Enseñanza de la programación de computadoras para principiantes: un contexto histórico

Gonzalo Martín Rodríguez Carrillo[1]

Recibido: Septiembre 09 de 2014 ***Aprobado:*** Diciembre 12 de 2014

Resumen:
Una de las problemáticas vigentes en la educación es la selección, ajuste y justificación de las didácticas adecuadas para el desarrollo de los cursos. En algunos programas académicos de ingeniería el interés se concentra en los cursos introductorios de programación de computadoras, puesto que en estos se presentan altos índices de deserción y pérdida académica, situación dramática debido a que son precisamente estos cursos los que proporcionan la fundamentación básica para los diferentes programas académicos. En la selección de la metodología existen requerimientos que se deben satisfacer, por ejemplo: el énfasis del programa académico, el perfil de formación, la integración y comunicación con la industria. Con respecto a la metodología, algunos autores priorizan el lenguaje de programación inicial como factor determinante en la apropiación de los conceptos, mientras que otros autores enfatizan en el ambiente de desarrollo (IDE por sus siglas en Inglés). Como resultado de esta revisión se encontró que actualmente existe una tendencia hacia la elección de lenguajes de poco impacto en la industria, tales como Python así como el empleo de IDE orientados hacia la lúdica. Una elección adecuada requiere la identificación de aspectos tales como el currículo, el contexto de desarrollo del programa, la formación del estudiante, los recursos disponibles en el aula, la apropiación de las herramientas por parte del docente, y por supuesto, la disposición del estudiante.

Palabras clave: Deserción, Didáctica, Educación, IDE, Lenguajes de programación, Metodologías de enseñanza.

Abstract:
One of the current education problems is the selection, fit and proper justification of didactics for the development of the courses. In some engineering programs the focus is given by introductory courses of computing programming, because these present the higher rates of desertion; this represent a problematic situation since these courses provides the basic foundation which are essential for different academic programs. For the methodology selection there are requirements to be met, such as: the main emphasis of the program, the education profile, the integration and communication with the industry. Some authors prioritize the initial programming language as a determining factor in the appropriation of concepts, while others emphasize the integrated development environment (IDE). In this review we found that there is a trend towards choosing computer languages with a low impact on the industry such as Python, as well as those IDE oriented to ludic. A suitable choice requires the identification of a number of variables such as the curriculum, the context of the program development, the student training, the available resources in the classroom, the appropriation of the tools by the professor, and the available to the student, of course.

Keywords: Desertion, Didactics, Education, IDE, Programming languages, Teaching methodologies.

[1] Matemático, docente del Programa de Tecnología en Informática – UNIMINUTO Sede Principal.

I. INTRODUCCIÓN

Los cursos introductorios de programación de computadoras se han convertido en una problemática para las facultades responsables de diseñar y proponer los planes de estudio, debido a la inclusión de nuevos factores involucrados en los procesos de enseñanza, que deben ser considerados en el momento de abordar la estructuración de tales planes [2]. En este artículo se exploran diversas estrategias y metodologías, propuestas y aplicadas, como métodos alternativos para la enseñanza de la programación de computadoras; las metodologías exploradas coinciden en fijar el paradigma de programación, esto es, se orientan hacia los contenidos y las temáticas aunque, en general, permiten la flexibilidad suficiente para ser aplicadas en cualquiera de los niveles educativos que se considere. La exploración realizada permitirá confrontar, evaluar, proponer y apropiar elementos que se puedan incluir en la metodología que se seleccione para facilitar el proceso de enseñanza-aprendizaje en este campo, en particular, cuando los grupos de estudiantes provengan de distintas áreas del conocimiento, siguiendo la tendencia actual. Para el desarrollo de este trabajo se exploraron artículos de investigación e informes referenciados desde las bases de datos Scopus y DOAJ, empleando las palabras clave "teaching programming", considerados desde el año 2000. La razón de esta fecha viene justificada por el auge del paradigma orientado a objetos, así como por el surgimiento de metodologías y propuestas alternativas de enseñanza de la programación. Algunos resultados no se consideraron en la revisión debido a que presentaban propuestas no generalizables así como trabajos aplicables a contextos muy particulares.

Este documento está estructurado por secciones así: la sección 1 incluye la introducción, metodo y estructura del artículo, la sección 2 explora los paradigmas de programación como hilo conductor tradicional en la enseñanza de la programación de computadoras, la sección 3 contextualiza algunos de los factores discutidos, presentando los orígenes de la computación; estos permiten identificar el perfil inicial de los programadores. La sección 4 presenta una revisión de los planes curriculares y conocimiento esperado que debe manejar un estudiante de ciencias de la computación. La sección 5 presenta las propuestas pedagógicas significativas que evidencian los casos de éxito en la aplicación de las mismas en cursos tradicionales así como para los grupos heterogéneos. Finalmente, en las secciones 6 y 7 se presentan las conclusiones y referencias correspondientes a esta revisión.

