Revista Educación en Ingeniería
ISSN 1900-8260
Diciembre de 2010 • N°. 10 • Pp 49-60 • Publicada en línea por la Asociación Colombiana de Facultades de Ingeniería -ACOFI- www.acofi.edu.co
Enviado: 08/10/2010 • Aprobado: 09/11/2010

APORTES DE LA PROGRAMACIÓN FUNCIONAL AL CURRÍCULO DE INGENIERÍA DE SISTEMAS COMO PRIMER CURSO DE PROGRAMACIÓN

Omar Ivan Trejos Buriticá
Universidad Tecnológica de Pereira, Pereira (Colombia)

Resumen

Una de las discusiones más comunes entre los docentes de un programa de ingeniería de sistemas es acerca de cuál debe ser el contenido del primer curso de programación de computadores que se debe ofrecer a los estudiantes que ingresan a la carrera y cuál debe ser el primer paradigma de programación con el que deben tener contacto los estudiantes. El presente artículo proporciona una serie de argumentos para que se considere la posibilidad de incorporar el paradigma de programación funcional en el primer curso de programación de computadores en una carrera de ingeniería de sistemas. Si bien, los argumentos expuestos pueden ser debatibles, es deseo del autor que este artículo dinamice la discusión al respecto y proporcione más criterios para futuros análisis.

Palabras clave: Paradigma de programación, programación funcional, introducción a la programación, programación de computadores, currículo de ingeniería de sistemas

Abstract

One of the most common discussions among teachers of system engineering programs is what should be the contents of the first computer programming course to be offered to new students and, what should be the first paradigm the students should have contact with. This article provides a series of arguments to consider the possibility to incorporate the functional programming paradigm as a first course in computer programming for systems engineering. Although the arguments can be quite debatable, it is the wish of the author that this article invigorates the discussion and provides more criteria for future analyses.

Keywords: Programming paradigm, functional paradigm, introduction to programming, computer programming, System engineering curricula

Introducción

La discusión acerca de cuál debe ser el primer paradigma de programación con el que se deben enfrentar los estudiantes que llegan a un programa de ingeniería de sistemas es un tema recurrente entre los docentes de esta área. A lo largo de más de veinte años de experiencia y catorce libros escritos, el autor de este artículo ha recogido una serie de elementos de juicio que somete a consideración de los lectores, especialmente docentes, para que tengan un panorama un poco más aproximado de lo que es la programación funcional, su razón de ser como primer curso de programación en ingeniería de sistemas y los aportes que este cambio genera en relación con el resto de asignaturas de la carrera.

En la mayoría de programas de ingeniería de sistemas esta discusión, tan activa como ha sido, también ha traído consigo decisiones de cambio dado que el norte de la programación, desde la perspectiva de paradigma, puede ser tan móvil como se quiera. Sin embargo, teniendo en cuenta algunos elementos comunes de la carrera ingeniería de sistemas, se exponen en consideración toda una serie de argumentos que pueden llegar a unificar criterios tanto en el contenido de dicho curso como en la estructura.

Cabe aclarar que el paradigma que se seleccione como primer paradigma de programación para ser compartido con los estudiantes, marca la pauta para que esa puerta hacia el futuro se abra o, muchas veces, se cierre. No se han de desconocer otros factores que influyen en esto tales como la calidad de los docentes de la asignatura, los recursos con que se cuenten y la estructura organizativa de la universidad, entre otros; pero el presente artículo se concentrará en aportar argumentos a la discusión centrando su interés en la sugerencia de ser el paradigma de programación funcional la mejor opción para atender los requerimientos de un primer contacto de los estudiantes con el mundo de la programación.

Al respecto de los argumentos que se plantean a lo largo del artículo, se preferencia la programación funcional por tres razones que, ampliadas, son la base para proponer los elementos de juicio que aquí se exponen: en primera instancia la necesidad de entregar herramientas sólidas desde el principio de la carrera a los futuros ingenieros de tal forma que les permita escalar los conceptos.

En segundo lugar, la necesidad que puede surgir de articular en esta primera asignatura los conceptos principales que se han de asimilar y apropiar a lo largo de todo el programa de ingeniería de sistemas teniendo en cuenta que la programación de computadores constituye un pilar fundamental en el desarrollo de dichos conceptos. Finalmente, en tercer lugar, la necesidad de estructurar un pensamiento que sea tan tecnológico como metodológico, tomando por tecnológico el hecho de que el ingeniero recurra a las herramientas que el mundo moderno le proporciona y entendiendo lo metodológico como el correcto aprovechamiento y utilización de dichas herramientas.

