INVESTIGACIÓN & DESARROLLO, No. **4**, Vol. **1**: 85 – 94 (2004)
ISSN 2518-4431

ADA EN LA ENSEÑANZA DE LA PROGRAMACIÓN

Luis Marcél Barrero Mendizábal
Universidad Privada Boliviana
mbarrero@upb.edu
(Recibido el 17 de mayo 2004, aceptado para publicación el 19 de septiembre 2004)

RESUMEN

El objetivo del presente artículo es proponer una metodología para la enseñanza de la programación, basada en los conceptos y principios de la ingeniería del software proponiendo además el lenguaje ADA como el más apropiado para este propósito.
Para respaldar la segunda afirmación, en el artículo de procederá a comparar técnica y objetivamente ADA con otros lenguajes de programación que actualmente son utilizados en la enseñanza.

Palabras Clave: ADA, Ingeniería de Software, Lenguajes de Programación.

1. INTRODUCCIÓN

Lo más importante para una persona que desea iniciarse en el arte de la programación, es el primer contacto. En este sentido se debe seleccionar correctamente la metodología y herramientas a ser empleadas.

Un buen programa para la enseñanza de la programación debe considerar los siguientes puntos: Definir los objetivos y establecer una metodología apropiada para alcanzar los objetivos planteados.

Para definir los objetivos, se deben tener en cuenta los siguientes aspectos:

- Contexto de desarrollo del programa: Es necesario conocer la motivación que lleva a la persona a buscar aprender a programar. Si se trata de un contexto académico, se debe considerar además el objetivo global que se tiene definido en el programa curricular para ser coherentes con el mismo.
- Formación previa de los estudiantes: Es decir comprender el nivel de conocimientos previos que tienen los estudiantes al acceder al programa de formación. Conocimientos matemáticos, lógicos; capacidad de análisis y abstracción; conocimientos de programación, etc.
- Esta información es importante, pues de ella dependerá fijar el punto de partida para asegurarse que todos alcancen el objetivo planteado.
- Definir el alcance del programa: Es decir definir el nivel de desempeño que se desea que los estudiantes alcancen al finalizar el programa.
- Fundamentar el programa en conceptos de ingeniería de software: En todo momento se debe buscar el formalismo aportado por las metodologías de ingeniería de software para que los estudiantes adquieran una disciplina de trabajo que les servirá en su desempeño profesional.

En lo que concierne a la metodología, en la actualidad algunas instituciones de formación han optado por un acercamiento práctico, es decir, el programa se lleva a cabo directamente en los laboratorios, donde los estudiantes aprenden a desarrollar programas o escribir líneas de código desde el primer día.

Cuando se aplica esta metodología, se enfrentan los siguientes problemas:

- El estudiante no está obligado a aplicar una metodología formal en el desarrollo de una aplicación.
- El estudiante tiende a trabajar siguiendo el principio de prueba/error, donde el resultado correcto, cuando es encontrado, es más por aproximación que por la aplicación de una metodología formal.
- El estudiante confunde entre él *Aprender a Programar* y *Aprender un Lenguaje de Programación*. De esta manera, confunde los conceptos de *Algoritmo* y *Programa en Código Fuente*.
- Es estudiante llega a pensar que con el trabajo realizado en laboratorio, donde incluso el o los problemas planteados fueron resueltos, es suficiente. No todos adquieren una costumbre de práctica que es con lo que en realidad el estudiante ganará experiencia para en el futuro hasta convertirse en experto.

DOI: 10.23881/idupbo.004.1-11i

2. DEFINICIÓN DE LA ESTRATEGIA

Habiendo analizado los problemas de los programas vigentes, a continuación se presentarán los objetivos y la metodología propuesta para crear un programa de Introducción a la programación.

- **Objetivos**

El objetivo del programa es introducir al estudiante en el mundo de la Computación. El programa debe tener por lo menos las siguientes partes:

- Introducción a la Tecnología: El estudiante debe conocer los componentes de una computadora y su funcionamiento. También debe conocer cómo la información es representada al interior de una computadora, concepto bastante importante para comprender la importancia de los Tipos de Datos.
- Introducción a la Ingeniería de Software: El estudiante debe comprender la importancia de adquirir una disciplina en el desarrollo de aplicaciones informáticas. Esta metodología le permitirá también poder analizar, plantear y resolver problemas desde un punto de vista abstracto.
- Introducción a la Programación: El estudiante debe poder formalizar las soluciones encontradas a los problemas planteados para culminar con la implementación de la solución en un lenguaje de programación.