II. PARADIGMAS

Durante la estructuración de un curso preliminar en programación de computadoras, e independientemente de la pertinencia de la temática, es necesario identificar factores que influyen en el éxito (medido por la baja pérdida académica) del mismo. Así por ejemplo, en [18] se identifican el lenguaje de programación, las herramientas a emplear, la pedagogía a seguir y el aporte al currículo. Uno de estos factores es el paradigma de programación a emplear, pudiendo elegir entre los siguientes:

- **Paradigma imperativo**: se basa en el modelo de cómputo de la máquina de Turing, cobija los lenguajes de programación más ampliamente conocidos y empleados en la industria de desarrollo de software tales como: ***Basic***, ***C***, ***C++***, ***Fortran***, ***Php y Java***.
- **Paradigma funcional**: cuyo fundamento se encuentra en el modelo de cálculo lambda, que es similar al trabajo con expresiones matemáticas; algunos lenguajes de programación que siguen este paradigma son entre otros: ***Scheme*** (un popular dialecto de ***Lisp***) y ***Haskel***.
- **Paradigma declarativo**: está basado en el desarrollo de especificaciones o restricciones del problema a solucionar, representantes de este paradigma son los lenguajes ***Prolog*** y ***Maude***.

Estos paradigmas definen además categorías o subdivisiones, como el paradigma imperativo que clasifica los lenguajes como estructurados (lenguaje ***C***) y orientados a objetos (***C++***, ***Java***); o el paradigma declarativo que divide a los lenguajes en lógicos (lenguaje ***Prolog***) y algebraicos (***Haskell***, ***Standard ML***).

Actualmente algunos lenguajes de programación incluyen soporte a especificaciones no nativas, de tal manera que son considerados como multiparadigma, tal es el caso del lenguajes ***C++*** el cual permite el paradigma funcional, así como el lenguaje ***Java*** que lo ha incorporado recientemente (desde la versión 8), mientras que otros lenguajes han sido construidos ofreciendo soporte multiparadigma, como es el caso del sistema ***Maude***[2]. La elección del paradigma sugiere una delimitación previa de otros elementos

[2] http://maude.cs.illinois.edu/

tales como el tipo de formación a la que se orienta el proceso. Claramente, este es uno de los primeros factores a considerar en el contexto latinoamericano, en donde se tiene una marcada diferencia en los niveles de formación (técnico, tecnológico o profesional). Adicionalmente, la metodología a considerar debe incorporar una serie de factores externos como son:

- La heterogeneidad del grupo (formación previa).
- Lineamientos educativos nacionales e institucionales,
- La demanda o requerimiento local e internacional de los profesionales.

Junto a esta amplia gama de consideraciones influyen algunas tendencias, en particular los modelos pedagógicos en los que se inscriben los docentes involucrados en la formación, así como la orientación y formación previa de los mismos docentes.

Uno de los factores más importantes soportados recientemente, consiste en un fenómeno creciente en los cursos introductorios de programación de computadoras: se trata de la inclusión en los grupos de estudiantes de diversas áreas del conocimiento, tales como estudiantes de ciencias naturales, ingeniería, ciencias sociales y ciencias de la salud entre otros. Esto implica que los lineamientos tradicionales, que tienen como eje central la identificación de unos objetivos de formación concretos basados en una línea del conocimiento específica, deben ser revaluados. Un excelente ejemplo de este enfoque se encuentra en [16], que extiende los cursos iniciales hacía el planteamiento de todo el curriculo proyectado para una formación específica: La Bioestadística. Esta tendencia resulta muy natural, debido al impacto tecnológico actual, así es indispensable que un programa académico de cualquier área del conocimiento incorpore en su plan de estudios al menos una asignatura de programación de computadoras, y esto es equiparable a la necesidad de incluir un segundo idioma en los planes curriculares de las diferentes disciplinas de formación, razón por la cual resulta fundamental definir y garantizar que las metodologías empleadas en los procesos de enseñanza sean eficaces.

III. CONTEXTUALIZACIÓN

3.1 Breve historia de la computación

La programación de computadoras tiene profundas bases históricas pese a ser una actividad relativamente joven; ésta se desarrolla sobre un lenguaje de programación que adquiere sentido sobre un dispositivo electrónico, de manera que revisar la programación de computadoras es equivalente a explorar el desarrollo y surgimiento de los equipos de cómputo, siendo estos "solamente" la implementación de un modelo conceptual propuesto, por lo general el modelo de Turing.

3.2 David Hilbert y el modelo de Turing

En el Congreso Internacional de Matemáticos en el año 1900 celebrado en la ciudad de París, se plantearon los lineamientos que debían seguir las matemáticas durante el siglo XX. El matemático David Hilbert (Figura 1) propuso 23 problemas a resolver; el segundo de estos enunciados exigía: "demostrar la consistencia de un sistema axiomático".

Figura 1. David Hilbert.
Fuente: http://www.famousscientists.org/david-hilbert/

Figura 2. Alan Turing. Fuente: http://www.petshopboys.co.uk/

Aunque todos los elementos del párrafo deben definirse para los no especialistas, podemos obviar esta tarea indicando que el enunciado es equivalente al formular el problema de: "Definir un algoritmo que determine, para todo enunciado del sistema, si este es verdadero".

