Revista Educación en Ingeniería

ISSN 1900-8260

Junio de 2009 • N°. 7 • Pp 120-128 • Publicada en línea por la Asociación Colombiana de Facultades de Ingeniería -ACOFI- www.acofi.edu.co

Enviado: 30/04/2009 • Aprobado: 29/05/2009

UN CAMBIO DE PARADIGMA EN LA ENSEÑANZA DE FUNDAMENTOS DE PROGRAMACIÓN EN INGENIERÍA DE SISTEMAS

Ricardo Timarán Pereira, Universidad de Nariño, San Juan de Pasto (Colombia)
Anívar Chaves Torres, Javier Jiménez Toledo Institución Universitaria CESMAG, San Juan de Pasto (Colombia)
Juan Carlos Checa Mora, Universidad Cooperativa de Colombia,San Juan de Pasto (Colombia)
Hugo Ordóñez Erazo, Universidad Mariana , San Juan de Pasto (Colombia)
Constanza Colunge,Corporación Universitaria Autónoma de Nariño, San Juan de Pasto (Colombia).

Resumen

En este artículo se presentan los resultados del proyecto de investigación cuyo objetivo fue aplicar el modelo de programación funcional con lenguaje Scheme en la enseñanza de fundamentos de programación en Ingeniería de Sistemas. La investigación se desarrolló en cinco instituciones de educación superior de la ciudad de Pasto (Colombia), que forman parte de la Red Universitaria de Investigación en Sistemas de Nariño - RUISNAR. Estas instituciones utilizaban el paradigma de programación imperativo y lenguajes como C y Java en los primeros cursos de programación. Los resultados obtenidos contribuyen a soportar la decisión sobre cuál es el modelo de programación más conveniente para iniciar a los estudiantes de Ingeniería de Sistemas en el campo de la programación.

Palabras clave: Paradigma, enseñanza de programación, modelo funcional, lenguaje Scheme

Abstract

In this paper, the results of the research project, whose objective was to apply the functional programming model with Scheme language in the teaching of programming foundations in Systems Engineering, are presented. The research was conducted in five higher education institutions of the Pasto city (Colombia), which form part of the Systems Research University Network of Nariño – SYRUNAR. These institutions used the imperative programming paradigm and languages like C and Java in the early programming courses. The obtained results help to support the decision on which is the programming model more appropriate to introduce students of Systems Engineering in the field of programming.

Keywords: Paradigm, teaching of programming, functional model, Scheme language

Introducción

Existe un debate sobre cuál es el modelo de programación más apropiado para la enseñanza de la programación en los primeros cursos. Algunas alternativas que se discuten son: programación estructurada, programación orientada a objetos o programación funcional. Los primeros forman parte del paradigma imperativo, mientras que el último, del modelo declarativo (García, 2003).

Esta discusión es relevante desde el punto de vista de Backus (1978), quien defiende que el objetivo básico de un primer curso de programación es proporcionar a los estudiantes los mecanismos necesarios para enfrentarse a la creación de programas para problemas pequeños, en el marco de expresividad de un paradigma de programación, enseñándoles un lenguaje y los conceptos, métodos y técnicas que les permitan abordar los problemas, llegando de un modo riguroso desde la especificación al programa correcto. Por lo tanto, es importante seleccionar el paradigma y el lenguaje a utilizar, los conceptos, métodos y técnicas a enseñar, y encontrar el modo de inculcar al estudiante el razonamiento riguroso y la preocupación por la corrección de los programas (García, 2003).

En las instituciones de educación superior de San Juan de Pasto (Colombia), en las que se ofrece el programa de ingeniería de sistemas se viene aplicando el modelo imperativo, comenzando con programación estructurada y continuando con programación orientada a objetos, utilizando lenguajes como C y Java, principalmente. El modelo declarativo no se ha utilizado para enseñar a los estudiantes la lógica de la programación, en algunos casos se utiliza para hacer las prácticas de algunas asignaturas de último semestre.

Por otra parte, en los programas de ingeniería de sistemas estudiados (Universidad de Nariño, Institución Universitaria CESMAG, Universidad Mariana, Universidad Cooperativa de Colombia, sede Pasto, Corporación Universitaria Autónoma de Nariño), se encontró que en promedio el 34% de los estudiantes reprueban las materias relacionadas con el área de programación. Dentro del porcentaje de los que aprueban, el 70% no ha desarrollado la lógica necesaria para dar una solución computacional a un problema específico.

