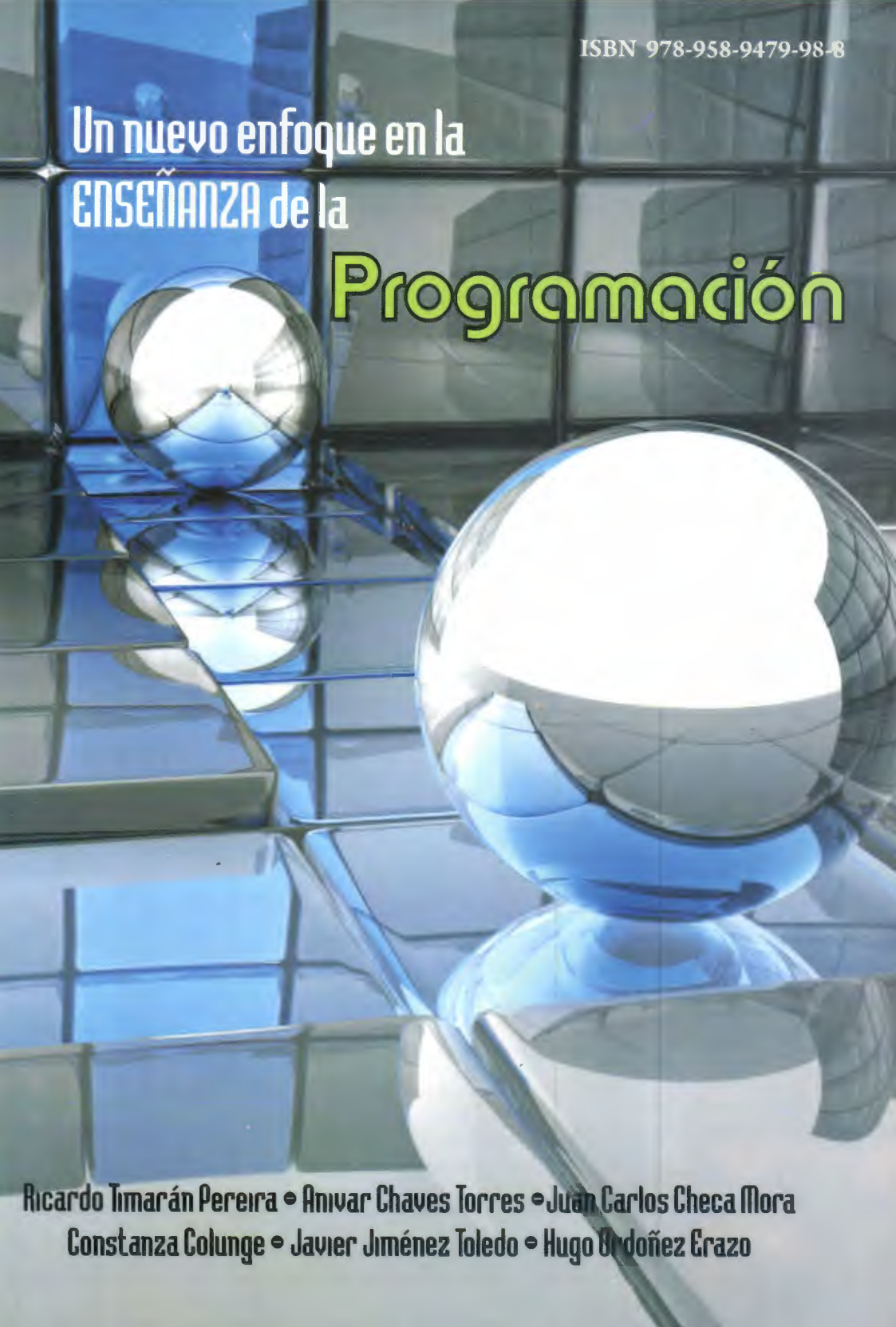
ISBN 978-958-9479-98-8

Un nuevo enfoque en la ENSEÑANZA de la Programación



Ricardo Timarán Pereira • Anivar Chaves Torres • Juan Carlos Checa Mora
Constanza Colunge • Javier Jiménez Toledo • Hugo Ordoñez Erazo