La respuesta a este enunciado fué dada en el año 1930 por el matemático Kurt Gödel, como negativa. Gödel ideó un metasistema en donde logró demostrar lo que se conoce como "la indecibilidad de los sistemas axiomáticos", esto es, "si un sistema es lo suficientemente amplio, no es posible demostrar que solo contiene proposiciones verdaderas".

Demostrada la propuesta de Hilbert, el esfuerzo se centró en formalizar el concepto de "Algoritmo", presente tanto en el problema de Hilbert como en desarrollo de Gödel, el llamado "Entscheidungs-problem" o "problema de la parada", que se traduce en determinar en un modelo de cómputo si este detiene su proceso o nó, para una entrada. Es así como dos matemáticos, a partir del trabajo de Gödel, Alan Turing (Figura 2) y Alonzo Church, formularon dos modelos de cómputo independientes pero equivalentes, conocidos respectivamente como el Modelo de la máquina de Turing y el cálculo Lambda.

La máquina de Turing emula el cómputo mecánico realizado por un ser humano (Figura 3). Pese a lo que se sugiera el nombre y la misma gráfica, la máquina de Turing no es un dispositivo mecánico ni electrónico, es un formalismo (esquema o modelo conceptual), que define una cinta infinita, un conjunto de estados, una función de transición y una cabeza lectora-escritora.

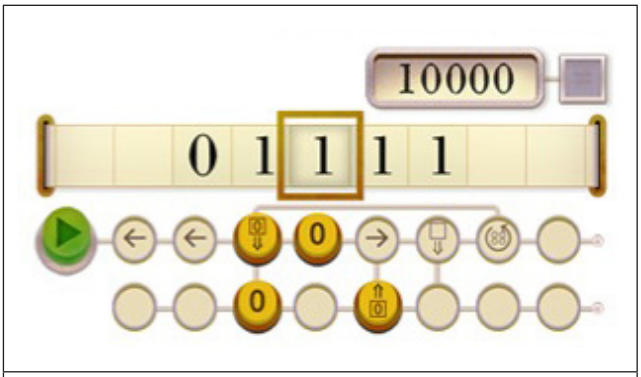


Figura 3. Esquema de una máquina de Turing
Fuente: Doodle de una máquina de Turing presentado por la empresa Google como homenaje en el centenario de Alan Turing

El cálculo Lambda por su parte, es un modelo conceptual que toma los conceptos "función matemática" y "recursión", para formalizar el modelo de cómputo.

De estos modelos, se implementó la máquina de Turing, dando origen al desarrollo tecnológico de los dispositivos de cómputo (computadoras u ordenadores) actuales, de forma paralela al desarrollo tecnológico surgieron, muy naturalmente, métodos de adiestrar o "programar" a estos dispositivos; estos evolucionan de tal manera que se requiere una clasificación de los mismos, dando origen a los diferentes paradigmas de computación, en donde el imperativo resulta ser el más cercano al modelo de Turing, tomando los conceptos de memoria y registros, presentes en los lenguajes de programación.

3.3 Perfiles y lenguajes

Dado el antecedente, la programación de computadoras inicia naturalmente como una actividad restringida a un grupo de profesionales con un perfil claramente definido: una fuerte formación en matemáticas y electrónica. Las tareas estaban bien delimitadas y se reducian al entrenamiento de un autómata en el desarrollo de un algoritmo (secuencia de instrucciones organizada lógicamente para realizar un trabajo), los equipos en los que se ejecutaban estos algoritmos eran costosos y voluminosos por lo que estaban restringidos a centros de enseñanza superior, empresas bancarias y sectores especializados. Los problemas a resolver en este periodo eran de corte científico, tales como el desarrollo de simulaciones y cálculos estadísticos en el ámbito del análisis numérico. Este primer acercamiento es, quizá, el culpable del estigma posterior para los cursos de programación como difíciles, ya que implican un conocimiento especializado para su aprovechamiento.

Los primeros cursos de formación toman así el nombre del área de la ciencia en donde surgen, llamados cursos CS (*computer science*), asi tenemos:

- CS1: Curso introductorio.
- CS2: Curso de algoritmia básica, búsquedas y arreglos.
- CS3: Curso de estructuras de datos.

Estos cursos están diseñados, ofrecidos y orientados por y hacia especialistas. Los lenguajes de programación iniciales adicionan un nivel de dificultad a estos cursos, debido a que los equipos deben programarse sobre el mismo lenguaje de la máquina, con eventualmente alguna "estandarización" aplicable solamente a unos pocos miembros de una familia de máquinas; este estándar es el lenguaje ensamblador o "assembler". Así por ejemplo, en un ensamblador moderno como NASM, la impresión del popular programa "hola mundo" presentada en la Tabla 1, sugiere la complejidad a manejar:

Tabla 1: “hola mundo” en NASM	Tabla 2: “hola mundo” en C

section .data
mensaje: db 'Hola Mundo',10
sz: equ $-mensaje
section .text
global _start
_start:
mov eax,4;
mov ebx,1;
mov ecx,mensaje;
mov edx,sz;
int 80h;
mov eax,1;
mov ebx,0;
int 80h;
	#include<stdio.h>
	int main()
	{
	printf(“Hola mundo”);
	return 0;
	}

Fuente: El autor	Fuente: El autor

Este segmento de código enfatiza lo poco intuitivo que resulta este tipo de programación, en contraparte con la versión en lenguaje *C*[3] (Tabla 2).