Ante esta situación la Red Universitaria de Investigaciones en Sistemas de Nariño (RUISNAR) validó la aplicación del modelo funcional, que forma parte del modelo declarativo, y el lenguaje Scheme para mejorar el aprendizaje de la programación. En este documento se presentan los resultados de dicha investigación.

Este artículo se organiza en seis secciones. En la siguiente sección, se aborda, desde una perspectiva conceptual, los paradigmas imperativo y declarativo. En la tercera, se explica cómo ha evolucionado la programación y cómo ha sido enseñada según las teorías y metodologías propias de cada momento. En la cuarta, se presentan datos detallados sobre la metodología aplicada en la investigación. En la quinta, se describen y analizan los resultados de la investigación y finalmente, en la última sección, se dan las conclusiones y futuros trabajos del proyecto.

Modelos de programación

Los modelos de programación se agrupan en dos grandes paradigmas: imperativo y declarativo. Cada uno de éstos comprende: fundamentación teórica, metodología, técnicas y herramientas para el desarrollo de software.

El paradigma imperativo es el más antiguo (1950), los primeros diseñadores se dieron cuenta que la utilización del concepto de variable unido al comando de asignación constituía una abstracción simple pero muy efectiva para poder controlar la información almacenada en la memoria de un computador (Serón, Magallón y Baldassarri, 2006).

En el paradigma imperativo se encuentran los modelos: estructurado, orientado a objetos y orientado a eventos. Estos se caracterizan por describir los programas mediante sentencias que manipulan variables de memoria y cambian el estado del programa. Es decir, los programas imperativos son conjuntos de instrucciones que le indican al computador como realizar una tarea.

En la programación orientada por este modelo se diseñan los programas teniendo en cuenta la arquitectura del computador. Esto implica describir

paso por paso cómo operar los datos en memoria, mediante sentencias que ejecuta el procesador, para alcanzar la solución del problema. Por esto, se dice que la programación imperativa se centra en cómo encontrar la solución.

Por otra parte, el modelo declarativo señala las características que debe tener la solución, sin describir cómo procesarla. Es decir, especifica qué se desea obtener pero no cómo obtenerlo. Un programa declarativo se construye señalando hechos, reglas, restricciones, ecuaciones, transformaciones y otras propiedades derivadas del conjunto de valores que configuran la solución. La programación en un lenguaje declarativo es potencialmente de más alto nivel que la programación imperativa.

El modelo declarativo es un paradigma de programación basado en la lógica en el que se estudian de forma simple muchos aspectos avanzados de los lenguajes de programación modernos. Este estilo de programación encuentra numerosas aplicaciones industriales en campos como las bases de datos, ingeniería del software, procesadores de lenguajes, lenguaje natural, investigación operativa y seguridad de redes (Van Roy y Haridi, 2004).

El objetivo fundamental de los lenguajes de programación declarativa en sentido amplio, es proporcionar un alto nivel de abstracción, de forma que la especificación de un problema sea un programa capaz de resolverlo. En definitiva se intenta liberar al programador de describir detalladamente la secuencia de acciones que debe realizar la máquina para obtener el resultado buscado, como es habitual en los lenguajes imperativos, en los cuales el programador no solo tiene que especificar las restricciones, sino que además tiene que definir cómo se resolverán esas relaciones.

La característica fundamental del paradigma declarativo es que no existe la asignación ni el cambio de estado en un programa. Las variables son identificadores de valores que no cambian en toda la evaluación. Sólo existen valores y expresiones matemáticas que devuelven nuevos valores desde los declarados. A partir de esta información, el sistema debe ser capaz de derivar un esquema de evaluación que permita computar una solución, es decir, no existe una descripción paso a paso de cómo llegar a la solución.

En términos generales, al desarrollar software en el paradigma de programación declarativo, en contraste con el imperativo, se tiende a enfatizar más la evaluación de expresiones que la ejecución secuencial de comandos. Las soluciones declarativas son más cercanas a la manera en que funciona la mente humana al permitir a los programadores describirlas como sistema de expresiones que serán evaluadas.

Enseñanza de la programación

Según García (2003), la enseñanza de la programación se ha desarrollado acorde con las etapas evolutivas de la programación. Dentro de la evolución de la programación se distinguen cuatro grandes etapas caracterizadas por la aplicación de diferentes teorías y herramientas propias del momento histórico y del grado de avance de la ciencia.

En la primera etapa, la programación se concibe como un arte y las principales herramientas del programador son: el lenguaje elegido y sus habilidades personales. El proceso de desarrollo de software no contaba con el respaldo de una metodología ni con teorías que explicaran el funcionamiento de los programas.