Un Nuevo Enfoque en la Enseñanza de la Programación

Un Nuevo Enfoque en la Enseñanza de la Programación

Ricardo Timarán Pereira
Universidad de Nariño

Anívar Chaves Torres
Institutución Universitaria CESMAG

Juan Carlos Checa Mora
Universidad Cooperativa de Colombia - Pasto

Constanza Colunge
Corporación Universitaria Autónoma de Nariño

Javier Jiménez Toledo
Universidad de Nariño

Hugo Ordóñez Erazo
Universidad Mariana

RUISNAR
Red Universitaria de Investigacion en Siste nas de Nariño

UN NUEVO ENFOQUE EN LA ENSEÑANZA DE LA PROGRAMACIÓN

Editorial Universitaria Universidad de Nariño
ISBN: 978-958-9479-98-8

Edición y diagramación:
María Elena Mesías - Anívar Chaves Torres

Diseño de portada
Mauricio Riascos

Impreso en el Centro de Publicaciones
de la Universidad de Nariño.

San Juan de Pasto, Nariño, Colombia

A mi madre Rosa María
que desde arriba ilumina
y protege mi camino,
y a mi esposa Amanda,
que es el amor de mi vida.
Ricardo

A mis padres:
Emérita y Pedro,
por su apoyo y su confianza.
Aníval

Para Julián,
su hermanito y su mamá.
Juan Carlos

A Dios por ser fuente de sabiduría,
a mi hija Valery por ser mi fuerza,
a Jaime por su apoyo y comprensión y
a mis padres y hermanos
por su ejemplo y confianza.
Constanza

A mis padres: F. Javier Jiménez
y Carmen Toledo, por su enorme amor
y sabiduría, a mi hijo Davis,
mi esposa Ximena a mis sobrinos
y hermanos por su incondicional apoyo.
Javier

A DIOS,
fuente de toda sabiduría y conocimiento.
A mi madre Helena a quien amo.
Hugo

RED DE UNIVERSIDADES REGIONALES LATINOAMERICANAS

UREL

Capítulo Nariño

Carlos Folleco Erazo
Rector Universidad Cooperativa de Colombia
Presidente Red UREL Capítulo Nariño

Padre Evaristo Acosta Maestre
Rector Institución Universitaria CESMAG

Jaime Tito Colunge Benavides
Rector Corporación Universitaria
Autónoma de Nariño

Luis Enrique Ortega Ruales
Director Zonal Universidad Nacional
Abierta y a Distancia – UNAD

Orlando Rodríguez García
Director Fundación Universitaria San Martín
Sede Pasto

Deyanira Ortega
Directora Universidad Antonio Nariño
Sede Pasto

Gonzalo Hernández Arteaga
Director Regional Universidad Javeriana

Silvio Sánchez Fajardo
Rector Universidad de Nariño

Hermana Martha Estela Santa Castrillón
Rectora Universidad Mariana

Melba Rivas Benavides
Directora Universidad Santo Tomás
Sede Pasto

Luis Antonio Heredia
Coordinador Fundación Universitaria
del Área Andina
Sede Pasto

Rocío de la Espriella Guerrero
Directora Escuela Superior de Administración
Pública – ESAP

Roberto Narváez Moreno
Director Fundación Universitaria Remington
Sede Pasto

Sara Ángela Arturo González
Directora Regional Servicio Nacional
de Aprendizaje – SENA

Gerardo Mesías Méndez
Director Ejecutivo Cámara de
Comercio de Pasto

RED UNIVERSITARIA DE INVESTIGACIONES

UREL

Capítulo Nariño

RED UNIVERSITARIA DE INVESTIGACIÓN EN SISTEMAS DE NARIÑO

RUISNAR

Ricardo Timarán Pereira
Doctor en Ingeniería
Master of Science en Ingeniería
Especialista en Multimedia
Ingeniero de Sistemas y Computación
Director

Anívar Chaves Torres
Especialista en Docencia Universitaria
Ingeniero de Sistemas
Investigador

Juan Carlos Checa Mora
Especialista en Multimedia Educativa
Ingeniero de Sistemas
Investigador

Constanza Colunge
Ingeniera de Sistemas
Investigadora

Javier Jiménez Toledo
Especialista en Docencia Universitaria
Ingeniero de Sistemas
Investigador

Hugo Ordóñez Erazo
Especialista en Gerencia Informática
Ingeniero de Sistemas
Investigador

CONTENIDO

LISTA DE CUADROS

LISTA DE FIGURAS

PRÓLOGO

Un sueño que se sueña solo,
es sólo un sueño,
un sueño, soñado juntos,
se transforma en realidad.
Raúl Seixas

Durante el mes de julio del año 2006, los Rectores de las universidades nariñenses, dieron inicio a la Red Universitaria de Investigaciones -UREL Capítulo Nariño, cuya misión se planteó en los siguientes términos: "Promover, gestionar, orientar y coordinar de manera concertada e interinstitucional la realización de investigaciones y actividades conexas, con calidad y pertinencia; de tal manera que aporten al desarrollo regional, en concordancia con la filosofía y principios de las instituciones que la integran", con la certeza de que el conocimiento, la ciencia, la tecnología y la innovación derivadas de la investigación, se constituyen en el eje orientador de prosperidad regional y, a la vez, son fuente de vida de todo progreso.