3.4 Masificación: nuevos actores

El surgimiento de nuevas tecnologías de transferencia de información permite reducir el costo de los sistemas de cómputo, lo que redunda en la masificación de los mismos y por ende en la inclusión de otras áreas del conocimiento en la programación de computadoras. Es así como los centros de estudio empiezan a evidenciar la necesidad de masificar la enseñanza de la programación; naturalmente, estos nuevos actores no tienen la experticia conjunta en electrónica y en matemáticas, como sus predecesores. Esta masificación permite atraer a expertos de otras áreas del conocimiento, en donde los métodos y los sistemas de cómputo tienen una amplia acogida. Surge así una demanda adicional de profesionales en programación de computadoras con conocimiento de un negocio específico, situación que plantea un nuevo reto a la Universidad: proponer los currículos que permitan involucrar especialistas. En [16], los autores presentan un ejemplo de esta tendencia, que se concreta en un currículo incluyendo materias como Bioestadística, así como un linea que emplea especialistas en biología, estadística y naturalmente ciencias computacionales. Esta masificación contribuye también al desarrollo de los lenguajes de programación, estos son sistemas con una sintaxis amigable al programador, como el caso del lenguaje ***C*** (Tabla 2), y se crean además lenguajes especializados conocidos como lenguajes de propósito específico.

Se llega a la presentación de la pregunta orientadora de este artículo: *¿Cómo iniciar los cursos de programación de manera tal que permitan la inclusión de grupos heterogéneos de estudiantes*? Este cuestionamiento no es novedoso en educación, puesto que cuando un área de conocimiento alcanza un nivel de aplicabilidad impregnando a otras áreas de conocimiento, empieza a ser incluida como conocimiento transversal y requerido en las disciplinas respectivas, caso de las matemáticas, la estadística y la ingeniería, entre otras. Este cuestionamiento tampoco es ajeno a las entidades reguladoras; en [4], se deja abierta la necesidad de trabajar “... otros curriculos como necesidades para disciplinas emergentes...” en computación. A continuación se mencionan dos de las tendencias que se presentan comunmente y son: a) Preparar exclusivamente a especialistas, este es un enfoque técnico en donde los grupos heterogéneos se deben adaptar a un lineamiento común, y b) Diseñar estrategias de inclusión tanto para especialistas como para no especialistas, orientado hacia lograr un aprendizaje individual.

IV. PLANES CURRICULARES

4.1 Referentes internacionales

Una regulación “de facto” y referente internacional para los curriculos de computación, lo consituye el trabajo propuesto por la “Association for Computing Machinery” (ACM) en conjunto con el “Institute of Electrical and Electronics Engineers” (IEEE). Este trabajo se organiza según el nivel u orientación específico, generando las siguientes áreas:

- Ciencia computacional (*Computer Science*)
- Ingeniería de Computación (*Computer Engineering*)
- Sistemas de Información (*Information Systems*)
- Ingeniería de Software (*Software Engineering*)

Cada área tiene su respectiva propuesta o “Curricula”. La Tabla 3 a continuación presenta los referentes asociados con cada una de estas áreas.

[3] El enlace: http://www.roesler-ac.de/wolfram/hello.htm presenta una recopilación del programa “Hola mundo” en diversos lenguajes de programación.

Tabla 3: Estructura curricular en computación según ACM-IEEE			
Área	**Orientación**	**Documento**	**Referente**
Computer Science (CS)	Software	Computer Science Curricula 2013	[4]
Computer Engineering (CE)	Hardware	Computer Engineering 2004	[5]
Information Systems (IS)	Negocio	IS 2002	[11]
Information Technology (IT)	Negocio	IT 2006	[IT2006]
Software Engineering (SE)	Software	Integrated Software & Systems Engineering Curriculum	[9]
Fuente: El autor			

Debido a la orientación, este trabajo sugiere cursos, contenidos y distribución de los mismos, estructurados por área de conocimiento y nivel de enseñanza. Un primer paso consiste en ubicar el área de influencia correspondiente. Aunque el trabajo presenta sugerencias sobre el camino que debe transitar un profesional, el mismo incluye sugerencias sobre los contenidos a dominar en los cursos introductorios como referente para este documento. Los lineamientos sugeridos por ACM suponen, muy naturalmente, un conjunto de estudiantes con una formación puntual.