El estudiante debe dominar cada uno de los componentes de un programa y los elementos esenciales de un lenguaje de programación, para terminar con la implementación de la solución.

El programa debe comenzar con los fundamentos más básicos, asumiendo que existen estudiantes que cuentan con poca o ninguna experiencia previa en Computación o Programación.

Al finalizar el programa, los estudiantes deben haber adquirido la capacidad de analizar un problema, plantear un algoritmo e implementar la solución en un lenguaje de programación. Todo trabajo desarrollado debe venir acompañado de un Manual para el Usuario y un Manual Técnico.

- **Metodología**

El primer cambio que debe aplicarse a la Metodología tradicional descrita anteriormente, es el ambiente donde se lleve a cabo el programa. Debe pasarse del laboratorio al aula. Desarrollando el trabajo en aula, se tiene la posibilidad de hacer mayor énfasis en la aplicación de una metodología que en el desarrollo de líneas de código, poniendo en manifiesto la importancia de la Ingeniería del Software.

No se debe permitir el acceso a los estudiantes a los laboratorios antes de llegar a la fase de codificación o implementación, para evitar que adquieran malos hábitos que vayan en contra de la metodología impartida.

Una vez que la Metodología a terminado de ser impartida, los estudiantes deben tener una cantidad suficiente de problemas planteados para ser analizados y resueltos con la guía del docente o instructor del programa. El trabajo de laboratorio debe estar propuesto de tal manera que los estudiantes utilicen las sesiones de laboratorio para poder absolver dudas posibles o recurrir a la guía del docente o instructor en la solución del problema y en ningún caso utilizarlas como único espacio/tiempo para buscar resolver los problemas propuestos.

Todos los trabajos desarrollados como prácticas, deben venir acompañados de una documentación completa.

Finalmente, el docente o instructor del programa debe evitar establecer un compromiso absoluto con un lenguaje de programación dejando en claro la diferencia existente entre *Saber Programar* y *Conocer un Lenguaje de Programación*. Para evitar el compromiso, el docente o instructor debe tener la capacidad de mostrar al estudiante que un algoritmo bien planteado puede ser implementado en cualquier lenguaje de programación.

De todas maneras, es importante poder seleccionar un lenguaje que permita asimilar los principios de la programación y que ayude en la adquisición de buenos hábitos del programador, en ese sentido, se procederá a analizar las diferentes alternativas posibles, considerando incluso diferentes paradigmas de programación existentes.

- **Paradigmas de Programación**

Los lenguajes de programación se organizan según las características que ofrecen, en *Paradigmas de Programación*, siendo las más comunes la Programación Estructurada, la Programación Orientada a Objetos, la Programación Lógica y la Programación Funcional.

Para determinar cuál es el mejor paradigma para el programa, existen muchos debates a nivel internacional. Cada paradigma cuenta con un importante grupo de partidarios y detractores. Muchas instituciones se inclinan por el uso del Paradigma Funcional. Como primer lenguaje, suelen utilizar *Scheme* [6] para enseñar a programar. El principal argumento que presentan para justificar la elección de este paradigma, es que se concentra más en la parte algorítmica del programa que en la parte sintáctica. Los lenguajes funcionales son simples, de pocas palabras reservadas y con poco o ningún control de tipos de datos. La principal desventaja de utilizar este paradigma, es que al basarse en una manera de pensar diferente a los demás paradigmas de programación, la transición a otro paradigma no siempre es fácil o directa.

Otro paradigma existente es la Programación Orientada a Objetos (POO), que es el resultado de una evolución natural de la Programación Estructurada. Esta evolución es fácil de comprender, pues elementos de la Programación Estructurada se hallan a la base de la Programación Orientada a Objetos.

Varios programas han optado por utilizar lenguajes basados en POO, la diferencia fundamental existen entre los programas que utilizan este último y el propuesto en este artículo, son los conocimientos con los que los estudiantes acceden al programa. En nuestro medio no todos los estudiantes llegan con conocimientos básicos de computación o programación.

Comenzar con un lenguaje de Programación Orientado Objeto, requiere también una capacidad de abstracción bastante profunda, de modo a poder comprender conceptos como Clases, Objetos, Mensajes y Herencia. Es posible realizar Programación Estructurada utilizando un lenguaje Orientado a Objetos, pero esa situación lleva al estudiante a adquirir un hábito de programación que no es el apropiado para el entorno de desarrollo.