Entendiendo la globalización como algo que afecta e impacta a cada ser y sector en particular y a toda la población mundial en su conjunto, en todos los campos; los investigadores de la Red, asumen esta situación, en sentido estricto, como posibilidades múltiples y, a su vez, ambivalentes, en busca de nuevos escenarios para crear y desarrollar conocimiento y abrir espacios, donde acontece la verda-dera vida de la humanidad, allí donde ocurre la producción y transmisión de saberes y el devenir de la cultura.

Con el ánimo de iniciar la labor investigativa, se convocó al Doctor en Ingeniería con Énfasis en Ciencias de la Computación, Ricardo Timarán Pereira, para liderar la conformación de un grupo de Investigación en Ingeniería de Sistemas; designación acogida y realizada con profesionalismo de excelencia. Este equipo de docentes investigadores denominado Red Universitaria de Investigación en Sistemas de Nariño -RUISNAR, son quienes hoy poseen el conocimiento, resultado de un trabajo responsable, sistemático y especializado, que les permite con autoridad del saber, orientar la formación profesional de los Ingenieros de Sistemas, para responder a las tendencias de la educación superior, de la sociedad y del mercado laboral; profesionales competentes en su disciplina y comprometidos con la realidad tecnológica de su entorno.

La simbiosis: investigación - TIC *(Tecnología - Información - Comunicación),* rompe los muros del saber, abre las puertas de los claustros universitarios, sobrepasando fronteras para alcanzar la multidimensionalidad del hombre, traspasando lo cercano y conocido, lo mediato e inmediato, incentivando a traspasar lindes imaginarios, ideológicos y de pensamiento. Hoy, un fenómeno no amerita estudiarse exclusivamente en tablados locales, donde se origina físicamente e impacta, sino en entornos globales donde tiene su verdadera génesis explicativa, sentando las bases de una investigación global sin fronteras.

Bajo este concepto, RUISNAR, presenta los resultados de un proceso investigativo, que desde el conocimiento sistemático, la integración de saberes, trabajo en equipo, ayuda mutua, cooperación y solidaridad; permitió el logro de los objetivos de la investigación titulada: *Validación del Modelo Funcional en la Enseñanza de la Programación con el Lenguaje Scheme en el Programa Ingeniería de Sistemas,* resultados que favorecen la formación de competencias profesionales específicas y, por lo tanto, el desarrollo del conocimiento del campo de la Ingeniería de Sistemas; logrando sinergia entre lo tecnológico y lo académico, para facilitar el proceso de aprendizaje del estudiante.

Es justo reconocer en los docentes investigadores, autores de este libro, la dinámica de un motor poderoso, que impacta el campo del conocimiento, infunde vida a la universidad y genera transformación en la academia. Significativo es entonces, para la comunidad universitaria nariñense, evidenciar con productos concretos como éste, su intervención en los adelantos científicos y tecnológicos, que transforman saberes, prácticas investigativas, formas de circulación del conocimiento y la legitimación del discurso en el campo específico.

Los libros, los caminos y los días, dan al hombre sabiduría. El libro, "Un nuevo enfoque en la enseñanza de la programación", exigió recorrer muchos caminos de: conocimiento, gestión, metodología, prácticas de investigación, relaciones de grupo, toma de decisiones, actualización y capacitación en esta área, que día a día, perfilaron la meta: un libro que será leído, consultado, analizado, criticado por los docentes Ingenieros de Sistemas, desde su experiencia en la enseñanza y la programación, igualmente será referenciado en muchos trabajos académicos, en los que brindará la posibilidad de nuevas perspectivas en el campo de la programación.

Para finalizar, los miembros de la Red Universitaria de Investigaciones -UREL Capítulo Nariño, mantenemos la invitación permanente, a estudiantes, docentes e investigadores universitarios a abrir mente y corazón, para hacer de la ciencia y la tecnología, hechos humanos, que permitan participar eficazmente, en un mundo imprevisible e impredecible.

