

Usando el Paradigma Funcional en los cursos introductorios de programación

Alfredo Paz-Valderrama Abigail Parisaca Vargas
Eliana Adriazola Herrera

10 de abril de 2006

Resumen

En este artículo se hace una propuesta para la implementación de los primeros cursos del área de computación bajo el enfoque de funcional primero. Este enfoque fue reconocido en la ACM Computing Curricula 2001 para una carrera de 4 años de duración, sin embargo, en nuestro país las carreras equivalentes tienen una duración de 5 años, lo que nos brinda la oportunidad de corregir deficiencias en la transición del nivel secundario al superior, en especial en el área de formación matemática.

1. Introducción

El uso del enfoque funcional primero fue inicialmente propuesto e implementado en el MIT [ASS85] y fue finalmente reconocido en la ACM Curricula 2001 [Cur01], como una propuesta exitosa. Sin embargo, la ACM Curricula fue diseñada pensando en una carrera de cuatro años de duración y para un alumnado con una mejor formación matemática [Pis03]. Pese a que en los últimos años, este enfoque ha sido progresivamente reemplazado por el orientado a objetos [Ric03] consideramos que el enfoque funcional primero podría ser especialmente útil en nuestro contexto, aportando el fundamento matemático y la capacidad de abstracción necesaria para llevar con éxito cursos más intensivos en el área de computación.

El artículo inicia con una explicación del contexto en el que se desarrolla la propuesta, para luego hacer una exposición de los objetivos que se deberían

alcanzar en los primeros cursos de programación, también se hace un análisis de las ventajas y desventajas de los enfoques “tradicionales” utilizando los paradigmas imperativo-orientado a objetos y el funcional; a continuación se hace una reflexión sobre los problemas que presentaría el uso de los lenguajes más populares en la implementación de la propuesta y finalmente, se enuncia nuestra propuesta tanto en la distribución de contenidos por curso, como en la elección de los lenguajes.

2. Contexto

En los primeros cursos de programación, el tema central suele ser el lenguaje, en lugar de los conceptos computacionales. De otro lado, los alumnos ingresantes poseen un deficiente desarrollo del área matemática [Pis03], lo que afecta directamente sus capacidades de resolución de problemas y abstracción.

Segun lo explica [KR01], concentrarse en el lenguaje era difícil de evitar, cuando se usaba lenguajes orientados a objetos o imperativos debido a la complejidad de su sintaxis. La opción del enfoque funcional, sin embargo, parecía carecer de este problema, aún cuando se usó el lenguaje nativo solamente.

Según [DH94] la opción de cambiar al enfoque funcional primero, conlleva algunos temores:

1. Nuestro escaso dominio del paradigma, comparado con nuestra experiencia en lenguajes imperativos y orientados a objetos
2. El no estar familiarizados con las herramientas del lenguaje (compilador, interprete, IDE)
3. Nuestra naturaleza orientada al objeto, fruto de los años de uso de esta tecnología.
4. La falta de experiencias previas en el medio.

3. ¿Qué se busca lograr en los primeros cursos de programación?

De acuerdo con [Mar99], independientemente del paradigma usado, un estudiante de computación debería adquirir las siguientes habilidades en los primeros cursos de programación :

- Capacidad de solucionar problemas computacionales: entenderlos, diseñar una solución y ser capaz de implementarla.
- Capacidad de abstracción y el rol que cumple ésta en lidiar con la complejidad del software, aprender a modelar problemas.
- “Separar las propiedades lógicas de los datos o acciones de sus detalles de implementación”
- Deberían empezar a desarrollar una base de conocimiento sobre al menos dos técnicas de diseño.
- Algoritmos desde el punto de vista del diseño y del análisis
- Aprender un lenguaje moderno: Diseño (Ingeniería del Software) y su implementación.

De otro lado para [DH94], los estudiantes deben ser capaces de producir programas que sean:

- Modulares
- Legibles
- Traceables
- Mantenibles

y también deben tener la capacidades de [DH94]:

- Analizar
- Especificar
- Diseñar

- Manejar la complejidad de programas de tamaño modesto.

En la ACM Curricula [Cur01], los objetivos de aprendizaje son mostrados con mayor detalle, nuestra propuesta será basada en estos.