4.2 Enfoque pedagógico: referentes internacionales

Infortunadamente los mecanismos de instrucción en la iniciación a la programación de computadoras han tenido un desarrollo muy limitado con respecto al mismo avance tecnológico, pese a la identificación de la problemática: inicialmente la metodología aplicada sigue el modelo de la escuela pedagógica tradicional, un modelo por imitación, en donde el instructor propone un problema y desarrolla e implementa su solución, esperando que el estudiante lleve este desarrollo a su propio contexto, rememorando las aplicaciones iniciales y el poder de cómputo de los equipos que se programaban. Este acercamiento surge muy naturalmente: los grupos tienen una formación similar y el contexto de aplicación también es homogéneo. Se destaca en estos inicios la importancia dada a la sintaxis de un lenguaje de programación, sobre la resolución de problemas. Varios estudios han probado la deficiencia de esta metodología en la enseñanza actual [12], pese a esto, es importante notar que algunos lenguajes de programación han surgido como un método de aprendizaje en programación de computadoras, tal es el caso del lenguaje Pascal y TeX, en el caso de TeX, este lenguaje mantiene su vigencia en los contextos de ciencias e ingeniería.

Muchos de los elementos iniciales han cambiado: los grupos son heterogéneos, provienen de diferentes currículos, el poder de cómputo y sus recursos asociados (memoria, espacio de almacenamiento, número de dispositivos conectados) ha crecido exponencialmente con respecto de sus antecesores, así que se esperan programas más potentes, llamativos y por supuesto que se integren con cualquier dispositivo. En contraparte, los lenguajes de programación proporcionan una interface más cercana al programador. La Tabla No. 4 permite contrastar paradigmas de programación, lenguajes de programación, modelo instruccional empleado junto con el hito correspondiente, desde una perspectiva cronológica en el inicio de la enseñanza de la programación de computadoras.

Tabla 4: Características y elementos involucrados en la enseñanza de la programación.				
Periodo y Característica	**1970 -1980**	**1980-1990**	**1990-2000**	**2000-2014**
Paradigma	Procedimental	Procedimental - OOP	OOP	OOP - Otros
Lenguaje	Ensamblador - C	C - C++	C++ - Java	Java – Python -Orientados a la Web
Modelo instruccional	Tradicional	Tradicional	Constructivista	Constructivista
IDE	Manual	Herramientas corporativas de ensamble	Eclipse	Eclipse

Periodo y Característica	1970 -1980	1980-1990	1990-2000	2000-2014
Hitos	Orientación hacia la sintaxis del lenguaje.	Seymour Paper: lenguaje logo (centros de enseñanza media)	Masificación de la computación: oferta de Sistemas Operativos gráficos.	Scratch: Producto de Media Lab de MIT. Bluej: producto de Universidad Kent Kodu: producto de Microsoft

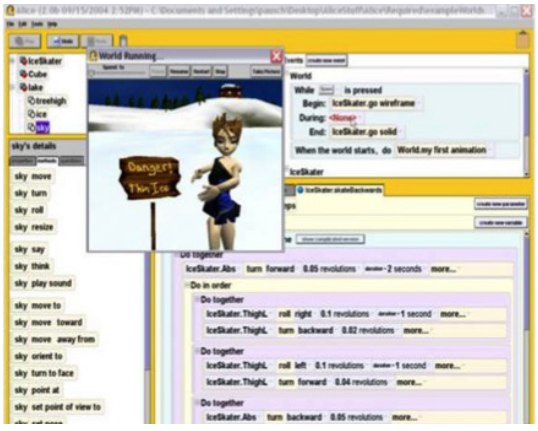
Fuente: El autor

V. PROPUESTAS

Considerando los factores mencionados a la luz de la escuela tradicional, los índices de pérdida académica y deserción en los cursos introductorios de programación de computadoras se elevan, de manera que la pregunta inicial se replantea así: *¿Cómo reducir los índices de deserción y mortalidad en los cursos de programación*? La respuesta ha impulsado el desarrollo de metodologías en la enseñanza de la programación, muy exitosas en particular, adicionando un factor no considerado previamente: la lúdica como uno de los ejes principales en el aprendizaje, de donde surgen propuestas como:

5.1 Desarrollo instruccional basado en vídeo juegos

En esta línea se encuentran resultados de proyectos muy interesantes como Alice, Greenfoot y Scratch, cuyo desarrollo ha partido del intento de reducción de las tasa de pérdida en los cursos de programación de computadoras. El trabajo pionero en esta línea fue el realizado por el pedagogo Seymour Papert con el tradicional lenguaje de programación Logo, expuesto en el clásico documento "Mindstorms: Children, Computers, and Powerful Ideas", empleado con gran éxito en centros de educación básica y media, enfocado especialmente hacia la enseñanza de las matemáticas.



quigua

Figura 4. Entorno Alice. Fuente: El autor

Figura 5. Entorno Greenfoot. Fuente: El autor

- Alice es un producto de la universidad Carnegie Mellon [13] (Figura 4).
- Greenfoot es el resultado del trabajo desde la universidad de Kent [15] (Figura 5).
- Scratch media Lab MIT [3] (Figura 6).

Estos entornos están orientados hacia la creación de historias o juegos interactivos, son de descarga libre y se encuentra fácilmente documentación, proyectos de ejemplo, asi como robustas redes de apoyo. Respecto a su curva de aprendizaje, Alice es un entorno profesional y requiere por lo tanto un poco más de dedicación, en comparación con Greenfoot y Scratch. Por su parte Greenfoot está orientado hacia el manejo desde Java mientras que Scratch está orientado hacia bloques y Alice admite C#, Java y C++.