Isabel Hernández Arteaga
Coordinadora Red Universitaria de Investigaciones
Red UREL Capítulo Nariño

1

INTRODUCCIÓN

El verdadero peligro de la tecnología
no es que las máquinas lleguen a pensar
como hombres,
sino que los hombres
están pensando como máquinas.
Albert Einsten

Una actividad importante en la Ingeniería de Sistemas es el desarrollo de software, en el que se utilizan varios paradigmas, entre ellos: el imperativo con los modelos procedural y orientado a objetos, el declarativo con los modelos funcional y lógico. Sin embargo, en las instituciones de educación superior (IES) de Pasto, en las que se ofrece el programa de Ingeniería de Sistemas, no se ha aplicado más que el modelo imperativo como contenido fundamental de las asignaturas del componente de programación.

En la actualidad, existe un debate sobre la conveniencia de aplicar un enfoque estructurado, orientado a objetos o funcional para iniciar los fundamentos de la programación en Ingeniería de Sistemas.

El objetivo de este proyecto fue: aplicar y evaluar el modelo funcional utilizando el lenguaje Scheme en la enseñanza de los fundamentos de la programación en Ingeniería de Sistemas, con el fin de desarrollar en el estudiante la capacidad de entender y utilizar la recursividad, como apoyo fundamental en la solución de problemas en el campo del desarrollo de software.

1.1 UN PRIMER CURSO DE PROGRAMACIÓN

El objetivo básico de un primer curso de programación es proporcionar a los estudiantes, sin ninguna experiencia en programación, los mecanismos necesarios para enfrentarse a la creación de programas para solucionar problemas pequeños, en el marco de expresividad de un paradigma de programación, enseñándoles un lenguaje de programación, conceptos, métodos y técnicas que les permitan abordar los problemas, llegando de un modo riguroso desde la especificación al programa correcto (García, 2003).

Las cuestiones que hay que resolver son: la elección del paradigma y del lenguaje, la selección de los conceptos, métodos y técnicas, y encontrar el modo de inculcar al estudiante el razonamiento riguroso y la preocupación por la corrección (García, 2003).

A los estudiantes les resulta muy difícil el paso del enunciado del problema al programa. La programación les supone la entrada en un universo de ideas y pensamientos totalmente nuevo y es necesario proporcionarles de forma cuidadosa los elementos que les permitirán afianzar esas ideas y adquirir destreza en esa forma de razonar que posibilita crear algoritmos. Por ello, es necesario que primero se les explique la resolución de muchos problemas, para transmitirles el tipo de razonamiento que lleva del problema al esbozo de la solución. Luego, serán ellos quienes deberán enfrentarse de forma individual a ejercicios propuestos. Hay que evitar crear programadores

compulsivos, ansiosos por sentarse delante de la máquina para escribir directamente el código y establecer con el computador una batalla hasta obtener el programa que se supone calcula el resultado deseado, a través de un ciclo escribir-ejecutar-modificar. Para inculcar el rigor en los estudiantes, se les debe exigir que expresen su solución en una notación algorítmica no ejecutable, y sobre ella analicen la corrección, como paso previo a escribir el programa (García, 2003).

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.

El modelo imperativo es un paradigma de programación que describe la programación en términos del estado del programa y sentencias que cambian dicho estado. Los programas imperativos son un conjunto de instrucciones que le indican al computador cómo realizar una tarea. La implementación de hardware de la mayoría de computadores es imperativa; prácticamente todo el hardware de los computadores está diseñado para ejecutar código de máquina, que es nativo al computador, escrito en una forma imperativa. Esto se debe a que el hardware de los computadores implementa el paradigma de las Máquinas de Turing. Desde esta perspectiva de bajo nivel, el estilo del programa está definido por los contenidos de la memoria, y las sentencias son instrucciones en el lenguaje de máquina nativo del computador (por ejemplo el lenguaje ensamblador).

El problema del modelo imperativo consiste en que el estudiante desarrolla unas estructuras mentales muy secuenciales que hacen 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.

Existe otro modelo de programación denominado declarativo, en el cual se incluye la programación lógica y funcional. El objetivo del modelo funcional es conseguir lenguajes expresivos y *matemáticamente elegantes*, en los que no sea necesario bajar al nivel de la máquina para describir el proceso llevado a cabo por el programa, evitando el concepto de *estado* del cómputo. La secuencia de computaciones llevadas a cabo por el programa se regiría única y exclusivamente por la *reescritura* de definiciones más amplias que otras, cada vez más concretas y definidas.