4. Posibles enfoques

La ACM Curricula 2001 [Cur01], propone seis posibles enfoques para la enseñanza de los cursos introductorios de una carrera de ciencias de la computación¹. tres de ellos proponen empezar con programación:

- Imperativo primero
- Objetos primero
- Funcional primero

y otros tres que proponen no empezar con programación:

- Una visión panorámica
- Algoritmos primero
- Hardware primero

Aunque el enfoque imperativo primero es el más tradicional y el orientado a objetos es uno de los más usados en la actualidad, nuestra propuesta será usar el enfoque funcional primero como introducción al enfoque orientado a objetos, aprovechando la similitud de conceptos.

5. El enfoque funcional primero

Este enfoque empezó a usarse en la década de los 80 por el MIT, para lo cual se desarrolló un lenguaje de programación funcional basado en LISP (Scheme). Sin embargo, no fue hasta la publicación del libro *Structure and Interpretation of Computer Programs* [ASS85], cuando la propuesta empezó a ganar popularidad, de hecho este se convirtió en uno de los libros más vendidos del *MIT Press*.

¹Cabe resaltar que la ACM Curricula está pensada para carreras de 4 años de duración.

Según se muestra en [Cur01] “...un curso introductorio bajo el enfoque funcional primero, debería ser seguido por un curso intensivo, donde se cubrán los conceptos de diseño y programación orientada al objeto.”

A continuación se mostrarán las principales ventajas y desventajas del enfoque, en nuestra experiencia.

5.1. Ventajas del enfoque

Las principales ventajas del enfoque se resumen en [Cur01]

- El uso de un lenguaje poco conocido en el medio, reduce los problemas de la diversidad en los conocimientos previos.
- La sintaxis minimalista permite concentrarse en los conceptos.
- Algunos conceptos importantes como recursión y polimorfismo, pueden ser cubiertos de una manera más natural y temprana.

Se considera que a las ventajas anteriores, se pueden agregar las siguientes, dada nuestra realidad y contexto.

- Permite cubrir conceptos matemáticos abstractos de manera práctica, como las demostraciones y recursión principalmente.
- Pone en evidencia, de forma temprana, la relación entre los cursos matemáticos y su aplicación en la computación.

En nuestro contexto, estas ventajas resultan muy importantes, debido a los problemas que presenta el área de formación matemática en las escuelas secundarias [Pis03].

5.2. Desventajas del enfoque

En [Cur01] se resaltan principalmente dos desventajas:

- El probable excepticismo de los estudiantes que ya tienen alguna experiencia con lenguajes imperativos.
- El enfoque exige un alto nivel de abstracción de forma muy temprana.

En nuestra experiencia, el primer problema no fue difícil de superar, fue bastante sencillo mostrar a los estudiantes que el enfoque funcional enriquecía sus conocimientos, incluso a quienes tenían alguna experiencia en el desarrollo de software. Suponemos que esto se debe a que en el medio, mayormente, la industria del software aún produce software de manera artesanal.

En cuanto al segundo inconveniente, se tuvo que hacer esfuerzos para intentar coordinar el trabajo hecho en los cursos del área matemática. Consideramos que este fue el mayor problema en la implantación de la propuesta, pero como se discutirá más adelante, el rescatar el pensamiento abstracto fue uno de los principales aportes del enfoque.

Aparentemente la implantación del enfoque acarrea otros problemas además de los mencionados, ya que este fue progresivamente dejado de lado al finalizar la década de los 90, sediendo terreno frente al enfoque orientado a objetos primero. Una descripción de esto se puede encontrar en [FFFK04] o en [Ric03]. Según estas experiencias los principales problemas en la implantación del enfoque fueron:

- Deficiencias del lenguaje Scheme [FFFK04]
- La problemática transición del paradigma funcional al imperativo [Ric03]
- La elección de el(los) “siguiente(s)” lenguaje (s). [Ric03]

La experiencia de [Ric03] nos pareció de especial interés. En ella, los autores intentan implementar una secuencia de tres cursos introductorios empezando con el paradigma funcional (usando scheme), para luego seguir con el paradigma orientado al objeto usando primero Java y finalizar con C++ en el tercer semestre. El principal problema que experimentaron fue que al final de la secuencia, los estudiantes no tenían el dominio necesario en ninguno de los lenguajes, para empezar los cursos intermedios. Esto, debido a que, semestre a semestre, el tiempo necesario para aprender la nueva sintaxis, tomó mucho más tiempo del esperado. De esta experiencia se extrae una conclusión muy importante:

Cada nuevo trozo de sintaxis requiere un tiempo de aprendizaje
que, potencialmente, podría crecer de manera exponencial.