Figura 6. Entorno Scratch. Fuente: El autor

5.2 Desarrollo instruccional orientado hacia el paradigma

De las propuestas incluidas en este enfoque, dos son el eje principal que fundamenta a los proyectos:

- Dr. java[4]
- Bluej[5].

Dr. Java es un proyecto de "Rice University" que propone un entorno enfocado en el código, buscando ser intuitivo, incluye varios elementos de apoyo interesantes como entorno de verificación y soporte a Junit, sin llegar a ser tan abrumador como un entorno comercial, el enfoque original orientado a las propuestas pedagógicas de soporte se encuentra principalmente en [1] y [19].

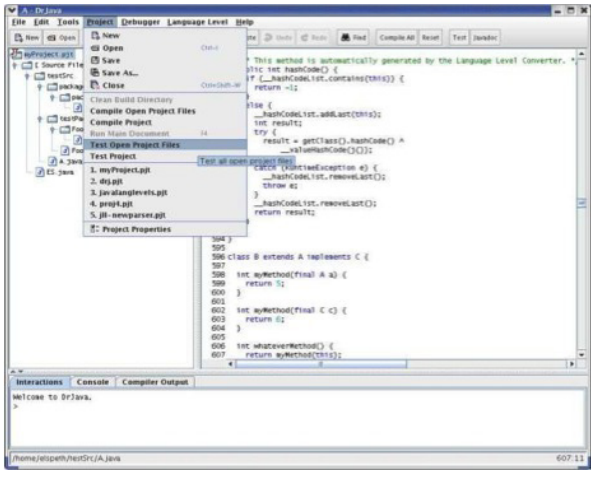


Figura 7. Entorno Dr. java. Fuente: El autor

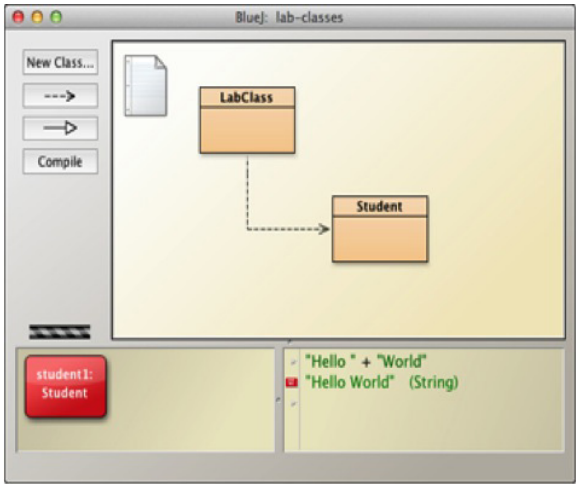


Figura 8. Entorno Bluej. Fuente: El autor

Bluej es un proyecto de "Kent University" diseñado específicamente como herramienta de apoyo en la comprensión de conceptos de orientación a objetos; el entorno tiene la cualidad de obviar la creación de un programa principal de tal manera que permite definir clases Java y pasar directamente a la creación de objetos e interactuar dinámicamente con ellos. Este esquema permite que el estudiante identifique las instancias y sus atributos al tiempo que verifica la efectividad de los métodos escritos.

Algunos trabajos previos que corresponden con este mismo enfoque, son:

- Raptor[6] (Figura 9): es un entorno de programación basado en diagramas de flujo.
- Iconic Programmer[7] es un entorno de programación propietario, orientado hacia el desarrollo de diagramas de flujo.
- DFD: es un entorno limitado, orientado hacia los diagramas de flujo
- PSeInt[8] (Figura 10): es un entorno libre, orientado hacia el desarrollo de procesos estructurados, tiene un novedoso conjunto de palabras clave en idioma español.

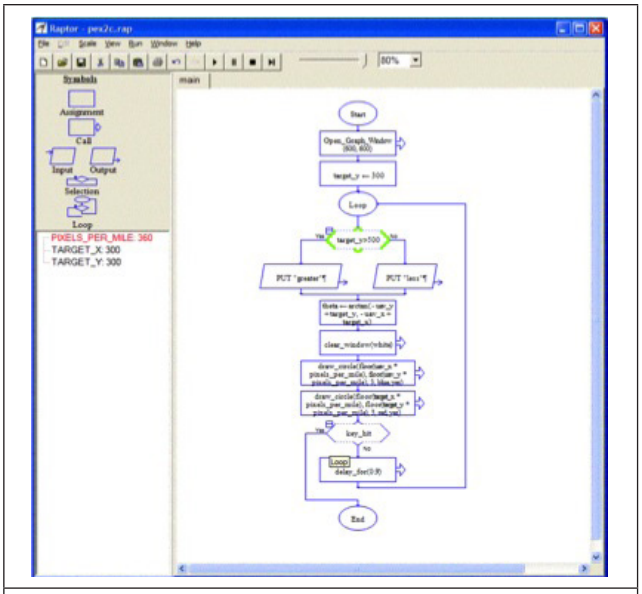


Figura 9. Entorno Raptor. Fuente: El autor

[4] http://www.drjava.org/ enlace oficial al proyecto
[5] http://www.bluej.org/ enlace oficial al proyecto
[6] http://raptor.martincarlisle.com/ página oficial del proyecto
[7] http://www.edutoolresearch.com/ página oficial del proyecto
[8] http://pseint.sourceforge.net/ página oficial del proyecto