La tecnología sin metodología ubica al futuro ingeniero de sistemas en el mundo de los usuarios rasos; metodología sin tecnología lo convierte en un excelente teorético pero con poca oportunidad de aplicación. El paradigma de programación funcional, utilizado y aprovechado adecuadamente, proporciona los elementos necesarios, aún en la etapa inicial de la formación en ingeniería como es el primer curso de programación, para que el estudiante visualice el mundo tecnológico y metodológico con el cual se va a enfrentar.

Primera aproximación: ¿Qué enseñamos?

Una de las aristas de la citada discusión estriba en la definición acerca de qué enseñar en un

primer curso de programación en un programa de ingeniería. Tres opciones salen al escenario académico como posibles candidatas temáticas: la primera que surge como inquietud es construir un contenido temático basado en lógica de programación; la segunda alternativa corresponde a la estructuración de un contenido basado en un paradigma de programación y la tercera opción es la enseñanza de un lenguaje de programación (Hughes, 1984).

Aunque parecieran tan cercanas, las tres opciones tienen diferencias significativas que posibilitan tres orientaciones diferentes en el curso de introducción a la programación mencionado. La lógica de programación va hacia la fundamentación matemática general de la computación y su relación con el pensamiento matemático aplicado a la programación de computadores. Cuando se pretende estructurar un curso orientándolo hacia la lógica de programación, el profesor ha de tener tanto una buena fundamentación matemática como claridad al respecto de la aplicación y el significado de los conceptos que se involucran.

La lógica de programación va hacia el buen aprovechamiento de las matemáticas como base lógica general para la posterior aplicación en ambientes computacionales. Hablar de lógica de programación implica hablar de lógica matemática y esto no va ligado a ningún lenguaje de programación. También implica tener claros los principios metodológicos científicos que permitan hacer que la lógica pueda ser vista tanto en su dimensión como en su significado para que se pueda articular con posteriores aplicaciones computacionales.

Una de las ventajas de iniciar a los estudiantes nuevos en la programación de computadores a través de un curso de lógica de programación radica en que pueden conocer y apropiar una gran cantidad de conceptos que, posteriormente, podrán ser aplicables a cualquier paradigma de programación dado que éstos se fundamentan en las matemáticas y es precisamente ésta, la que constituye un componente fundamental necesario para el aprendizaje de la programación de computadores.

Como desventaja se podría pensar en que el panorama que adquiere el nuevo estudiante puede llegar a ser tan amplio que se puede convertir en una serie de conceptos etéreos frente a los cuales podría no tener suficiente claridad como para llevarlos al mundo de las aplicaciones computacionales.

Cuando el curso se orienta hacia un paradigma de programación entonces se está pensando en seleccionar un área específica de la amplia lógica de programación (heredada de la lógica matemática) para ir hacia sus fundamentos matemáticos específicos. Incluirá una claridad conceptual, por parte de los docentes, suficiente como para poder relacionar posteriormente el paradigma con la aplicación y permitir que ambos, armonizados, se muevan dentro de un mismo contexto.

Se necesitará que los docentes conozcan y apropien adecuadamente tanto las bases matemáticas del paradigma como sus aplicaciones y el espectro completo de los lenguajes de programación que se relacionan. De la misma manera, será útil que al menos uno o dos docentes del cuerpo profesoral de esta área, cuenten con experiencia con algún lenguaje de dicho paradigma.

Una de las ventajas que podríamos destacar cuando un curso introductorio de programación se orienta hacia un paradigma específico de programación, es que los conceptos pueden ser mucho más cercanos al estudiante. Sin embargo, el riesgo que se tiene es que el curso se desfigure y termine orientándose mucho mas al lenguaje que al paradigma lo cual no es del todo malo pero tal reorientación si no es planeada puede causar dificultades de aprendizaje y aplicación en el estudiante.

La tercera opción que se puede tener en la planeación de un curso introductorio de progra-

mación, es pensar en que esté orientado hacia un lenguaje de programación. El poder cautivador que tiene un lenguaje de programación no lo tiene ningún otro aspecto temático. En tal caso, se ha de considerar que los docentes deben conocer de fondo el lenguaje de programación. Sin embargo, la gran desventaja que presenta esta opción radica en el hecho de que, por lo cautivadora que es, el estudiante puede llegar a ser, con una buena conducción, un experto en ese Lenguaje de Programación pero desconociendo la fundamentación matemática y lógica que subyace al mismo.

¿Es necesario que el estudiante conozca dicha fundamentación matemática y lógica? El autor de este artículo considera que si el estudiante conoce bien los fundamentos matemáticos y lógicos de un paradigma de programación, entonces podrá obtener mucho más provecho del lenguaje de programación asociado a dicho paradigma.