La segunda etapa arrancaría con la reunión de Garmisch de 1968 en la que se reconocen las dificultades de programar y la existencia de una crisis del software. Los trabajos de Hoare (1972) sobre tipos de datos, Dijkstra y Feijen (1988 [1984]) sobre programación estructurada abrieron el camino a una visión más formal de la programación, al tiempo que ofrecieron mecanismos de abstracción para dominar la complejidad. También se considera influyente en esta segunda etapa la técnica del refinamiento por pasos sucesivos de Wirth (1971). Mientras que en la primera etapa, la abstracción estructurada era el principal mecanismo para reducir la complejidad y organizar los programas, en esta segunda etapa surgen las abstracciones de datos o tipos abstractos de datos (TAD).

La tercera etapa está caracterizada por la aparición de un conjunto importante de tecnologías de desarrollo de software, la definición de estándares y la importancia del modelado de software. Esta etapa se inicia con el reconocimiento, a principios de la década de 1990, del paradigma orientado a objetos (OO) como el más adecuado para producir software de calidad, esto es, extensible y reutilizable.

La cuarta etapa se propone como una alternativa a los modelos de programación utilizados en las tres primeras y corresponde a la utilización del modelo de programación declarativa que incluye el modelo funcional y el lógico.

En cuanto a la forma de introducir a los estudiantes en la programación, se nota la influencia de las visiones que iban emergiendo en el ámbito de la investigación y la práctica del desarrollo de software, de modo que también se pueden distinguir cuatro etapas. Una etapa inicial en la que la enseñanza se limitaba a describir los elementos de un lenguaje de programación concreto junto con un conjunto de ejemplos que ilustraban su sintaxis y semántica. Con la programación estructurada arrancó una nueva etapa caracterizada por el reconocimiento de Pascal como lenguaje más adecuado para enseñar a programar y el uso del refinamiento por pasos sucesivos cómo principal técnica de diseño de programas. A finales de los ochenta este esquema era utilizado en la mayoría de centros universitarios de todo el mundo (García, 2003). Dos principios básicos de la programación estructurada eran la exigencia de que el flujo de ejecución del programa coincidiese con la secuencia de instrucciones del código, como forma de mejorar la comprensión de los programas, y la creación de los programas aplicando el refinamiento por pasos sucesivos, como medio de dominar la complejidad.

En la década de 1990, el éxito del paradigma orientado a objetos provocó, cómo era de esperarse, el surgimiento de propuestas para enseñar la programación a través de los conceptos de este modelo. La utilización generalizada de lenguajes como C++ y Java, y la utilización del lenguaje de modelado unificado en muchas instituciones de educación superior son evidencia de la acogida de este modelo en la enseñanza (Botero, 2006).

El problema del modelo imperativo consiste en que el estudiante desarrolla unas estructuras mentales muy secuenciales que hace que invierta una gran cantidad de líneas de código y tiempo para la solución de un problema. La raíz de la dificultad para el desarrollo de programas no está en el conocimiento del lenguaje de programación, sino en el conocimiento y aplicación de la lógica aplicada al modelado del problema y el diseño de la solución. Comprender adecuadamente el problema y diseñar una solución, construyendo un modelo cuando sea necesario, exige conocimiento y creatividad, mientras que la implementación o codificación en un lenguaje de programación es un proceso mecánico.

En los últimos años, con el auge de la inteligencia artificial, los sistemas expertos y las bases de datos, se intensifica el interés por el modelo declarativo y dentro de éste el modelo de programación funcional.

La utilización del modelo funcional en la enseñanza de la programación fue una propuesta realizada por John Backus (1978), en su conferencia con motivo de la recepción del prestigioso premio Turing otorgado por la ACM (*Association for Computing Machinery*). Se pronunció en contra del uso de variables, la sentencia de asignación y el uso de sentencias de control. Consideró la programación en el modelo imperativo como carente de propiedades matemáticas que permitan razonar sobre los programas y propuso abiertamente la programación funcional (modelo declarativo) como alternativa. En este modelo la asignación y el uso de variables es innecesario, la recursividad sustituye a las sentencias de control y el Cálculo Lambda (Church, 1934) proporciona el modelo matemático que hace posible razonar acerca de los programas y las funciones de orden superior pasan a ser el mecanismo clave para la reutilización de código previamente escrito.