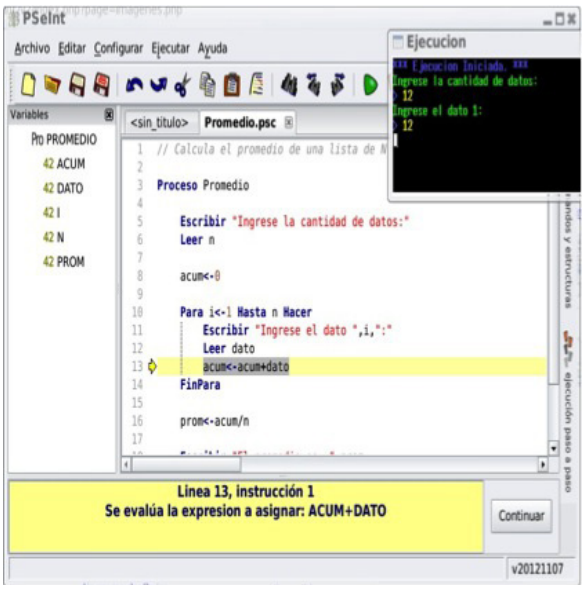


Figura 10. Entorno PseInt. Fuente: El autor

La mayoría de estos desarrollos han sido abandonados debido al surgimiento del paradigma orientado a objetos. Los proyectos mencionados aplican efectivamente en desarrollos exclusivamente estructurados.

5.3 Desarrollo instruccional enfocado en temáticas particulares

Estos proyectos están orientados al desarrollo de una característica o propuesta sugerida por el investigador, y en general integra herramientas existentes con metodologías:

- En [6] se propone el uso de XP (Extremme Programming), acompañado de las herramientas *CMaptools* y el entorno *Bluej* como un modelo de aprendizaje orientado hacia la simulación del proceso real en la industria, con tablas de asignaciones y calendarios para trabajo en parejas.
- El trabajo expuesto en [17], muestra un caso de éxito aplicado en un perfil de estudiantes específico (facultad de educación), siguiendo la práctica de programación por parejas, una práctica incluida en XP.
- Un enfoque estrictamente pedagógico lo presentan los investigadores: Eckerdal, Thuné y Berglund (Universidad Uppsala, Suecia) en [18], concluyendo para su contexto una aproximación estándar dada por:
  - Aproximación empleando la orientación a objetos.
  - Aproximación desde UML hacia el código.
  - Desde seudo-código hacia código.
  - Desde el algoritmo hacia su implementación.
- En [10] los autores presentan una propuesta llamada "**A**pproach **D**eployment **R**esult **I**mprovement" (ADRI), en donde se conjugan etapas de planificación e implementación de resultados y conclusiones enfatizando en la enseñanza de la programación, la necesidad de planificar y verificar los resultados, expuesta la metodología como un proceso iterativo cercana al trabajo industrial.

Recientemente se ha identificado un fenómeno interesante: la vuelta a la enseñanza de la programación enfatizando en un lenguaje de programación, tal es el caso del lenguaje Python, como lo muestra [8], en donde se prueba que este lenguaje se ha posicionado en los cursos de inicio en programación, en la mayoría de los centros de Estados Unidos.

VI. CONCLUSIONES

Una metodología adecuada para la enseñanza de los cursos iniciales de programación de computadoras, debe incorporar varios factores desde la pertinencia de la temática ofertada: objetivos de aprendizaje y pre-conceptos del estudiante, hasta el contexto de los programas académicos para los que se oferta el curso. En este ejercicio se identifican claramente dos tipos de perfiles:

- Cursos introductorios orientados hacía el especialista.
- Cursos introductorios considerando grupos interdisciplinares.

Cualquiera que sea el perfil de los grupos de influencia, se deben considerar factores que en general, no resultan evidentes para el orientador, de tal manera que cualquier propuesta involucra un serio trabajo interdisciplinario de pedagogos, especialistas, orientadores y evaluadores, que exige además, armonía con la filosofía institucional. Algunos de los trabajos más exitosos en estos campos involucran un compromiso e inmersión de los actores: docentes, estudiantes y directivos. Desde la academia, las alternativas visibles para enfrentar estos retos son:

- Dinamizar el currículo.
- Especializar la educación mediante la oferta de cursos en herramientas puntuales: por ejemplo

Matlab, R, Mathematica, etc., de acuerdo con el interés común.

- Definir sus metodologías propias: propuestas como [14] orientadas hacia el entorno de desarrollo y propuestas como [12], orientadas hacia el desarrollo de proyectos, las cuales mantienen estas constantes aunque desde orientaciones diferentes.