Surgen varias preguntas asociadas a la inquietud presentada hasta ahora en esta primera aproximación: ¿Se sabe qué se está enseñando en los cursos introductorios de programación? ¿Se sabe si los contenidos introductorios de programación están orientados hacia la lógica de programación, hacia un paradigma de programación o hacia un lenguaje de programación? ¿Los docentes son conscientes de la diferencia?

Son estas preguntas las que se deberían responder para poder entrar en la discusión. Ahora bien, no podría establecerse que una de las tres posibilidades sea estrictamente mejor que las otras dos pero lo que sí se podría afirmar es que, para el proceso de aprendizaje de los estudiantes nuevos que llegan a un programa de ingeniería de sistemas, será mucho más enriquecedor no sólo que los docentes tengan claro cuál es la orientación de los contenidos de sus cursos de programación sino que la compartan y la hagan efectiva.

En este sentido vale la pena señalar que muchas veces los docentes de una misma asignatura la imparten con diferentes criterios; unos lo hacen orientándola hacia la lógica de programación, otros hacia el paradigma de programación y otros hacia el lenguaje de programación y, muchas veces no son conscientes de dicha diferencia, ni tienen una comunicación académica fluida que les permita detectarla. Es el estudiante quien finalmente recibe tal suerte de confusiones que, en estos términos, resultan ser poco enriquecedoras.

Lo anterior conduce a presentar unos antecedentes de la programación funcional que pueden permitir hacer unas inferencias para tener más elementos de juicio en la discusión que está planteada.

Antecedentes de la programación funcional

Los primeros pasos orientados a lo que actualmente se conoce como la programación funcional los estableció el matemático y lógico norteamericano Alonzo Church. Su trabajo más conocido fue el desarrollo del llamado cálculo lambda, una matemática que permitió el aprovechamiento y estructuración del concepto de función, y fue en el año 1936 cuando demostró la existencia de problemas indecidibles, es decir, problemas que no pueden ser resueltos con un algoritmo incluso si se llega a disponer de tiempo y espacio ilimitado.

Se cuentan dentro de estos problemas, el que establece que dada una frase de cálculo de predicados de primer orden, decidir si ella es un teorema. Por su parte, Alan Touring, alumno de Church, demostró claramente que éste es uno de estos problemas. Fue precisamente el cálculo lambda el que influenció el diseño de lenguajes como LisP y lo que posteriormente se conoce como programación funcional.

Posteriormente, hacia los años sesentas, John McCarthy formuló el lenguaje de programación LisP basado en los conceptos tomados de las

teorías de Church y Touring. McCarthy hacia el año 1955 acuñó el término "inteligencia artificial" y también fue el primero en sugerir públicamente la oferta de servicios a partir de un sistema computacional compartido, algo impensable para sus tiempos.

El Lenguaje de Programación LisP es una familia de lenguajes de programación basados en el paradigma de programación funcional. Actualmente pertenecen a esta familia Common Lisp y *Scheme*. En primera instancia, el LisP (List Processing) fue creado como una notación matemática práctica para los programas de computador que tomaba como base el cálculo lambda de Church. Una de las características de la sintaxis del Lenguaje Lisp es su particular intercambiabilidad del código y de datos. Todo el código se escribe con expresiones S o sexp que no son más que estructuras de datos basadas en listas escritas entre paréntesis.

Durante la misma década de los sesenta, Peter Landin formuló un lenguaje experimental de programación llamado ISWIM que es el acrónimo de "If you See What I Mean" mejorando en su planteamiento algunos de los conceptos incorporados por sus antecesores y haciendo uso de la programación funcional. Este lenguaje nunca fue implementado como tal pero su formulación, matemática muy bien organizada, le hizo ser fuerte influyente en el desarrollo posterior de la programación funcional. Como sucesores directos de este lenguaje, pueden contarse SASL, Miranda y ML.

Hacia la década de los setenta, John Backus (protagonista en la formulación del lenguaje de programación fortran y una de las mentes más brillantes de IBM) se interesó en la programación funcional y realizó toda la formulación de un lenguaje al que llamó FP (Functional Programming) y con el cual quería demostrar que era posible construir lenguajes de programación que se liberaran del modelo de Von Newmann. Este fue un lenguaje puramente académico que se transformó en el lenguaje FL y que fue terminado hacia el año 1991.