En caso de desarrollar un programa **Estructurado**, en un entorno **Orientado Objetos**, el estudiante se verá enfrentado a los siguientes problemas (para ilustrar la comparación se considerará Java como lenguaje orientado objetos):

- No se presentan los conceptos propios a la Programación Orientada Objetos. El estudiante no tiene la capacidad de diferenciar entre los conceptos de Clase y Objeto.
- El docente debe presentar un entorno de desarrollo, donde (dependiendo del lenguaje seleccionado) deba omitir la explicación de la obligatoriedad del uso de ciertas palabras reservadas como `class`, `public`, etc.

En la Figura 1 se puede ver un ejemplo de código fuente de un programa escrito en Java y que permite desplegar en la pantalla un mensaje.

```
import java.io.*;

public class HelloWorld {
  public static void main (String [] args)
  {
      System.out.println ("Hello World!\n");
      return;
  }
}
```

Figura 1 - *Hello World!* en Java.

- Si seguimos con el ejemplo de Java, en caso de estar programando una aplicación de consola, las funciones de entrada de datos no se aplican intuitivamente.

Antes de leer los datos de la fuente (que puede ser simplemente el teclado), es necesario crear instancias de un dispositivo de entrada y recién leer los datos. En la Figura 2, vemos un ejemplo donde el programa pide al usuario que pueda introducir su nombre, para saludarlo en forma personalizada.

```
import java.io.*;

public class SaludoPersonalizado {
  public static void main(String[] args) {
    String Nombre = "";
    System.out.print("¿Cuál es su nombre? ");
    BufferedReader stdin = new BufferedReader(
                               new InputStreamReader(System.in));
    try {
      Nombre = stdin.readLine();
    }
    catch(IOException e) {
      System.out.println("Error de IO.");
    }
    System.out.println("Buen día " + Nombre + ".");
    return;
  }
}
```

Figura 2 - *Saludo personalizado* en Java.

En el programa, que podría ser de los primeros que realice el estudiante, éste se ve confrontado a una serie de palabras y conceptos nuevos. Antes de comprender el funcionamiento mismo del programa, el estudiante deberá familiarizarse con conceptos como: Instanciación de objetos (`new`), Acceso a métodos (`stdin.readLine()`) y el manejo de Excepciones (`try … catch`).
Finalmente, otro problema (que por cierto no es exclusivo de Java), es el control de los desbordes de capacidad a tiempo de realizar las operaciones. En la Figura 3, tenemos un código que aparentemente funciona correctamente, pero que despliega un resultado incorrecto. Se trata del incremento de una variable inicializada en `127`, el resultado esperado es `128`, pero el desplegado por el programa es **`-128`**. El problema reside en el rango de números que permite representar el tipo de datos `byte`: (`-128..127`), entonces por el desborde de capacidad, el resultado desplegado en pantalla es **`-128`**, sin señalar ningún tipo de error.

```
import java.io.*;

public class Overflow {
  public static void main(String[] args) {
    byte b = 127;
    System.out.println("b = " + b);
    b++;
    System.out.println("b = " + b);
    return;
  }
}
```

Figura 3 - *Overflow* no detectado en Java.

Esta situación puede confundir al estudiante que no tenga conocimientos sobre la representación interna de la información y la manera cómo se realizan las operaciones aritméticas en la computadora.

En base a los argumentos planteados, se llega a la primera conclusión parcial: Para llevar adelante un programa de Introducción a la Programación, se debe recurrir al paradigma de programación estructurada. En la formación del estudiante se deben considerar otros programas complementarios que los prepare en los paradigmas Orientado a Objetos, Lógico y Funcional.

- **Lenguaje de Programación**

Teniendo la figura clara sobre el Paradigma de Programación a ser utilizado, el siguiente paso es seleccionar el lenguaje de programación más adecuado para llevar adelante el programa.

La lista de lenguajes que pueden ser utilizados es extensa, sin embargo en esta comparación se considerarán solamente algunos: C, Pascal y ADA.

Uno de los principios de la Programación Estructurada es *"dividir para reinar"*. Es decir, el Programador debe ser capaz de identificar los diferentes componentes de los que estará hecha la solución y ver la posibilidad de

implementarlos como módulos o unidades independientes. Esta división se ve reflejada en el uso correcto de las Funciones, Procedimientos y Módulos.