En la actualidad, existe un debate sobre la conveniencia de aplicar un enfoque estructurado, orientado a objetos o funcional para iniciar la enseñanza de la programación. Incluso, en la última propuesta curricular de Ingeniería de Sistemas, ACM/IEEE, no se ha pronunciado sobre este debate. Considera que cualquiera de estos enfoques puede ser adecuado.

Validación del modelo funcional y lenguaje Scheme

La red de investigación RUISNAR realizó una investigación, con cinco instituciones de educación superior que ofrecen el programa ingeniería de sistemas con el objetivo de comprobar que la utilización del modelo de programación funcional con lenguaje Scheme en la enseñanza de fundamentos de programación contribuye más que el modelo imperativo, en el desarrollo de habilidades en la formulación de algoritmos y desarrollo de programas.

La investigación se desarrolló bajo el enfoque cuantitativo, aplicando el método empírico-analítico. Se propuso un diseño experimental con pre-prueba, posprueba y grupos experimental y de control. El diseño corresponde a un cuasiexperimento, se estableció como grupo experimental, los estudiantes de un determinado semestre que cursan una asignatura de programación (ver cuadro 1) y como grupo de control los estudiantes que ya cursaron la asignatura en un semestre anterior (ver cuadro 2). La técnica de recolección de datos fue la encuesta. Se utilizaron cuestionarios para determinar los conocimientos sobre los fundamentos de programación y ejercicios de programación para establecer la aplicación de dichos fundamentos.

Se preparó un curso de fundamentos de programación orientado bajo el modelo funcional y se escogió el lenguaje Scheme para el desarrollo de las prácticas, teniendo en cuenta que, según García (2003), es necesario que el primer lenguaje de programación que el estudiante aprenda, sea sencillo y que le permita utilizar fácilmente todas las estructuras de un lenguaje para la solución de problemas. Este es el caso del lenguaje funcional Scheme que posee una sintaxis extremadamente simple, que permite que el lenguaje pueda dominarse fácilmente en sólo seis meses (Coello, 1996).

El curso comprendió el estudio de cuatro ejes temáticos: modelos de programación, fundamentos de programación, lenguaje Scheme y la aplicación con PLT Dr.Scheme.

Cuadro 1. Grupos experimentales

Institución	Sem.	Asignatura	I.H.S.	Número estudiantes
Universidad de Nariño	3	Taller de programación II	6	74
Institución Universitaria CESMAG	2	Modelos de Programación	6	79
Universidad Mariana	8	Electiva Programación Funcional	4	32
Universidad Cooperativa	9	Ingeniería del Conocimiento	4	14
Corporación Universitaria Autónoma de Nariño	2	Programación Funcional	4	22
Total estudiantes				216

Cuadro 2. Grupos de control

Institución	Sem.	Asignatura	I.H.S.	Número estudiantes
Universidad de Nariño	5	Estructuras de información	4	38
Institución Universitaria CESMAG	4	Programación orientada a objetos	6	61
Universidad Mariana	9	Desarrollo de aplicaciones multiplataforma	4	12
Universidad Cooperativa	10	Electiva Profesional	4	7
Corporación Universitaria Autónoma de Nariño	5	Programación O-O	4	6
Total estudiantes				124

En el eje de modelos de programación se contempló el estudio de los diferentes paradigmas haciendo énfasis en el funcional. En el eje de fundamentos de programación se estudiaron conceptos como: tipos de datos, expresiones, funciones, entrada y salida de datos, decisiones, cadenas, arreglos, iteraciones y recursividad. En el eje de lenguaje Scheme se explicó la implementación de los fundamentos de programación con este lenguaje. En el eje de aplicación se contempló el uso de la herramienta PLT Dr. Scheme, un editor y compilador de lenguaje Scheme, para el desarrollo de las prácticas del curso.

Se diseñó la ficha de desarrollo temático distribuido en dieciséis semanas en la cual se determina el tema a tratar en cada una, los propósitos que se buscan, la metodología a seguir, los recursos a utilizar y finalmente la estrategia de evaluación a aplicar. Los temas del curso se distribuyeron en ocho unidades: modelos de programación, fundamentos de Scheme, entrada y salida de datos, estructuras de control, matrices y cadenas, recursividad, listas y archivos, éstos se muestran en el cuadro 3. La metodología a seguir, los recursos a utilizar y finalmente la estrategia de evaluación a aplicar, fueron de estricto cumplimiento para todos los docentes participantes en la investigación.

Análisis de resultados

Al comenzar el curso se aplicó una evaluación para determinar los conocimientos de los fundamentos de programación que ya poseían los estudiantes. Los resultados se presentan en el cuadro 4.