Durante los mismos años setentas, Robin Milner (científico británico en computación) desarrolló un marco teórico para el análisis de sistemas concurrentes y, con ello, pudo formular el cálculo de sistemas comunicantes y su sucesor, el cálculo pi. Esto le permitió, basado en los avances que se habían logrado al respecto de la programación funcional, formular un lenguaje de programación que denominó ML (Meta Lenguaje). Este lenguaje fue concebido para desarrollar tácticas de demostración en el sistema LCF que es una técnica para demostración automática de teoremas.

ML resultó ser, con el tiempo, un lenguaje de programación funcional impuro o híbrido dado que permitía la utilización de los conceptos funcionales propios de su paradigma pero también permitía el uso de instrucciones del paradigma imperativo. En la actualidad Ocaml tal vez sea el más común de los lenguajes de programación derivados de ML. En la misma década en que se dan los aportes de Robin Milner con su lenguaje de programación ML, se presenta un lenguaje orientado a la programación lógica por parte de Colerauer y Rousell. Inicialmente éste era un proyecto que tenía por objetivo el procesamiento del lenguaje natural, sin embargo, poco a poco fue tomando la envergadura de todo un proyecto y se convirtió con el tiempo en el lenguaje Prolog.

Inicialmente Prolog era un lenguaje interpretado pero solo fue hasta 1983 cuando David Warren desarrolló un compilador capaz de traducir Prolog en un conjunto de instrucciones de una máquina abstracta. Las versiones iniciales de este lenguaje eran diferentes unas de otras, en cada una de sus implementaciones, especialmente en lo que se refería a su sintaxis sin embargo hacia el año 1995 se promulgó la norma ISO/

IEC 13211-1 llamada Normal ISO-Prolog que permitió que las versiones tuvieran un estándar de referencia para su construcción.

Hacia los años ochentas, David Turner propuso un lenguaje de programación al cual denominó Miranda, cuyo objetivo es obtener una versión comercial de un lenguaje funcional que fuera no-estricto y con características puramente funcionales. Por primera vez el paradigma funcional salía del contexto académico en el cual se había incubado y desarrollado para abrirse paso en el mundo comercial.

Este lenguaje contempló una gran cantidad de características de la programación funcional en su estado original que son las que permiten que el desarrollar software sea realmente sencillo y ágil. A pesar de que en el medio colombiano no ha sido común el lenguaje de programación Miranda, un programa resulta muy fácil de concebirse ya que consiste en un conjunto de declaraciones en donde prima la recursividad como elemento fundamental y los datos de tipo algebraico.

Precisamente hacia el año 1987, la madurez del lenguaje de programación Miranda llevó a consolidar una propuesta mucho más elaborada y que capitalizara mejor las características de la programación funcional. Esta propuesta se llamó Haskell en honor al apellido de su creador Haskell Curry. El lenguaje de programación Haskell fue el resultado de unificar criterios acerca de las diferentes versiones de lenguajes funcionales que existían en el mercado en el momento (Van Roy, 2003).

Derivado de estos aportes, a partir del paradigma de programación funcional irrumpe en el mundo académico el lenguaje de programación *Scheme* que, si bien fue formulado en la década de los setenta, su presencia actual en el mundo académico permite abrir horizontes más prácticos y eficientes en relación con la utilización de otros lenguajes de programación.

El objetivo del lenguaje *Scheme* consiste en no acumular grandes unidades funcionales sino construir aplicaciones sólidas a partir de unas funciones que cumplan escasamente su objetivo pero que, enlazadas apropiadamente, se conviertan en un conjunto lógico suficiente para resolver los problemas que se generen.

Al igual que su padre LisP, *Scheme* ofrece una serie de estructuras básicas del lenguaje apoyado en listas y no cuenta con una sintaxis explicita para crear registros o estructuras o, específicamente, para desarrollar programación orientada a objetos. Sin embargo la mayoría de implementaciones ofrece, independientemente, estas funcionalidades.

Entre las ventajas de *Scheme* se puede contar que tiene una sintaxis significativamente reducida si se compara con la mayoría de los lenguajes comerciales. El uso de la notación prefija posibilita la no existencia de ambigüedad en las expresiones y abre un camino de resolución mucho más seguro. *Scheme* está construido para facilitar la programación funcional pura aunque por ser un lenguaje híbrido permita la utilización de algunas instrucciones que alteran dicho paradigma tal como la instrucción set!. También permite crear procedimientos anónimos.

Para definir el lenguaje *Scheme* en la actualidad existen dos estándares: el oficial promulgado por IEEE y un estándar conocido como Revised Report on the Algorithmic Language *Scheme*, abreviado normalmente como RnRS en donde n es el número de la revisión. En la actualidad la revisión más popular es R5RS y la más actualizada es R6RS aún no popularizada. Una de las herramientas más generalizadas para la utilización del Lenguaje *Scheme* es el entorno DrScheme desarrollado por el MIT.