6. El enfoque Orientado a Objetos primero

La [Cur01] propone dos posibles implementaciones del enfoque: usando una secuencia de dos o tres cursos. En cualquiera de las dos implementaciones, el curso debería empezar con la noción de objetos y herencia. Al igual que en el caso del enfoque funcional primero, a continuación expondremos sus principales ventajas y desventajas.

6.1. Ventajas del enfoque

Como se resalta en la [Cur01] “la principal ventaja del enfoque, es la temprana exposición a la programación orientada al objeto, la cual está ganando importancia tanto en la academia, como la industria”. Esto podría implicar una notable reducción en el tiempo de aprendizaje dedicado a la sintaxis, en especial si el resto de cursos fueran dictados usando el mismo lenguaje.

6.2. Desventajas del enfoque

Como se expone en diversos artículos [KR01], [CDP03], [GTD⁺98] y en la misma [Cur01], el principal problema del enfoque está en la complejidad de la sintaxis de los lenguajes de programación comúnmente usados (Java y C++). Esto debido a que estos lenguajes no fueron diseñados para la educación.

Este problema puede provocar que el curso se aleje de los conceptos computacionales, para concentrarse en los asuntos de la sintaxis necesarios, para implementar correctamente conceptos tales como entrada/salida o incluso herencia. La abundancia y popularidad de bibliografía especializada en la enseñanza del lenguaje (más que en los conceptos relevantes para la computación) fomentan mayor confusión.

En nuestra experiencia, otro problema importante fue la dificultad que presentaba establecer de forma clara la importancia de la matemática en la ciencia de la computación (computación). Conceptos como álgebra, conjuntos, dominio y rango de una función no tienen una implementación nativa clara y directa.

7. Problemas con la elección de los lenguajes

La implementación de la estrategia funcional primero, requiere que se elija un lenguaje funcional y algunos otros que soporten el paradigma orientado al objeto. En los siguientes apartados se argumenta las desiciones tomadas en nuestra propuesta.

7.1. ¿Scheme o Hugs?

Algunos autores como [FFFK04], afirman que Scheme sólo parece funcionar en el MIT, porque posee una excelente calidad de alumnos. El mismo [FFFK04] analiza algunos problemas de Scheme:

- La programación funcional con Scheme es muy distinta a otras formas de programación
- El lenguaje Scheme no prepara adecuadamente, a los estudiantes, para los cursos avanzados
- Scheme por sí solo es un lenguaje “débil”, para sus uso en un curso introductorio requiere de la construcción de una biblioteca.

Al ser Haskell un lenguaje más moderno que Scheme, sus diseñadores tuvieron cuidado de corregir los problemas señalados por lo que se muestra, comparativamente, más prometedor.

7.2. La elección del siguiente lenguaje

Como se analiza en [DH94], no existen motivos para usar el paradigma imperativo en los primeros cursos de programación, cuando el orientado a objetos es capaz de cubrir un mayor y más completo espectro. Por esto, nuestro análisis se centrará en los lenguajes orientados a objetos más usados: C++, Java, C#.

La elección del lenguaje ha utilizar para introducir el paradigma Orientado al Objeto, se hará siguiendo el mismo principio general:

Hay que centrarse en los contenidos y elegir el mejor lenguaje para darlos

A continuación analizaremos algunos de los problemas que se podrían presentar, al elegir C++ o Java como lenguaje.

7.2.1. Deficiencias de C++

La siguiente descripción se basó en el artículo [BAA03], en el cual se cubre con mayor detalle las deficiencias, nosotros sólo resaltaremos aquellas que fueron más relevantes en nuestra experiencia.

- Las referencias. “la referencia” (&) usa el mismo símbolo que “la indirección” (&). La confusión causada por esta homonimia puede tardar muchos años en ser aclarada. En nuestra experiencia la única solución a este problema está en manos de las habilidades pedagógicas del profesor del curso.
- Error en la implementación de getline. Se requiere presionar dos veces la tecla **Enter** para que el texto sea leído. Esto sólo está presente en visual C++ 6.0
- Tratamiento no uniforme para las constantes de tipo caracter.

```
int n='99'
cout<<"n = "<<n<<endl;
/* Imprimirá:
   n = 14649
   resultado del intento de convertir al ASCII 57 ('9')
   a un valor entero
   57*256+57 */
```

- El problema del getline vacío.

```
cin>>precio;
getline(cin, producto);
// producto siempre será null
```

- El procesamiento del cout. presente en GNU/Linux, también.

```
#include<iostream>
using namespace std;
main(){
```

```

        int x=9;
        cout<<x<<" "<<x++<<" "<<x<<endl;
    }
    /* Podría imprimir:
       10 9 9 */

```

- Fácil confusión entre los operadores de asignación y de comparación. Muchos compiladores no reportan ningún error.
- No es posible concatenar dos literales de tipo string.
- No es posible comparar dos literales de tipo string con el operador de comparación.

```

if("hola" == "adios")
    return 1;
//El compilador no reporta ningún error.