Cuadro 3. Temas del curso

1. PARADIGMAS DE PROGRAMACIÓN
1.1 El concepto de paradigma de programación
1.2 Tipos de paradigmas de programación
1.3 Paradigma funcional

2. FUNDAMENTOS DEL LENGUAJE SCHEME
2.1 Historia
2.2 Características del lenguaje Scheme
2.3 Estructura de un programa en Scheme
2.4 Elementos del lenguaje Scheme

3. ENTRADA Y SALIDA DE DATOS
3.1 Escritura de datos
3.2 Lectura de datos

4. DECISIONES
4.1 Las decisiones en programación
4.2 Tipos de decisiones
4.3 Funciones para implementar decisiones
4.4 Decisiones anidadas

5. ITERACIONES
5.1 Contadores y acumuladores
5.2 Forma especial let
5.3 Expresión do

6. RECURSIVIDAD
6.1 La recursividad como técnica de programación
6.2 Estructura de una función recursiva
6.3 Ejecución de funciones recursivas
6.4 Envolturas para funciones recursivas
6.5 Tipos de recursividad
6.6 Recursividad en Scheme
6.7 Ejemplos de soluciones recursivas
6.8 Eficiencia de la recursividad

7. ARREGLOS
7.1 El concepto de vector
7.2 Definición de un vector en Scheme
7.3 Funciones para manejo de vectores
7.4 Búsqueda de un elemento en un vector

8. LISTAS
8.1 Manejo de listas con Scheme
8.2 Funciones para manejo de listas

Cuadro 4.Resultados preevaluación grupos piloto

No.	Pregunta	% Correctas	% Incorrectas	% No Contestadas
1	Un paradigma de programación es…	43,9	37,5	18,6
2	El lenguaje de programación Scheme está asociado al paradigma…	29,2	29,8	41,0
3	Una serie finita de pasos lógicos para solucionar un problema recibe el nombre de…	63,0	26,0	11,0
4	Una variable es…	89,9	9,6	0,5
5	En programación, una función es…	63,8	29,8	6,4
6	Cuál de las siguientes expresiones esta en notación prefija	48,6	25,3	26,0
7	Dentro de un programa una estructura condicional permite	77,6	14,7	7,7
8	Dentro de un programa un ciclo se utiliza para	86,0	11,8	2,2
9	Un arreglo es	51,5	28,2	20,3
10	Se entiende por recursividad …	33,7	45,5	20,8
Promedio		58,7	25,8	15,5

Se evidencia que todos los temas son conocidos por algunos de los estudiantes, algunos por la mayoría como el concepto de variable y ciclo, otros son conocidos solo por una minoría como es el caso de lenguaje Scheme y recursividad.

El mismo instrumento se aplicó a los estudiantes que en cada institución habían cursado la asignatura en el semestre anterior bajo el modelo imperativo (grupos de control). Los resultados se presentan en el cuadro 5.

Cuadro 5. Resultados preevaluación grupos de control

Pregunta	% Correctas	% Incorrectas	% No contestadas
1	56,6	32,6	10,8
2	31,0	41,6	27,4
3	73,2	23,6	3,2
4	94,5	5,5	0,0
5	72,1	24,6	3,3
6	53,8	23,5	12,7
7	77,8	13,5	8,8
8	93,6	6,4	0,0
9	68,6	21,0	10,4
10	43,6	36,5	19,9
Promedio	66,5	23,9	9,7

De igual manera que los grupos experimentales, los temas más conocidos fueron variables y ciclos y los menos conocidos, lenguaje Scheme y recursividad. Este resultado se presenta por cuando estos temas son propios del modelo funcional y no se estudian en los cursos diseñados desde el modelo imperativo.

De acuerdo con los cuadros 4 y 5, mientras el promedio de respuestas correctas de los grupos piloto fue de 58,7%, el de los grupos de control fue de 66,5%, a la vez que el promedio de respuestas incorrectas fue mayor en los grupos piloto que en los de control, 25,8% y 23,9% respectivamente, y de igual manera, el promedio de las preguntas no contestadas fue mayor en los grupos piloto que en los de control, con valores del 15,5% contra 9,7%.