Cuatro aportes se solidificaron al punto de convertirse en los grandes conceptos que nutren los elementos de juicio expuestos: a) el concepto

de función, b) el concepto de modularidad, c) el esquema de una función y d) el esquema funcional, que a continuación se detallan.

El concepto de función

Constituye el núcleo de la programación funcional así como lo es la célula para el ser humano o la familia para la sociedad. Una función es, en palabras sencillas, un pequeñísimo programa que a) cumple con un objetivo específico y puntual, b) tiene un nombre identificativo, c) recibe argumentos para cumplir con su objetivo y d) normalmente devuelve un resultado.

La función es el alma de la programación funcional y a partir de ella se pueden construir aplicaciones del tamaño que se quiera simplemente enlazando varias funciones en el orden en que la lógica de solución de un problema indique. Por ejemplo, si se desea construir una función que calcule la suma de dos argumentos que deben ser diferentes de cero entonces una posible solución podría ser la siguiente:

```
;; Función que calcula el resultado de una suma
( define (suma a b )
        ( if ( and (not= a 0) (not= b 0) )
                (+ a b )
                0
        ) )
```

En este brevísimo ejemplo, las cuatro características de la función son absolutamente claras: a) el objetivo de la función es calcular el resultado de una suma, b) el nombre que identifica la función es suma, c) los argumentos que recibe esta función para cumplir con su objetivo se han llamado a y b y d) el resultado que devuelve la función será la suma de a + b en el caso de que a y b sean diferentes de cero o devolverá el valor 0 en el caso en que tanto a como b o uno de los dos sea igual a 0.

Invirtiendo el sentido de esta explicación se podría pensar entonces en que si previamente se tienen claras las partes de una función, cualquier función será fácilmente alcanzable. A continuación un ejemplo: Construir una función que permita calcular el área de un triángulo. En este caso, se van a determinar las cuatro características antes de construir la función.

De esta forma se tiene que: a) el objetivo de la función es calcular el área del triángulo (que sabemos que está dado por la semisuma de multiplicar la base por la altura, es decir, *(base x altura)/2*, b) el nombre identificativo que a otorgar a esta función es *Area_Triangulo*, c) los argumentos que necesita esta función para poder calcular el área de un triángulo corresponderán a un valor para la base y otro valor para la altura y d) el valor que debe devolver esta función deberá ser el resultado de reemplazar en la fórmula el valor de la base y la altura y hacer el cálculo respectivo.

Lo único que se adicionará a estos elementos es que se validará la base y la altura para que no tengan valores iguales a cero, dado que si en un triángulo su base o su altura son iguales a cero significa que no hay triángulo. De esta manera, entonces una propuesta funcional de solución a lo planteado podría ser:

```
;; Función para calcular el Área de un Triángulo
( define (Area_Triangulo base altura)
        ( if ( and (not= base 0) (not= altura 0) )
                ( / ( * base altura ) 2 )
                0
        ) )
```

Se observa en esta solución que la función se define a través de la instrucción *define*, que tanto los condicionales como las expresiones se escriben en notación prefija, que en caso de que la condición sea verdadera la función retornará el valor de calcular la fórmula del triángulo pero

en caso de que la condición sea falsa la función retornará el valor 0. Ha de prestarse particular atención a la buena colocación y utilización de paréntesis.

Finalmente debe quedar claro que a una función la puede llamar otra función y que el valor que retorna la función llamada es lo que recibe la función llamadora. También es posible, acudiendo al concepto de recursión, que una función se llame a sí misma pero eso será tema de otros artículos.

El concepto de modularidad

La modularidad consiste en que un programa se construya a partir de la interrelación entre funciones de manera que cada función sea independiente pero a la vez interdependiente, esto es, que cada función cumpla con un objetivo específico pero que en conjunto puedan cumplir con un objetivo mayor acudiendo a aquello de que la suma de las partes deberá ser superior que las partes por separado (Felleisen, 2003).

De esta forma, el enunciado de construir una función que calcule el área de un triángulo podría convertirse en construir un programa que calcule el área de un triángulo. La diferencia es que como se va a construir un Programa y no una sola Función, entonces se necesitan por lo menos dos funciones.