```

- No hay chequeo de los índices de un arreglo.

7.2.2. Deficiencias de Java

Java es un lenguaje relativamente joven, comparado con C++, dentro de sus objetivos de diseño siempre estuvo la seguridad², por lo que es de suponer que se corrigieron muchos de los “problemas” de C++, ya descritos. Sin embargo, al igual que C++, Java no fue un lenguaje diseñado para la educación sino para la industria y algunas de sus características no son adecuadas para un primer curso de programación. La experiencia de [GTD⁺98] al usar Java como lenguaje, nos muestra algunas de ellas:

- Los estudiantes no mostraron dificultades en **crear y usar** objetos pero, aparentemente, no entendían claramente el concepto de objeto ya que no eran capaces de conseguir distinguir qué era un objeto y qué no.

²Entendida como la ayuda que podría proporcionar el compilador para evitar errores de programación

- Los estudiantes mostraron serias dificultades en distinguir la declaración de un método del uso de este, en particular el principal problema fueron los parámetros.

Todo esto resultó coherente con nuestra propia experiencia, sin embargo tenemos la esperanza que estos problemas se muestren en un grado mucho menor, después de la experiencia previa con el paradigma funcional.

7.2.3. Deficiencias de C#

Una experiencia en el uso de este nuevo y prometedor lenguaje es analizada en [Reg02], en ella se mencionan algunas importantes características del lenguaje:

- Aunque C# se anuncia como un “mejorC++”, realmente es más parecido a Java.
- Al igual que en el caso de C++ y Java, C# ofrece buenas soluciones para los problemas de programación conocidos de Java. Esto sería especialmente en los primeros cursos de programación.
- C# posee una sintaxis más compleja que Java.

Esta última característica nos hizo desistir del uso de este lenguaje, además de todo lo que implicaría usar una tecnología propietaria por tradición.

8. Nuestra propuesta

En base a los argumentos previamente expuestos, se propone usar una secuencia de cuatro cursos introductorios bajo el enfoque funcional primero como introducción al paradigma de la orientación a objetos, extendiendo la propuesta de la ACM Curricula 2001 para una carrera de 5 años.

A continuación se describen los contenidos y lenguajes elegidos para la implementación de la propuesta.

8.1. Contenidos

Para la implementación de la propuesta, la tabla1. muestra una posible distribución de los contenidos, para los cursos introductorios de computación, usando la terminología de la ACM curricula 2001 [Cur01]. Al igual que las tablas desarrolladas en [Cur01] para las estrategias imperativo primero y orientado a objetos primero la tabla1 establece un número de horas³ mínimo (no un máximo o definitivo) de dictado por tema.

Functional-first Topics approach	CS111F	CS101O	CS102O	CS103O	Total
DS5. Graphs and trees	0.5	0.5	3	2	6
PF1. Fundamental programming constructs	6	8	2	4	20
PF2. Algorithms and problem-solving	3	2	2		7
PF3. Fundamental data structures	4	4	6	12	26
PF4. Recursion	4	2	2	5	13
PF5. Event-driven programming		1	3		4
AL1. Basic algorithmic analysis		1	1	2	4
AL2. Algorithmic strategies	1	1	1	3	6
AL3. Fundamental computing algorithms	2	2	3	5	12
AL5. Basic computability	0.5			1	0.5
PL1. Overview of programming languages	2	2		1	5
PL2. Virtual machines	1	1	1		3
PL4. Declarations and types	1.5	2	1		4.5
PL5. Abstraction mechanisms	1	2	2		5
PL6. Object-oriented programming	1	8	4	8	21
PL7. Functional programming	3				3
SP1. History of computing	0.5	0.5			1
SE1. Software design	2	2	2		6
SE2. Using APIs	1	3	1		5
SE3. Software tools and environments	2	1			3
SE5. Software requirements and specifications	1	2	2		5
SE6. Software validation		1	1	1	3
Total de horas base por curso	37	46	37	44	

Cuadro 1: Distribución de contenidos, para una estrategia funcional primero en cuatro semestres

³Cada hora indicada equivale a 50 minutos de dictado

Al no haberse encontrado documentación sobre experiencias similares, la tabla1 es sólo producto de nuestra experiencia en el dictado de los cursos introductorios.

