producción como Java y Visual Basic.Net; es un buen preámbulo a la etapa de codificación en alguno de estos u otros lenguajes. La figura 6 ilustra el seudocódigo para este problema, donde las palabras reservadas del seudolenguaje se escriben en negrita, en contraposición a los demás identificadores para nombres de clase – *Energía* y *Proyecto* –, atributos – *masa* y *e* –, métodos –*Energía( )*, *asignarMasa( )*, *obtenerMasa( )* y *calcularEnergía( )*– , variables locales – *ener* – y objetos –*e*.

```
clase Energía
   privado real masa
   público Energía( )
   fin_metodo
   //-------------------------------
   asignarMasa(real m)
      masa = m
   fin_metodo
   //-------------------------------
   real obtenerMasa( )
      retornar masa
   fin_metodo
   //-------------------------------
```

```
    //------------------------------
    real calcularEnergía( )
        real ener
        const real C = 2.997925 * Mat.elevar  (10, 10)
        ener = masa * Mat.elevar(C,  2)
        retornar  ener
    fin_metodo
fin_clase
//-------------------------------------------------------------------------------------
clase Proyecto
    Energía  e
    estatico principal( )
        e = nuevo Energía( )
        Flujo.imprimir  ("Ingrese  la masa del objeto en gramos:")
        real ms = Flujo.leerReal( )
        e.asignarMasa(ms)
        Flujo.imprimir("Energía  producida  =  " + e.calcularEnergía( ) + " ergios")
    fin_metodo
fin_clase
```

Figura 6. Seudocódigo para el problema de la ecuación de Einstein.

6. CONCLUSIONES

- Los cambios curriculares en el área específica de los programas de ingeniería de sistemas y afines, se deben promover desde los primeros niveles de formación, para garantizar avances significativos en un menor tiempo en los niveles intermedios y avanzados.
- Los componentes pedagógicos, didácticos, cognoscitivos, son fundamentales al momento de iniciar estudios de introducción a la programación. El constructivismo y la didáctica con elementos del aprendizaje por descubrimiento, significativo y basado en problemas, pueden garantizar un proceso cognitivo exitoso.
- El currículo, la didáctica y la pedagogía para el aprendizaje de la programación de computadores, debe sustentarse sobre reflexiones serias que conduzcan a metodologías inspiradas en las perspectivas de la ingeniería de sistemas, las tecnologías dominantes de la informática, la aproximación a problemas reales, el rigor abstracto y lógico y la puesta en común entre las necesidades expresas por el sector productivo, las expectativas de los estudiantes y las propuestas de los docentes.
- El paradigma de programación orientado a objetos no debe esperar a saltos curriculares. De esta forma, la abstracción de objetos y comportamientos debe hacer parte de la modelación desde los inicios del proceso de aprendizaje, en los cursos de lógica de programación y estructuras de datos, de tal manera que se traten de forma natural en otras asignaturas relacionadas con lenguajes de programación, bases de datos e ingeniería de software.
- Un enfoque constructivista crea variadas estrategias de enseñanza-aprendizaje: el *aprendizaje por descubrimiento*, donde la nueva información o conocimiento se adquiere a través de los propios esfuerzos del estudiante (tiempo de trabajo independiente), con los aportes del aprendizaje por exposición o instrucción impartido por el docente (horas de trabajo presencial); el *aprendizaje colaborativo*, donde el estudiante puede retroalimentar sus conocimientos con los colegas de su grupo o con comunidades externas contactadas a través de la web; el *aprendizaje significativo*, donde el aprendizaje del alumno depende de la estructura cognitiva previa que se relaciona con la nueva información; y el *aprendizaje basado en problemas*, que incluye características de los ambientes antes citados, donde tanto la adquisición de conocimientos como el desarrollo de habilidades y actitudes resulta importante, dado que un grupo pequeño de alumnos se reúne, con la facilitación de un tutor, a estudiar y resolver un problema seleccionado o diseñado especialmente para el logro de ciertos objetivos de aprendizaje.

REFERENCIAS BIBLIOGRÁFICAS

[1] Sommerville, I. Ingeniería del software. Madrid:Pearson Educación, 2005.

[2] Léxico en programación orientada a objetos. Consultado en agosto 10 de 2010. Disponible: http://ojarami.com/riosur.net/index.php

[3] Proyecto Cupi2, Universidad de los Andes. Facultad de Ingeniería, Departamento de Ingeniería de Sistemas y Computación. Consultado en junio 23 de 2010. Disponible: http://cupi2.uniandes.edu.co/sitio/

[4]Gayo, D.et al. Reflexiones y experiencias sobre la enseñanza de la programación orientada a objetos como único paradigma. Actas de las IX Jornadas de Enseñanza Universitaria de Informática. Cádiz. Julio de 2003. Consultado en abril de 2007. Disponible: http://www.di.uniovi.es/~dani/publications/jenui03.pdf

[5] Villalobos, J. y Casallas, R. Fundamentos de programación: aprendizaje activo basado en casos. México: Pearson Educación, 2006.

[6] Booch, G., Rumbaught, J. y Jacobson, I. El Proceso unificado de desarrollo de software. Madrid: Pearson Educación, 2000.

[7] Botero, R., Castro, C., Taborda, G., Valencia, M. y Maya, J. Lógica y programación orientada a objetos: un enfoque basado en problemas. Grupo GIISTA. Medellín: Divegráficas, 2009.

[8] XXVII Reunión Nacional y VI Encuentro Iberoamericano de Instituciones de Enseñanza de la Ingeniería. El profesor de ingeniería, profesional de la formación de ingenieros. Cartagena de Indias: ACOFI, 2007.

[9] Sexto Simposio Iberoamericano en Educación, Cibernética e Informática- SIECI 2009, Orlando, Florida - EE.UU. Disponible: http://www.iiis.org/CDs2008/CD2009CSC/CISCI2009/

[10] Castro, C., Taborda, G., Botero, R. (2009). Método y entorno inte¬grado de desarrollo para el aprendizaje en lógica de programación orien¬tada por objetos. Revista Iberoamericana de Sistemas, Cibernética e In¬formática. [En línea]. 6, (2). Disponible: http://www.iiisci.org/journal/risci/

ARTÍCULO II

INTRODUCCIÓN A LOS MODELOS DE CALIDAD DEL SOFTWARE BASADOS EN PROCESOS

Darío Enrique Soto Durán
Profesor de Tiempo Completo, Facultad de Informática
Tecnológico de Antioquia - Institución Universitaria
dsoto@tdea.edu.co

Adriana Xiomara Reyes Gamboa
Profesora de Tiempo Completo, Facultad de Ingeniería
Politécnico Colombiano Jaime Isaza Cadavid