A continuación se caracterizarán cada una de las funciones:

- Primera función: a) su objetivo será el de recibir los argumentos y verificar que dichos argumentos sean diferentes de cero. En caso de que uno o los dos argumentos sean iguales a cero deberá desplegar un aviso, en caso contrario deberá llamar a la función que realizará el cálculo, b) el nombre de esta primera función será lectura, c) tendrá dos argumentos llamados base y altura que son los que posteriormente le pasará a la segunda función, y d) devolverá el valor que reciba de la segunda función.
- Segunda función: a) su objetivo será el de calcular el valor del área del triángulo basado en la fórmula convencional que la geometría proporciona para tal fin, b) se llamará *Area_Triángulo*, c) como argumentos recibirá el valor de la *base* y la *altura* para realizar el cálculo y d) devolverá como resultado el valor de calcular el área del triángulo.

Visto así, una propuesta de solución a lo planteado podría ser la siguiente:

```
;; Función que recibe los argumentos, los valida
e invoca a la función que calcula el área
( define (lectura base altura)
      ( display "Digite valor para la base...>")
      ( set! base (read) )
      ( display "Digite valor para la altura..>")
      (set! altura (read) )
      ( if ( and ( not= base 0 ) ( not= altura 0 ) )
          (Area_Triangulo base altura)
          ( display "Error en la entrada de datos")
      ) )

;; Función que calcula el área del triángulo
( define (Area_Triangulo base altura)
          ( / ( * base altura ) 2 ) )
```

Como puede notarse la validación de datos se realizó en la primera función y allí mismo se estableció el mensaje de error en caso de que los argumentos sean iguales a cero. La segunda función realiza el cálculo del área del triángulo.

Esquema general de una función

A partir de estas reflexiones, y concibiendo la función como ese núcleo principal del paradigma de programación Funcional, puede pensarse en que toda función tiene una estructura muy

sencilla que bien puede describirse fácilmente de la siguiente forma:

Una función es un pequeño programa formado por las siguientes partes:

a. **Objetivo.** Constituye la razón de ser de la función. Una función existe si tiene un objetivo concreto y definido. La función debe alcanzar escasamente el objetivo que lo define la necesidad del problema que se quiere resolver. El objetivo es lo que realmente hace útil a la función.
b. **Nombre.** El nombre de la función es el identificador que diferencia una función de otra y que permite invocarla de manera única. En esencia, el nombre pretende darle una identificación propia a la función para que ésta pueda ser llamada en cualquier momento.
c. **Argumentos.** Son los valores que hacen que la función "funcione". Son equivalentes a los ingredientes de una receta y por tanto son necesarios no solo para poder llamar una función sino para que ésta logre su objetivo. Los argumentos los establece el problema y, en algunos casos, requiere participación activa del programador.
d. **Validaciones.** Son las restricciones que deben verificarse en relación con los argumentos que han de alimentar la función. Las validaciones las determina el programa a partir del análisis que el programador haga del objetivo.
e. **Proceso.** Es el camino que hemos establecido a partir de nuestra lógica para que la función pueda alcanzar el objetivo tomando como insumos los argumentos que le llegan cuando es invocada. El proceso lo establece el problema y lo diseña el programador.
f. **Resultado.** Es el objetivo logrado a partir de la ejecución de la función. El resultado es la presencia misma de la función cuando esta es invocada y es la cristalización del logro del objetivo. El resultado se evalúa con las necesidades del problema y debe confrontarse para saber si se ha cumplido con él o no.

Esquema funcional

Finalmente, se define el esquema funcional como el gráfico que representa no sólo a las funciones de manera independiente y con todos sus componentes, sino la relación entre ellas de manera que, en conjunto, sean la solución al problema que se haya planteado. Para ejemplificarlo, podría decirse que el esquema funcional es la fotografía de la Modularidad y que el Esquema de una Función es la fotografía de ella.

Problemas que aborda la programación funcional

Uno de los propósitos más expeditos que justifica el estudio de la programación funcional corresponde a los tres problemas que resuelve por sus características funcionales (Trejos, 2002):

a. Simplificación de objetivos. Por las razones que la modularidad establece, una función siempre se podrá concebir como una unidad bastante simple y, en caso de que dicha unidad sea más compleja de lo pensado, entonces se procede a subdividirla y, con ello, a construir funciones que, enlazadas, la reemplacen consiguiendo su objetivo original por un camino más simple.
b. Detección y corrección ágil de errores lógicos. Una de las competencias más difíciles de lograr por los programadores es la detección y corrección ágil de errores lógicos. Dado que las funciones tienen una estructura tan similar, entonces la corrección de dichos errores resulta fácil.
c. Reutilización del código. Consiste en que las funciones, por ser unidades independientes, pueden ser usadas en otras aplicaciones tal y como fueron concebidas originalmente. La reutilización del código es uno de los problemas mejor resueltos por el paradigma de programación funcional.