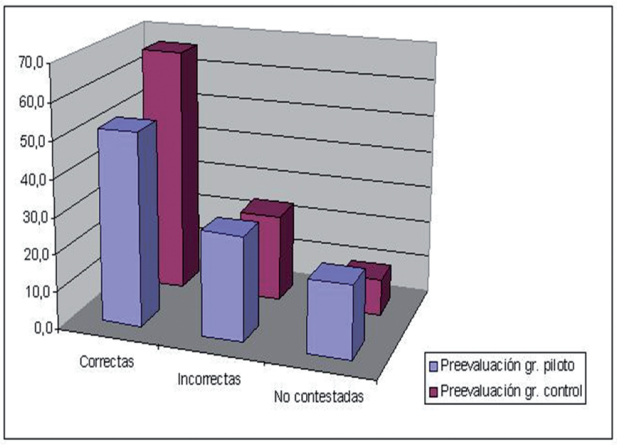
Comparando los resultados de la evaluación diagnóstica con los grupos piloto y de control, que se muestran en la figura 1, se puede decir que el conocimiento del grupo piloto, por sus condiciones académicas, es inferior en 6,8% en los temas evaluados, al grupo de control. Este resultado era el esperado al iniciar la investigación.

El curso de programación funcional con Scheme, en las diferentes instituciones, se desarrolló según los contenidos, el orden, los propósitos, la metodología, los recursos y la estrategia de evaluación planeados por la red de investigación RUISNAR.

Al finalizar el semestre se realizó la post-evaluación con el mismo instrumento de la pre-evaluación para determinar el avance de los grupos piloto en el

aprendizaje de los fundamentos de programación. Los resultados se muestran en el cuadro 6.

Figura 1. Consolidados de la preprueba



Cuadro 6. Resultados post-evaluación grupos piloto

No.	Pregunta	% Correctas	% Incorrectas	% No Contestadas
1	Un paradigma de programación es…	71,5	27,1	1,4
2	El lenguaje de programación Scheme está asociado al paradigma…	90,7	7,7	1,6
3	Una serie finita de pasos lógicos para solucionar un problema recibe el nombre de…	81,1	17,7	1,2
4	Una variable es…	88,4	10,4	1,2
5	En programación, una función es…	85,7	13,3	1,0
6	Cuál de las siguientes expresiones esta en notación prefija	94,0	3,6	2,4
7	Dentro de un programa una estructura condicional permite	91,8	4,6	3,6
8	Dentro de un programa un ciclo se utiliza para	90,4	8,4	1,2
9	Un arreglo es	56,4	35,2	8,4
10	Se entiende por recursividad …	76,7	19,9	3,4
	Promedio	82,7	14,8	2,5

Para evaluar la competencia en la solución de problemas utilizando el lenguaje Scheme se diseñó y aplicó, a los grupos piloto, una prueba final práctica compuesta por cuatro problemas orientada a evaluar, en primer lugar tres aspectos fundamentales en la solución de problemas: la comprensión del problema, la utilización de estructuras de programación y la eficacia de la solución. Los resultados de la evaluación de estos criterios se muestran en el cuadro 7. En segundo lugar la prueba se orientó a evaluar temas específicos de los fundamentos de programación: decisiones, ciclos, matrices y recursividad. Los resultados de esta prueba se muestran en el cuadro 8.

Cuadro 7. Evaluación final sobre criterios de solución de problemas

Criterios de evaluación	% Excelente	% Bueno	% Regular	% Malo	% Deficiente	% No presentó
Compresión problema CR1	43,1	22,3	17,6	5,4	1,6	10,1
Utilización de estructuras CR2	41,2	14,8	18,8	10,6	4,6	10,1
Solución efectiva CR3	33,0	11,4	16,6	18,3	10,7	10,1

Cuadro 8. Evaluación final sobre fundamentos de programación

Fundamentos	% Excelente	% Bueno	% Regular	% Malo	% Deficiente	% No presentó
Decisiones	50,8	17,8	16,8	6,8	3,7	4,3
Ciclos	37,6	18,1	23,3	9,9	3,6	7,5
Matrices	42,8	16,6	13,8	11,1	5,8	9,8
Recursividad	25,1	12,2	16,8	17,9	9,4	18,6

Los resultados de cada uno de los criterios discriminados por problema se muestran en el cuadro 9.