Las Funciones, son un conjunto de expresiones, que al finalizar, devuelven un resultado que debe ser recuperado por la entidad que la invocó. Los Procedimientos, son también un conjunto de expresiones, pero que al finalizar, en principio, no devuelven ningún tipo de resultado. Finalmente un Módulo, es un conjunto de funciones y/o procedimientos que tienen un objetivo común.

El comportamiento de estas funciones o procedimientos, puede estar controlado por uno o varios parámetros, que adaptarán su funcionalidad al contexto donde son invocadas. En términos de programación se habla de pasaje de parámetros.

Existen dos modos de *pasar* los parámetros: el pasaje de parámetros por valor y por referencia. En el primer caso, la rutina recibe en una variable local (el parámetro formal), una *copia* del valor del parámetro actual. En el segundo caso, lo que se pasa es una referencia hacia el parámetro actual, permitiendo su modificación por la rutina.

A tiempo de enseñar el Paradigma de Programación Estructurada, se debe hacer pasar (entre otros), estos dos conceptos esenciales.

En el caso del Lenguaje C, se tiene un par de problemas fundamentales con los dos conceptos mencionados:

En el lenguaje C, todas son *Funciones*, no existe el concepto de *Procedimiento*. En realidad, si se desea definir un procedimiento, se debe crear una función que no devuelva ningún resultado (`void`), por lo tanto el concepto de Procedimiento no sólo no es natural, sino hasta llega a parecer forzado, Figura 4.

```
#include <stdio.h>

void
procedimiento()
{
  /*
   * return sin valor devuelto
   */
  return;
}

int
main ()
{
  procedimiento();
  exit (0);
}
```

Figura 4 - *Procedimientos* en C.

Por otro lado, el estudiante puede pensar que el omitir el tipo del resultado en una función, implica automáticamente `void`, lo que es absolutamente falso, pues según la definición estándar, en caso de omitir el tipo del resultado en la función, se asume que se trata del tipo entero (`int`) [4]. Esta confusión puede sentirse reforzada por el hecho que en el lenguaje C, cuando se invoca a una función, no se está obligado a recuperar el resultado en una variable, se la puede omitir (situación bastante común, pues la mayor parte de las funciones devuelve un resultado que no siempre es tomado en cuenta. ¿Alguien recuerda el significado del resultado de la invocación a la función `printf`?).

En el caso del pasaje de parámetros, en C todos los parámetros son pasados por valor, es decir la función sólo recibe en una variable local, una copia del valor de la variable original. Esto obliga a recurrir a la noción de puntero (con todas sus implicaciones), cada vez que en el programa se necesite un pasaje de parámetros por referencia (ver ejemplo en la Figura 5). Nuevamente se tiene a una solución que no parece natural para el problema.

Finalmente, el lenguaje C realiza muchos supuestos. Cuando se ejecutan ciertas operaciones matemáticas, realiza conversiones implícitas de tipos de datos, impidiendo que el programador tenga el control sobre la manera en la que desea realizar las operaciones.

```
#include <stdio.h>
```

```
void
ppref(int *pf)
{
  (*pf)++;

  return;
}

int
main ()
{
  int va = 0;

  printf ("Valor antes del Procedimiento: %d\n", va);
  ppref(&va);
  printf ("Valor después del Procedimiento: %d\n", va);

  exit (0);
}
```

Figura 5: *Pasaje de Parámetros por Referencia* en C.

Con estos otros argumentos, se llega a una segunda conclusión parcial:

El lenguaje de programación C, no es de los más aptos para enseñar a programar, pues tiene muchas deficiencias en la implementación de conceptos fundamentales. Sin embargo, la formación del programador debe complementarse en algún momento con el estudio de este lenguaje, considerando la gran difusión y aceptación con la que cuenta.

- **Análisis de otro lenguaje: Pascal.**

Pascal fue creado por *Niklaus Wirth* con dos objetivos principales [8]:

- Para tener a disposición un lenguaje apropiado para enseñar programación con una disciplina sistemática.
- Para tener un lenguaje que pueda ser confiable y eficiente en las computadoras disponibles en la época (1970).

Pascal fue utilizado durante mucho tiempo para enseñar programación en muchas universidades alrededor del mundo. Como fue diseñado por Wirth con fines académicos, cumple con las dos características mencionadas anteriormente: Diferencia entre Procedimientos y Funciones y tiene implementados los dos tipos de pasajes de parámetros.