Programación funcional y requerimientos de un programa

Si se analiza una función detenidamente se podría decir que es un símil, a menor escala, de un programa, de una aplicación o de un proyecto. Se puede establecer una relación entre las características de una función (objetivo, nombre, argumentos, validaciones, proceso y resultados) y las mismas características de un programa, de una aplicación o de un proyecto.

Ha de suponerse entonces que acostumbrar al cerebro a concebir la solución de un problema a partir de estas características en pequeña escala, tal como es el caso de la función, permitirá posteriormente extrapolar dichos conceptos y encontrarlas en proyectos y problemas a mayor escala (Trejos, 2004).

Análisis y discusión de resultados

A lo largo de 6 semestres de prueba, revisión y decantación de haber adoptado el paradigma de programación funcional y el lenguaje de programación *Scheme* (bajo el entorno DrScheme), se pueden establecer algunas inferencias sobre la base de tener más de veinte años de experiencia en la enseñanza de la programación de computadores y varios libros escritos.

La realización de pruebas parciales es otro aporte importante de este paradigma dado que para verificar la efectividad de un programa en el logro de su objetivo no se tiene que hacer una agotadora prueba completa sino que solamente se prueban las funciones sobre las cuales se tiene alguna. Dado que las funciones son siempre unidades simples y sencillas, las pruebas también son sencillas, confiables y fáciles de realizar. Una de las dificultades que tiene la programación es la realización de pruebas agotadoras dado que muchas veces son tan complejas que dificultan la comprensión de la solución. El pensamiento que se deriva de la modularización hace más sencilla la comprensión del paradigma funcional.

Entre las posibles desventajas que se puedan contar con la apropiación de este paradigma es que a partir de la experiencia que se adquiere con el paradigma de programación funcional, todo se quiere llevar al plano de las funciones. El concepto de función y su gran utilidad en la construcción de aplicaciones puede ser aplicable a cualquier paradigma de programación.

Las funciones (núcleo de la programación funcional), cuando se han asimilado y apropiado, permiten entender la programación funcional en toda su plenitud, facilitan el aprovechamiento de los recursos con que cuenta la programación estructurada y simplifican la concepción de la programación orientada a objetos, dado que un objeto es, en palabras simples, un conjunto de atributos junto con un conjunto de métodos.

Los atributos pueden asociarse con el concepto de los argumentos, elementos esenciales para el buen desempeño de las funciones, y los métodos no son más que las mismas funciones. De manera que asimilar y apropiar el concepto de función simplifica el aprovechamiento de los recursos de estos otros dos paradigmas. *Scheme* permite que se hagan procesos estructurados y se apliquen los principios de la POO.

Un proceso inverso también es posible, sin embargo la experiencia ha demostrado mayor efectividad y menos dificultad cuando el primer paradigma visto es el de la programación funcional. Es allí en donde vale la pena abrir la discusión para que el colectivo docente pueda llegar a las que considere las mejores conclusiones para sus estudiantes.

Una de las aplicaciones modernas del paradigma de programación funcional y de su filosofía asociada consiste en el moderno Work Breakdown Structure que no es más que el desglose sistemático de las operaciones que conforman el todo de un proyecto con sus respectivo análisis

independiente e interdependiente por unidades de trabajo de manera que cada una sea medible y cuantificable. Esta definición no es más que el escalamiento de la programación funcional y de sus conceptos asociados, aplicados a la administración y gerencia de proyectos.

Ahora bien, en relación con las preferencias de pensamiento derivadas del modelo cerebral 4Q formulado por Ned Hermann en donde plantea la estructura del cerebro a nivel de habilidades cognitivas de alto nivel, como la unión de cuatro cuadrantes que el autor nominó con letras: el cuadrante A se ocupa de todo lo Lógico, el cuadrante B se ocupa de lo Secuencial, el cuadrante C se ocupa de lo Interpersonal (lo social) y el cuadrante D se ocupa de lo Imaginativo (Lumsdaine, 1994).

Se quiere dejar en claro que el paradigma funcional requiere la utilización activa de los cuatro cuadrantes dado que el cuadrante lógico es el que permite la construcción de una solución a partir de las funciones, el cuadrante secuencial permite la estructuración de la misma a partir de la relación entre las funciones, el cuadrante interpersonal posibilita el enriquecimiento de la solución a través de la interacción con otros estudiantes y el cuadrante imaginativo facilita la búsqueda creativa de la solución.