Cuadro 9. Evaluación final sobre fundamentos de programación

Totales – Porcentajes	Ejercicio 1			Ejercicio 2			Ejercicio 3			Ejercicio 4		
	CR1	CR2	CR3	CR1	CR2	CR3	CR1	CR2	CR3	CR1	CR2	CR3
Excelente	53,9	55,7	42,7	39,2	43,8	29,9	48,3	42,0	38,1	37,0	27,7	25,1
Bueno	20,3	16,5	16,5	26,7	15,8	11,7	21,5	16,4	11,9	21,0	11,4	6,7
Regular	16,5	16,9	16,9	23,5	22,5	23,9	15,2	12,7	13,5	15,7	24,7	12,8
Malo	2,4	3,9	14,0	2,6	10,1	17,1	5,1	13,1	15,1	11,7	15,7	28,2
Deficiente	2,8	2,8	5,6	0,4	0,4	9,9	0,0	5,9	11,5	3,2	9,1	15,8
No presentó	4,3	4,3	4,3	7,5	7,5	7,5	9,8	9,8	9,8	18,6	18,6	18,6

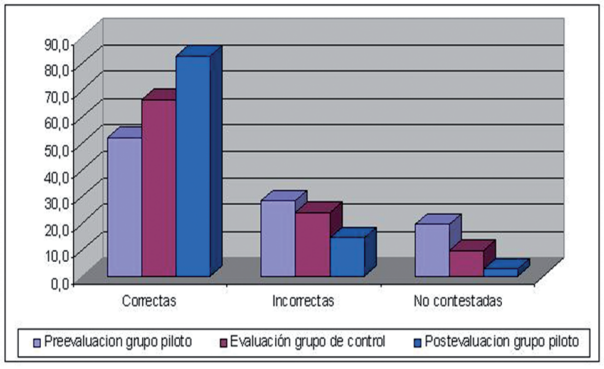
El ejercicio 1 estuvo enfocado a evaluar el manejo de estructuras de decisión; el ejercicio 2, estructuras iterativas; el 3, manejo de matrices y el 4, aplicación de recursividad.

CR1 (criterio 1) corresponde a comprensión del problema, CR2 a utilización de estructuras de programación y CR3 a eficacia de la solución.

Al comparar los resultados de la preevaluación y la postevaluación con los resultados obtenidos por el grupo de control (figura 2) se encuentra que los estudiantes aumentaron el promedio de respuestas correctas en un 24%, a la vez que disminuyeron el promedio de incorrectas en un 11% y las preguntas no contestadas bajaron en un 13%, con respecto a la prueba diagnóstica. Por otra parte, los grupos piloto superaron en respuestas correctas con un 16,2% a los grupos de control, tuvieron un promedio de respues-

tas incorrectas 9,1% menos que los grupos de control y 7,2% menos de preguntas no contestadas a las de los grupos de control.

Figura 2. Resultados preevaluación y postevaluación



Analizando los resultados de la prueba final práctica que se presentan en el cuadro 7, se puede decir que el modelo funcional y, particularmente, el lenguaje Scheme, permitió que un 65,4% de los estudiantes tuviera una buena comprensión de los problemas planteados (columnas excelente y bueno), un 56% hiciera una correcta utilización de las estructuras de programación y un 44,4% lograra una solución efectiva.

En cuanto a la aplicación de los fundamentos de programación con el lenguaje Scheme, de acuerdo con el cuadro 8, el 68,6% de los estudiantes aplicó correctamente (columnas excelente y bueno) las estructuras de decisión, el 55.7% las estructuras iterativas, el 59,4% matrices y el 37,3%, soluciona problemas utilizando recursividad. En general, el 55,25% de los estudiantes que participaron en los grupos piloto alcanzó un buen desempeño en el desarrollo de programas con el lenguaje Scheme, el 17,7% tuvo un desempeño aceptable y el 27,1% un desempeño deficiente.

Conclusiones y trabajos futuros

Se han presentado los resultados de la primera experiencia del grupo de universidades participantes en la investigación de la red RUISNAR, de aplicar el modelo funcional con el lenguaje Scheme, como un cambio de paradigma en la enseñanza de los fundamentos de programación en ingeniería de sistemas.

El modelo de programación funcional no es nuevo; sin embargo, la aplicación en las instituciones de educación superior de Pasto constituyó un proyecto novedoso, que cambió no solo la forma de entender la programación, sino que facilitó su aprendizaje a los estudiantes de ingeniería de sistemas de una manera más agradable y productiva.

Los resultados obtenidos demostraron que la enseñanza del modelo de programación funcional con el lenguaje Scheme, en los primeros cursos de programación, facilita a los estudiantes de ingeniería de sistemas, el desarrollo de habilidades en la formulación de algoritmos, elaboración de programas y la comprensión de las estructuras fundamentales de un lenguaje de programación, con respecto al modelo imperativo. Mejora la capacidad de entender y utilizar la recursividad como una herramienta para optimizar su desempeño en la construcción de programas que brinden una solución efectiva a problemas específicos. Con base en esta experiencia y considerando sus beneficios, se recomendó incluir dentro de los planes de estudio de ingeniería de sistemas de las instituciones participantes, la enseñanza de este modelo de programación.