En 1998, en la conferencia internacional SIGAda patrocinada por la ACM, dos oficiales norteamericanos presentaron una comparación entre el uso de Pascal y Ada como primer lenguaje de Programación [7]. Entre los problemas identificados, se pueden rescatar los siguientes:

- En Pascal, las funciones de Entrada/Salida (`Read/Write`), aparecen al estudiante como provistos directamente por el compilador. Es decir no se comprende que se tratan de funciones y procedimientos que son parte de una biblioteca estándar.

En la Figura 6, se tiene un código escrito en Pascal para desplegar un mensaje en la pantalla. El procedimiento `WriteLn` es invocado directamente sin necesidad de incluir ninguna unidad o biblioteca. Como se indicó anteriormente, esta situación puede llevar al estudiante a confundir entre los componentes que son propios del lenguaje y aquellos que son propios a la arquitectura donde se ejecuta el programa.

```
program HelloWorld;

begin

WriteLn ('Hello World!');

end.
```

Figura 6 - *Hello World!* en Pascal.

- No todos los entornos de desarrollo son estándar: Si bien se ha definido un estándar para el Lenguaje Pascal, los diferentes editores han añadido sus propias unidades, dando al estudiante la impresión que las funciones definidas en esas unidades, son parte del lenguaje, cuando en realidad son añadiduras que seguramente no hallarán en otro compilador. Por ejemplo, la unidad `Crt` implementada inicialmente por *Borland*, ofrece el Procedimiento `ClrScr` (que permite limpiar la pantalla), al no ser parte de Pascal, no está disponible en todos los compiladores y los estudiantes que aprendieron a programar en ese entorno, vanamente la buscarán en otras implementacione.

Estas razones nos permiten llegar a una tercera conclusión parcial:

Pascal puede ser utilizado para enseñar a programar, pero tiene también falencias que puedan confundir al estudiante.

Finalmente, queda por considerar ADA como lenguaje a ser utilizado en el programa. En el Anexo, se presenta una breve historia del Lenguaje de Programación, de esta historia los puntos más importantes son:

- Es un lenguaje hecho a pedido.
- Los requerimientos fueron establecidos por el exigente Departamento de Defensa de los Estados Unidos de Norteamérica.
- Es un lenguaje estandarizado desde su concepción. Con lo que el lenguaje esencialmente es el mismo en todas las plataformas, con excepción de las unidades o módulos dependientes de la arquitectura.

Si bien ADA en ningún momento fue concebido, como Pascal, como un lenguaje a ser utilizado en la enseñanza de la Programación, tiene mucho en común con éste. Se podría decir que ADA es un descendiente de Pascal (no sólo en su sintaxis, sino sobre todo en sus raíces europeas).

ADA toma entonces los fundamentos de Programación Estructurada planteados por Pascal y corrige las falencias hasta entonces detectadas:

- Las funciones de Entrada/Salida (`Get`/`Put`), son parte de una biblioteca estándar, que contiene las funciones y procedimientos que serán utilizados en el programa. Dependiendo del tipo de datos a ser utilizados, el programador debe incluir en su programa las bibliotecas apropiadas o crear las instancias correspondientes.
  Académicamente hablando, esta situación permite al estudiante comprender que los diferentes tipos de datos, deben ser tratados de manera diferente y que en caso de necesidad, podrá crear sus propias funciones de Entrada/Salida (sobrecarga de operadores).

- La conversión de tipo debe ser explícita. Al ser ADA un lenguaje fuertemente *tipado*, las operaciones sólo pueden efectuarse si han sido definidas para el contexto donde se las desea utilizar. Por ejemplo, no es posible sumar un valor entero con un valor real, el operador suma no ha sido definido en ese contexto. El programador tiene dos alternativas: Según el resultado que desee obtener, debe convertir explícitamente el valor entero en real (o viceversa) o puede sobrecargar el operador suma, para definir este nuevo contexto de aplicación (este último ejemplo es bastante importante sí se trabaja con Tipos Derivados). En los dos casos, ni el compilador, ni el lenguaje, toman decisiones que pueden ser importantes en el desarrollo del programa, es el programador quien en todo momento controla y domina la situación.
  En el contexto académico, esto transfiere al estudiante una disciplina de programación y una responsabilidad con el código que está siendo desarrollado.