Los programas escritos en un lenguaje funcional están constituidos, únicamente, por definiciones de funciones, entendiendo éstas no como subprogramas clásicos de un lenguaje imperativo, sino como funciones puramente matemáticas, en las que se verifican ciertas propiedades. Otras características propias de estos lenguajes son la no existencia de asignaciones de variables y la falta de construcciones estructuradas como la secuencia o la iteración (lo que obliga en la práctica a que todas las repeticiones de instrucciones se lleven a cabo por medio de funciones recursivas).

La ventaja del modelo funcional es que el estudiante desarrolla la capacidad de entender y utilizar la recursividad, como apoyo fundamental en la solución de problemas, permitiendo una sustancial disminución en líneas de código y tiempo.

Desde una perspectiva más pragmática, se puede decir que la sintaxis de un lenguaje funcional como Scheme es extremadamente simple, lo que permite que el lenguaje pueda dominarse fácilmente en sólo seis meses. Los estudiantes suelen preferirlo,

sobre todo, cuando se trata del primer lenguaje de programación que aprenden, y los instructores lo elogian porque facilita la enseñanza de ideas de abstracción y diseño de algoritmos, tan útiles para formar buenos programadores (Coello, 1996).

1.2 SOBRE ESTE LIBRO

En este documento se presentan los resultados del proyecto de investigación *Validación del modelo funcional en la enseñanza de la programación con el lenguaje Scheme en el programa Ingeniería de Sistemas*, realizado por la Red Universitaria de Investigación en Ingeniería de Sistemas de Nariño - RUISNAR, conformada por las instituciones de Educación Superior con sede en la ciudad de Pasto, capital del departamento de Nariño (Colombia): Universidad de Nariño, Universidad Mariana, Universidad Cooperativa, Institución Universitaria CESMAG y Corporación Universitaria Autónoma de Nariño, y el cual fue financiado por la Red de Universidades Regionales Latinoamericanas UREL, capitulo Nariño.

1.3 ORGANIZACIÓN DEL LIBRO

Este libro se organiza en diez capítulos, en los que el lector encontrará información detallada sobre el proceso investigativo llevado a cabo, los resultados obtenidos en cada una de las insittuciones y los valores consolidados.

El capitulo 1 presenta una visión general sobre la discusión respecto al primer curso de programación, la selección del paradigma y el lengaje, las características de los paradigmas imperativo y declarativo y algunas ventajas particulares del modelo funcional que lo hacen idóneo para comenzar la enseñanza de la programación.

El capítulo 2 describe la evolución de la programación y el problema en torno a la enseñanza de sus fundamentos. También se hace una revisión de la historia del lenguaje

Scheme y cómo ha llegado a ser común en muchos cursos de programación en universidades de todo el mundo. El capítulo se cierra con una revisión de la enseñanza de la programación en las instituciones de educación superior de San Juan de Pasto, que participan en la investigación.

El capitulo 3 hace referencia al proyecto: validación del modelo funcional y el lenguaje Scheme en la enseñanza de los fundamentos de programación, describe las fases de la investigación, los productos que se desarrollaron en cada una y los resultados obtenidos con su correspondiente interpretación.

Los capítulos 4,5,6,7 y 8 corresponden respectivamente a los informes de investigación de las instituciones participantes: Universidad de Nariño, Institución Universitaria CESMAG, Universidad Mariana, Corporación Universitaria Autónoma de Nariño y Universidad Cooperativa de Colombia sede Pasto. Cada capítulo incluye información detallada sobre los grupos experimental y de control, evaluación de conocimientos antes y después de la enseñanza de fundamentos de programación con el modelo Funcional y el lenguaje Scheme, desarrollo del curso y los resultados obtenidos.

El capítulo 9 presenta las conclusiones del estudio.

Finalmente, el capítulo 10 hace algunas recomendaciones a las instituciones de educación superior con programas de Ingeniería de Sistemas, con base en los resultados de la investigación y la experiencia de la enseñanza de los fundamentos de programación con el lenguaje Scheme.