Otro factor es la relación entre el paradigma funcional y el aprendizaje significativo (Ausubel, 1986). El simple hecho de que el paradigma funcional posibilite ejemplos que sean aplicables tanto al mundo tecnológico como a la vida misma y que estos puedan ser planteados con las mismas características de análisis, permite que el estudiante permanentemente encuentre ejemplos prácticos que aproximen los conceptos con el aprendizaje.

Dos pensamientos más se fortalecen con el paradigma de programación funcional: el pensamiento matemático (a lo largo de las cinco aristas que incluyen sus competencias) y el pensamiento funcional. Como pensamiento matemático se consideran todos aquellos conceptos y elementos de juicio que, relacionados directamente con la matemática y derivados de ella, fortalecen la lógica en el ser humano para la construcción de una posible solución a un determinado problema.

Como pensamiento funcional se establece la posibilidad de modular la solución de cualquier problema (computacional o no) de forma que siempre se simplifique su construcción y puesta en marcha. Es de anotar que este pensamiento tiene aplicación tanto en el mundo de la programación de computadores como en la vida misma dado que cuando se simplifica un problema y se puede subdividir en módulos más simples, hallar la solución se hace sencillo.

Conclusiones

Con la discusión presentada y los elementos de juicio que se someten a consideración de los lectores es posible llegar a unas conclusiones más sólidas en cuanto a la escogencia del contenido de la primera asignatura de programación de una carrera de ingeniería de sistemas y al perfil de lo que se quiere enseñar.

La programación funcional, como paradigma, presenta unas ventajas y desventajas que son notorias y que bien pueden ser controvertibles, sin embargo, el presente artículo deja a consideración de los lectores un espectro más amplio de argumentos para que se enriquezca la discusión y se puedan llegar a conclusiones más generales y óptimas en este tema. Queda una expectativa amplia, por parte del autor, en relación con acceder a artículos similares que destaquen cualquier otro paradigma y realicen una comparación con el nivel de detalle con el que aquí se presenta el paradigma funcional.

Este artículo es la posición a favor del paradigma de programación funcional. A lo largo del tiempo,

la experiencia ha permitido que el autor compare tres paradigmas: el estructurado, el funcional y el orientado a objetos como primer paradigma que se imparte en un programa de ingeniería de sistemas y puede dar fe de que los resultados con el paradigma funcional han sido superiores en diferentes cursos, jornadas, metodologías y grupos de estudiantes.

En este campo no se puede desconocer la importante labor que tienen los docentes frente a la motivación que se pueda generar en los estudiantes que llegan a primer semestre de ingeniería de sistemas y, por ello, la simple escogencia de un paradigma o del perfil de la asignatura no es suficiente.

Referencias

Ausubel, David. (1986). Sicología educativa: Un punto de vista cognoscitivo, ISBN 968-24-1334-6, Editorial Trillas, México, pp. 46-85

Felleisen, Mathias et al. (2003). How to design programs: An introduction to computing and programming, The MIT Press, Septiembre, pp. 232

Hughes, John (1984). ¿Por qué es importante la programación funcional? Institutionen för Datavetenskap, Chalmers Tekniska Högskola, pp. 10

Lumsdaine, Edward et al. (1994). Creative problem solving: Thinking skills for a changing world, McGraw Hill, 978-0070390911, pp. 229

Trejos B., Omar Ivan. (2002). La esencia de la lógica de programación, Universidad Tecnológica de Pereira, Centro Editorial Universidad de Caldas, 958-33-1125-1, pp. 120 - 152

Trejos B., Omar Ivan, (2004). Fundamentos de Programación, Universidad Tecnológica de Pereira, Editorial Papiro, 958-8236-10-x, pp. 39

Van Roy, Peter. (2003). Concepts, technics and models of computer programming, Swedish Institute of Computer Science, Junio, pp. 135

Sobre el autor

Omar Ivan Trejos Buriticá

Ingeniero de Sistemas y Computación, Especialista en Instrumentación Física, Magister en Comunicación Educativa, Candidato a PhD en Ciencias de la Educación. Docente de Planta, Facultad de Ingeniería, Ingenieria de Sistemas y Computación, Universidad Tecnológica de Pereira (La Julita).
omartrejos@utp.edu.co

Ultimas publicaciones

Libro Fundamentos de Programación. Noviembre 2005, ISBN 958-8236-10-X, 152 p.

Libro Algoritmos: Problemas Básicos. Diciembre 2005, ISBN 958-8236-13-4, 110 p.

Libro Diálogos con papá, Agosto 2009, ISBN 978-958-44-4709-8, 315p

"Aplicación de la Programación Funcional al cálculo de una función trigonométrica", Revista Scientia et Technica, UTP, Año XVI, No. 45, Agosto 2010