8.2. Los lenguajes elegidos

Como se explica en la experiencia de [Ric03], el usar un lenguaje distinto en cada curso introductorio (Scheme, Java y C++), puede traer algunos inconvenientes:

- Lejos de lograr una amplitud de conocimientos, sólo se consigue un insuficiente dominio del lenguaje.
- Al finalizar los cursos introductorios, los alumnos sólo tienen un semestre de experiencia en cada lenguaje.
- El tiempo necesario para enseñar una nueva sintaxis en cada semestre, resulta ser excesivo en relación a los conceptos dictados.

Con la intención de evitar estos inconvenientes, se propone que el uso del lenguaje Java en el segundo y tercer curso introductorio; mientras que Haskell y C++ serían utilizados en el primer y cuarto curso introductorio, respectivamente. Para tal efecto se han reservado horas en el cuarto curso introductorio para el aprendizaje de la sintaxis. Se espera que los alumnos ganen suficiente dominio del lenguaje Java durante un año, como para que el aprendizaje de C++ sea menos problemático que en el caso de [Ric03].

9. Conclusiones

Nuestra experiencia en el uso de la propuesta ha mostrado que el enfoque funcional primero es una buena alternativa para nuestro medio, donde los alumnos ingresante muestran serios problemas en los fundamentos matemáticos y las carreras tienen una duración mayor a la esperada por la ACM Curricula.

Se espera que la prolongación de los cursos introductorios a cuatro semestres, permita que los estudiantes logren un buen dominio del lenguaje (Java) y obtengan un mejor desempeño en el aprendizaje de cualquier otro lenguaje.

Referencias

- [ASS85] Harold Abelson, Gerald Jay Sussman, and Julie Sussman. *Structure and Interpretation of Computer Programs*. MIT Electrical Engineering and Computer Science Series. MIT Press, Cambridge, Massachusetts, 1985.
- [BAA03] Joe Bergin, Achla Agarwal, and Krishna Agarwal. Some deficiencies of c++ in teaching cs1 and cs2. *SIGPLAN Not.*, 38(6):9–13, 2003.
- [CDP03] Stephen Cooper, Wanda Dann, and Randy Pausch. Teaching objects-first in introductory computer science. In *SIGCSE '03: Proceedings of the 34th SIGCSE technical symposium on Computer science education*, pages 191–195, New York, NY, USA, 2003. ACM Press.
- [Cur01] The Joint Task Force on Computing Curricula. Computing curricula 2001. *J. Educ. Resour. Comput.*, 1(3es):1, 2001.
- [DH94] Rick Decker and Stuart Hirshfield. The top 10 reasons why object-oriented programming can't be taught in cs 1. In *SIGCSE '94: Proceedings of the twenty-fifth SIGCSE symposium on Computer science education*, pages 51–55, New York, NY, USA, 1994. ACM Press.
- [FFFK04] M. Felleisen, R. B. Findler, M. Flatt, and S. Krishnamurthi. The structure and interpretation of the computer science curriculum. *Functional and Declarative Programming in Education*, 2004.
- [GTD⁺98] Amy Gale, Ewan Tempero, Gill Dobbie, Linton Miller, Peter Andrae, and Robert Biddle. Surprises in teaching CS1 with java. Technical Report CS-TR-98/9, School of Mathematical and Computing Sciences. Victoria University of Wellington, january 1998.
- [KR01] Michael Kölling and John Rosenberg. Guidelines for teaching object orientation with java. In *ITiCSE '01: Proceedings of the 6th annual conference on Innovation and technology in computer science education*, pages 33–36, New York, NY, USA, 2001. ACM Press.

- [Mar99] William Marion. Cs1: what should we be teaching? In *ITiCSE-WGR '99: Working group reports from ITiCSE on Innovation and technology in computer science education*, pages 35–38, New York, NY, USA, 1999. ACM Press.
- [Pis03] Team Pisa. Literacy skills for the world of tomorrow - further results from pisa 2003. Technical report, UNESCO, 2003.
- [Reg02] Stuart Reges. Can c# replace java in cs1 and cs2? In *ITiCSE '02: Proceedings of the 7th annual conference on Innovation and technology in computer science education*, pages 4–8, New York, NY, USA, 2002. ACM Press.
- [Ric03] Brad Richards. Experiences incorporating java into the introductory sequence. *J. Comput. Small Coll.*, 19(2):247–253, 2003.