2

ENSEÑANZA DE LOS FUNDAMENTOS DE PROGRAMACIÓN EN INGENIERÍA DE SISTEMAS

El número de genios que produce una nación
es proporcional al número de hombres
que recibe una cultura suficiente.
Stendhal

Uno de los aspectos más importantes en el programa Ingeniería de Sistemas dentro del área de formación específica, es el desarrollo de software o programación. En esta área, en la mayoría de las instituciones de educación superior (IES) de San Juan de Pasto que ofrecen este programa, dentro de su plan curricular se contempla únicamente el estudio del modelo imperativo de programación en el que se incluye la programación estructurada o procedural y la orientada a objetos.

2.1 LA ENSEÑANZA DE LA PROGRAMACIÓN

Dentro de la evolución de la programación se distinguen cuatro grandes etapas: en la primera, la programación se concibe como un arte y las principales herramientas

del programador son el lenguaje elegido y sus habilidades personales. La segunda etapa comenzó 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 Dijkstra sobre programación estructurada y de Hoare sobre tipos de datos abrieron el camino a una visión más formal de la programación, al tiempo que ofrecieron mecanismos de abstracción para dominar la complejidad. Otras ideas influyentes de esta segunda etapa son: la técnica del refinamiento por pasos sucesivos de Wirth, y los conceptos de ocultación de información de Parnas, y de jerarquía de tipos de Dahl. Mientras que en la primera etapa, la abstracción procedural 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 1990, del paradigma orientado a objetos (OO) como el más adecuado para producir software de calidad, esto es, extensible y reutilizable. Este paradigma encuentra sus raíces en la mencionada idea de Dahl, expuesta en un trabajo que describe los conceptos básicos subyacentes al primer lenguaje OO, Simula (Dromey 1982). Después vinieron otros lenguajes OO como Smalltalk, Eiffel, C++ y Java, y a partir de ellos la eclosión de la denominada tecnología de objetos.

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 surgiendo 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.

En la segunda etapa, con la programación estructurada, se reconoce al lenguaje Pascal como el más adecuado para enseñar a programar y el uso del refinamiento por pasos sucesivos como principal técnica de diseño de programas. A finales de la década de 1980 este esquema era utilizado en la mayoría de centros universitarios de todo el mundo. La programación estructurada exigía el cumplimeinto de dos principios: que el flujo de ejecución del programa coincidiera 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. Pero también propugnaba algo más profundo que pasó inadvertido para muchos: el cálculo de programas, esto es, la corrección debe guiar el proceso de creación del programa a través de una derivación formal a partir de la especificación.

En 1989, Dijkstra criticó duramente el estado de la enseñanza de la programación, ya que en su opinión se estaban formando programadores con una visión de la programación más cercana al arte que al de una disciplina sujeta a principios que permitiesen ejercerla con rigor. En su libro *A Method of Programming* Dijkstra (1984) presentó un método formal de enseñanza basado en dos principios básicos:

- Enseñar un lenguaje imperativo claro y sencillo, cuya semántica es definida por reglas de prueba.

- La principal tarea del estudiante no es escribir programas, sino dar una prueba formal de que el programa cumple la especificación.

Esta propuesta inició un intenso debate sobre el grado de formalismo en un primer curso de programación.

Por otra parte, inspirados en el clásico libro de Polya, *How to solve it?*, a principios de los ochenta surgieron algunas propuestas que consideraban la programación como una tarea de resolución de problemas. De todas ellas, la más elaborada fue la descrita por Dromey (1982) en su libro *How to solve it by computer*, en el que primero enseña un conjunto de heurísticas y técnicas de programación y luego las aplica a un conjunto de problemas clasificados en varias categorías.

Otro trabajo importante de la década de los ochenta fue el realizado por el grupo Anna Gram (1986), creado por la AFCET (Asociación Francesa de Informática) para identificar los principales razonamientos utilizados en la construcción de programas. En 1989, Scholl y Peyrin (1989) publicaron el libro *Esquemas Algorítmicos Fundamentales* en que proponen un enfoque para la introducción a la programación basado en su experiencia en el grupo. El éxito del paradigma Orientado a Objetos (OO) en la década de 1990 provocó, cómo es lógico, que surjan propuestas para introducir la programación a través de los conceptos OO.

La utilización del modelo funcional en la enseñanza de la programación comienza con las ideas de John Backus (1978) sobre la programación funcional. En su conferencia con motivo de la recepción del prestigioso premio Turing otorgado por la ACM (Association for Computing Machinery) en el año 1978, Backus se pronunció en contra del uso de variables, la sentencia de asignación y el uso de sentencias de control. Consideró la programación procedural como carente de propiedades matemáticas que permitan razonar sobre los programas y propuso abiertamente la programación funcional 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 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.