Como trabajos futuros están el documentar los resultados del rendimiento de los estudiantes que participaron en este proyecto, en las asignaturas posteriores del componente de programación y formalizar en los planes de estudio del programa de Ingeniería de Sistemas, de las universidades del departamento de Nariño (Colombia), la programación funcional con Scheme, como el primer curso de enseñanza de los fundamentos de programación.

Agradecimientos

A la Red de Universidades Regionales Latinoamericanas UREL Capítulo Nariño, por financiar el proyecto y a sus rectores y vicerrectores de investigación por su apoyo incondicional para la culminación exitosa de esta investigación.

Referencias

Backus, J. (1978). Can Programming Be Liberated From the Von Neumann Style? A Functional Style and its Algebra of Programs. Communications of the ACM, Vol. 21, No. 8, August, pp. 613-645.

Botero, R. (2006). Fundamentos de Programación con Orientación a Objetos. Tecnológico de Antioquia, Medellin, Colombia.

Church, A. (1941). The Calculi of Lambda Conversion. Annals of Mathematical Studies, No.6, Princeton University Press, Princeton N.J., USA.

Dijkstra, E. y Feijen, W. (1988: 1984). A Method of Programming. Addison-Wesley, Boston, USA.

García, J. (2003). Un Enfoque Semiformal para la Introducción a la Programación. Departamento de Informática y Sistemas. Universidad de Murcia, España.

Hoare, C. (1972). Proff of correctness of data representations. Acta Informática 1(1):271-281.

Serón, F., Magallón, J. y Baldassarri, S. (2006). Lenguajes de programación. Universidad de Zaragoza, España.

Van Roy P. y Haridi, S. (2004). Concepts, Techiques, and Models of Computer Programming. The MIT Press, Cambridge, USA.

Wirth N (1971). Program Development by Stepwise Refinement. Communications of the ACM, Vol. 14, No. 4, April. Pp 221 – 227.

Sobre los autores

Ricardo Timarán Pereira
Doctor en Ingeniería, Master of Science en Ingeniería, Especialista en Multimedia, Ingeniero de Sistemas y Computación. Profesor Asociado del Departamento de Sistemas de la Facultad de Ingeniería de la Universidad de Nariño. Director de los grupos de investigación GRIAS y RUISNAR. Sus campos de investigación son el Descubrimiento de Conocimiento en Bases de Datos, Procesamiento Analítico en Línea y Bodegas de Datos. Ciudad Universitaria Torabajo. San Juan de Pasto (Colombia)
ritimar@udenar.edu.co

Anívar Chaves Torres
Especialista en Docencia Universitaria e Ingeniero de Sistemas. Docente Tiempo Completo del Programa de Ingeniería de Sistemas de la Institución Universitaria CESMAG de Pasto. Profesor Hora Cátedra del Departamento de Sistemas de la Facultad de Ingeniería de la Universidad de Nariño. Sus áreas de trabajo son la programación y la investigación,
San Juan de Pasto (Colombia)
anivarchaves@yahoo.com.

Juan Carlos Checa Mora
Especialista en Multimedia Educativa e Ingeniero de Sistemas. Docente de la Universidad Cooperativa de Colombia, San Juan de Pasto (Colombia)
jccheca@gmail.com.

Javier Jiménez Toledo
Especialista en Docencia Universitaria e Ingeniero de Sistemas. Docente medio tiempo del Programa de Ingeniería de Sistemas de la Institución Universitaria CESMAG de Pasto, Profesor Hora Cátedra del Departamento de Sistemas de la Facultad de Ingeniería de la Universidad de Nariño. Sus áreas de trabajo son la programación, la computación gráfica y la investigación. San Juan de Pasto (Colombia)
javierjx@gmail.com

Hugo Ordóñez Erazo
Especialista en Gerencia Informática e Ingeniero de Sistemas. Docente tiempo completo del Programa Ingeniería de Sistemas de la Universidad Mariana, San Juan de Pasto (Colombia)
hugoeraso@gmail.com

Constanza Colunge
Ingeniera de Sistemas. Decana y docente de la facultad de Ingeniería de la Corporación Universitaria Autónoma de Nariño, San Juan de Pasto (Colombia)
conny@gmail.com