Por supuesto, este es un trabajo en conjunto: docentes y estudiantes, dado que los cambios en los currículos no se presentan, en general, con la celeridad esperada, se debe educar con la fundamentación y flexibilidad que le permita al estudiante explorar alternativas de especialización individual, este por ejemplo es uno de los ejes con los que trabajan las ofertas de los cursos masivos en linea o MOOCS, por sus siglas en Inglés.

En grupos heterogéneos la lúdica es una de las metodologías de enseñanza recomendada; en particular la enseñanza apoyada en entornos como Bluej, Alice y Greenfoot, han mostrado su efectividad en estos casos. El empleo de entornos de apoyo tales como Raptor, PSeInt y DFD que conservan una marcada orientación hacia la programación estructurada, se deben replantear debido a que corresponden con tecnologías revaluadas por la orientación a objetos.

La metodología a aplicar no necesariamente se debe tomar de alguno de los casos de éxito mencionados, es posible adaptar y definir una metodología propia de acuerdo con la trayectoria y experiencia de los involucrados, así por ejemplo, en UNIMINUTO se están empleando las competencias ACM-ICPC como eje integrador en la enseñanza y en general en todos los espacios se requiere una disposición y compromiso de los docentes y estudiantes en explorar y aceptar metodologías de enseñanza alternativas.

VII. REFERENCIAS BIBLIOGRÁFICAS

1. Allen Eric, Cartwright Robert and Stoler Brian, DrJava: *A lightweight pedagogic environment for Java*, SIGCSE 2002.

2. Barg, M., Fekete, A., Greening, T., Hollands, O., Kay, J., Kingston, J. H. & Crawford, K., *Problem based learning for foundation computer science courses*. Computer Science Education, 2000

3. Begosso, L.C., Da Silva, P.R., *Teaching computer programming: A practical review*, Frontiers in Education Conference, 2013

4. ACM/IEEE, *Computer Science Curricula 2013*, ACM, ISBN: 978-1-4503-2309-3, 2013

5. ACM/IEEE, *Computer Engineering* 2004, http://www.acm.org/education/education/curric_vols/

6. Diva~Sanjur Araúz, *Una metodología para el aprendizaje de la programación orientada a objetos basada en programación extrema (XP)*, Tesis de maestría, Universidad Tecnológica de Panamá, Panamá, 2010.

7. Eckerdal Anna, Thuné Michael, Berglund Anders, What Does It Take to Learn ‹*Programming Thinking*›*?* Proceedings of the First International Workshop on Computing Education Research, ICER ‹05, 2005, pages 142-135, ACM, New York, NY, USA.

8. Philip Guo, Python is Now the Most Popular Introductory Teaching Language at Top U.S. Universities, Blog@ACM, Communications of the ACM, July 7, 2014.

9. ACM/IEEE, *Integrated Software & Systems Engineering Curriculum* (iSSEc) Project , http://www.gswe2009.org/

10. Sohail Iqbal, Om Kumar Harsh, *A Self Review and External Review Model for Teaching and Assessing Novice Programmers*, International Journal of Information and Education Technology, Vol 3, 2013, Singapore.

11. ACM/IEEE, IS 2002, *Model Curriculum and Guidelines for Undergraduate Degree Programs in Information Systems*, http://www.acm.org/education/education/curric_vols/

12. J. Villalobos, M. Vela, *CUPI2-An Active Learning and Problem Based Learning Approach to Teaching Programming*, 8th ALE International Workshop, Bogotá - Colombia, Junio 2008.

13. Kelleher, Caitlin and Randy Pausch. *Lessons Learned from Designing a Programming System to Support Middle School Girls Creating Animated Stories*. Visual Languages and Human-Centric Computing. (Sept. 2006): 165-172.

14. Kölling, M. and Rosenberg, J., *Guidelines for Teaching Object Orientation with Java*, Proceedings of the 6th conference on Information

Technology in Computer Science Education (ITiCSE 2001), Canterbury, 2001.

15. Kölling, M.,The greenfoot programming environment, ACM Transactions on Computing Education, 2010.

16. Muhammad Ali, S. *Development of a Course Sequence for an Interdisciplinary Curriculum, Higher Education Studies*; Vol. 2, No. 3; ISSN 1925-4741 E-ISSN 1925-475X, Center of Science and Education, Canadian, 2012.

17. Irena Nančovska Šerbec, Branko Kaučič, Jože Rugelj, Pair Programming as a Modern Method of Teaching Computer Science, International Journal of Emerging Technologies in Learning (iJET), Vol 3, Iss SI2: MIPRO2008, Pp 45-49 , IAOE, 2008.

18. Pears, Arnold and Seidman, Stephen and Malmi, Lauri and Mannila, Linda and Adams, Elizabeth and Bennedsen, Jens and Devlin, Marie and Paterson, James, *A Survey of Literature on the Teaching of Introductory Programming*, Working Group Reports on ITiCSE on Innovation and Technology in Computer Science Education, 2007, pages 204-223, ACM, New York, NY, USA.

19. Brian Stoler, *A Framework for Building Pedagogic Java Programming Environments*, Masters Thesis, Rice University, 2002