Los orígenes teóricos del modelo funcional se remontan a la década de 1930, más precisamente al año 1934, cuando Alonzo Church (1941) introdujo un modelo matemático de computación llamado *Cálculo Lambda*. A pesar de que en esta época las computadoras aún no existían el Cálculo Lambda se puede considerar como el primer lenguaje funcional de la historia y sus fundamentos fueron la base de toda la teoría de la programación funcional y de los lenguajes funcionales desarrollados posteriormente. Se puede decir que los lenguajes funcionales modernos son versiones de Cálculo Lambda con numerosas ayudas sintácticas.

Existen dos grandes categorías de lenguajes funcionales: los funcionales puros y los híbridos. La diferencia entre ambos estriba en que los lenguajes funcionales híbridos son menos dogmáticos que los puros, al admitir conceptos tomados de los lenguajes procedimentales, como las secuencias de instrucciones o la asignación de variables.

En contraste, los lenguajes funcionales puros tienen una mayor potencia expresiva, conservando a la vez su transparencia referencial, algo que no se cumple siempre con un lenguaje funcional híbrido.

Entre los lenguajes funcionales puros, cabe destacar a Haskell y Miranda. Los lenguajes funcionales híbridos más conocidos son Lisp, Scheme, Ocaml y Standard ML (estos dos últimos, descendientes del lenguaje ML).

En cuanto al lenguaje SCHEME, sus origenes se remontan al año 1975, cuando Gerald Jay Sussman y Guy Lewis Steele Jr. desarrollaron un pequeño intérprete, usando el lenguaje MacLisp, que tuviera reglas de ámbito estático para estudiar la

teoría de los actores como un modelo de computación desarrollada por Carl Hewitt en MIT. Sussman y Steele se sintieron tan satisfechos con su mini-intérprete que decidieron llamarlo *Schemer*, pensando que con el tiempo se convertiría en otro lenguaje que se pudiera utilizar en inteligencia artificial, tal y como *Planner*, el lenguaje desarrollado por Hewitt. Sin embargo, el sistema operativo ITS limitaba los nombres a 6 letras, por lo que el apelativo del intérprete hubo de ser truncado a *Scheme*, que es como se le conoce hoy en día.

En 1976, los autores de Scheme anunciaron que este intérprete soportaba eficientemente los principales paradigmas de programación de ese momento, a saber: imperativo, funcional y orientado a objetos. Steele desarrolló como tesis de maestría un compilador de Scheme llamado Rabbit (Sussman y Steele, 1978).

Históricamente, Scheme contribuyó a estrechar la brecha entre los teóricos (que estudiaban el Cálculo Lambda y el modelo de actores) y los prácticos (implantadores y usuarios de Lisp) en el área de lenguajes de programación. Además, Scheme hizo la semántica denotacional mucho más accesible a los programadores y proporcionó una plataforma operacional que permitiera a los teóricos realizar sus experimentos. Por su reducido tamaño, no había necesidad de tener una versión centralizada del lenguaje que tuviera que soportar un gran número de plataformas, como sucedía con Lisp. De tal forma, brotaron por todas partes implementaciones y dialectos de Scheme hacia inicios de la década de 1980. Un ejemplo es el Scheme 311, desarrollado en la Universidad de Indiana varios años antes de que alguien hiciera un intento por producir una implementación aislada del lenguaje, que no tuviera que montarse sobre Lisp (Fessenden et al, 1983).

Hoy en día, Scheme es un lenguaje simple, pero poderoso; pequeño, pero flexible. Su naturaleza lo hace ideal para la enseñanza, incluso como primer lenguaje de programación, por su notable facilidad para incorporar diversos paradigmas con pocas

primitivas. Como lenguaje imperativo, conserva la pureza y elegancia que le proporcionan la recursividad y las funciones de orden superior, superando así a muchos de los lenguajes que hoy gozan de gran popularidad, tales como Pascal y C. En cuanto lenguaje orientado a objetos, cuenta con un sistema de paso de mensajes y de manipulación de objetos equiparable al de Smalltalk.

Actualmente el estudio del modelo funcional especialmente con el lenguaje Scheme se ha implementado en cursos no introductorios de programación, en los currículos de universidades, tales como: Universidad de Jaén, Universidad Politécnica de Valencia, Universidad de Murcia, Universidad de Córdoba y Universidad de Alicante en España; MIT, Universidad de Indiana, Universidad de Colorado, Universidad de Syracuse en Estados Unidos; Universidad Tecnológica Nacional en Argentina, Universidad Central de Venezuela, Universidad Nacional Autónoma de México, Pontificia Universidad Católica de Chile, Universidad Nacional de Trujillo en Perú, Universidades Javeriana y del Valle en Colombia.

2.2 EL COMPONENTE DE PROGRAMACIÓN EN LAS IES DE SAN JUAN DE PASTO

Por otra parte, en los programas de ingeniería de sistemas ofrecidos, se aprecia 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 especifico.

En el programa de Ingeniería de Sistemas de la Universidad de Nariño el componente de programación inicia en el primer semestre con la asignatura *Fundamentos de programación*, continua con tres talleres de programación y termina con *Estructuras de información* en quinto semestre cuyos contenidos

contemplan el estudio de lenguajes procedurales y orientado a objetos. No se incluye el estudio del modelo funcional en el plan curricular. De acuerdo a estadísticas del año 2006, el 26% de estudiantes reprobaron asignaturas del componente de programación y su mayor dificultad está en la aplicación de la lógica en la solución de problemas.

En el programa de Ingeniería de Sistemas de la Institución Universitaria CESMAG, el componente de programación inicia en el segundo semestre con la asignatura *Modelos de programación* y termina con la asignatura *Programación de alto nivel* en quinto semestre, cuyos temas se desarrollan con lenguajes imperativos, más no se aborda el estudio del modelo funcional en el plan curricular. De acuerdo a estadísticas del año 2006, el 43.3% de estudiantes reprobaron asignaturas del componente de programación y su mayor dificultad está en la aplicación de la lógica en la solución de problemas.

En el programa Ingeniería de Sistemas de la Universidad Cooperativa de Colombia, Pasto, dentro de las áreas de formación específica se encuentra el desarrollo de software o programación. El proceso de formación en este campo se inicia con la materia *Lógica y programación* y concluye con los *Seminarios regionales* y las electivas en *programación avanzada* en noveno y décimo semestre.

Hasta el momento, el componente de programación se ha manejado con los modelos procedural y orientado a objetos del paradigma imperativo, que según estadísticas aproximadas un 40% de los estudiantes de *Algoritmia y Programación I*, tienen la necesidad de repetir dichas asignaturas, haciendo que algunos de ellos decidan retirarse del programa ante los resultados obtenidos. Su mayor dificultad se encuentra en su baja estructuración lógica para aplicar los lenguajes de programación en la solución de problemas.

En el programa de Ingeniería de Sistemas de la Universidad Mariana el componente de programación inicia en el primer semestre con la asignatura *Algoritmos y Programación I*, luego, en segundo, *Algoritmos y Programación II*, y en tercero y cuarto se continúa con *Estructuras de Datos I y II*. En estos cursos se trabaja con lenguajes procedurales. El componente termina con *Programación Orientada a Objetos* en sexto semestre. No se contempla el estudio del modelo funcional en el plan curricular, aunque en décimo semestre hace una aproximación al modelo en el espacio académico *Inteligencia Artificial*. De acuerdo a estadísticas del año 2006, el 28% de estudiantes reprobaron asignaturas del componente de programación y la dificultad se vislumbra en el análisis y diseño de algoritmos.

La Corporación Universitaria Autónoma de Nariño ofrece los programas: Ingeniería Electrónica y Tecnología en Sistemas, en estos programas el componente de programación se inicia en el primer semestre con la materia *lógica computacional*; en el programa de Ingeniería Electrónica termina con *programación orientada a objetos* en octavo semestre y en Tecnología en Sistemas, con *programación concurrente* en séptimo semestre, trabajando con el modelo procedural y orientado a objetos. Al igual que las instituciones anteriores ha experimentado poblemas en el desarrollo de aprendizaje en estos cursos, los estudiantes presentan dificultades para diseñar, modelar y dar soluciones algorítmicas. El porcentaje de estudiantes que reprueban asignaturas del componente de programación es: 50% en Ingeniería Electrónica y 40% en Tecnología en Sistemas.

Ante esta situación surge la pregunta que RUISNAR ha intentado responder ¿La enseñanza del modelo de programación funcional utilizando lenguaje Scheme, en el componente de programación, mejorará en los estudiantes de Ingeniería de Sistemas, la capacidad para desarrollar programas que den solución efectiva a problemas específicos en el campo del desarrollo de software con respecto al modelo imperativo?.