- Todos los errores son señalados de manera estándar, haciendo referencia a los párrafos del Manual de Referencia del Lenguaje donde se puede hallar información adicional del problema detectado [3].
  Para el tratamiento de los errores, utiliza el manejo de Excepciones, concepto bastante extendido en el Paradigma de Programación Orientado a Objetos. Tiene una serie de Excepciones predefinidas, que ayudan al programador a detectar el error en tiempo de ejecución y de tratarlo en consecuencia, sin que esto signifique que el programa deje de ejecutarse [10].

- Al ser un lenguaje Estándar, todos los entornos de desarrollo son idénticos, haciendo que los programas desarrollados en ADA, sean absolutamente *portables*.
  En el caso de recurrir a funciones o procedimientos específicos (el caso por ejemplo de una función o procedimiento que limpie la pantalla), el programador debe conocer el entorno en el que se encuentra desarrollando e incluir en su programa las bibliotecas de paquetes correspondientes a su plataforma de desarrollo. Esta situación, más que ser una desventaja, es una ventaja que permite mostrar al estudiante la diferencia que existe al trabajar sobre diferentes arquitecturas tecnológicas.

- En el caso de los pasajes de parámetros, ADA diferencia dos tipos de pasaje por referencia: Si la variable sólo puede ser modificada (`out`), si la variable además de poder ser modificada, puede ser portadora de información (`in out`). Para completar la trilogía, el pasaje por valor, es sólo para variables que pueden ser consultadas (`in`).
- La posibilidad de pasar los parámetros con nombre, permite al estudiante realizar una asociación entre el parámetro formal (declarado al interior de la función o procedimiento) y el parámetro actual (con el que se invoca a la función o procedimiento). Esta situación permite además que se puedan definir valores por defecto para algunos parámetros formales, en caso de no especificar parámetros actuales (parámetros opcionales).
- ADA95 es también un lenguaje orientado a objetos, situación que nos permite presentar al estudiante, naturalmente, conceptos propios a este paradigma de programación, como el tratamiento de excepciones para el manejo de errores.

Estos antecedentes nos permiten llegar a una nueva conclusión parcial:

ADA es un lenguaje estructurado, implementa todos los conceptos importantes de este paradigma de programación. La rigidez que tiene permite al estudiante no sólo asimilar los conceptos de la programación estructurada, sino que le permite adquirir una disciplina de trabajo.

- **La Programación en la Universidad Privada Boliviana - UPB**

La metodología descrita en este documento está siendo aplicada desde el año 2002 en la Universidad Privada Boliviana en las materias de Introducción a la Programación (Programación I) y Estructura de Datos (Programación II). El objetivo principal de estas dos materias, es *Enseñar a Programar* que no debe ser confundido con *Enseñar a desarrollar en **un** lenguaje de Programación*.

Tradicionalmente, los cursos introductorios a la programación, se los llevaba a cabo en los laboratorios, argumentando que se trata de una materia completamente práctica. En el nuevo enfoque que se da a la materia, de aprender a aplicar una metodología de desarrollo, se llevan los cursos al aula, donde el estudiante debe primero desarrollar un algoritmo para luego, recién, pasar a las computadoras para su implementación. Se ha constatado que los estudiantes que "aprenden" a programar directamente en laboratorio o en máquina, no desarrollan un espíritu de análisis y aprenden a programar siguiendo una empírica metodología de *prueba y error*, situación que les impide luego aplicar metodologías de Ingeniería de Software a tiempo de desarrollar Aplicaciones más complejas.

Luego de dos años de trabajar con este método, se han visto progresos considerables en los estudiantes y en la calidad de los trabajos realizados como proyectos de fin de materia. Efectivamente, al finalizar cada uno de los cursos, los estudiantes individualmente o en grupos de dos, desarrollan un proyecto donde demuestran todas las habilidades adquiridas.

Para el curso de Introducción a la Programación, se propone al estudiante desarrollar un juego. Se proponen juegos puesto que al finalizar el primer semestre aún no tienen demasiados conocimientos específicos de su carrera para desarrollar problemas relacionados con su formación profesional, entonces se recurre a un entorno que aprovecha del principio lúdico para interesar al estudiante. Ya en el Proyecto de la materia de Estructura de Datos, los trabajos tienden a tener otra visión mucho más aplicada y concreta.

Entre los mejores programas presentados por los estudiantes se puede citar a los siguientes (esto, sin desmerecer el trabajo del resto de los estudiantes, que de todas maneras demostraron ingenio y creatividad en sus soluciones):
ADANOID, es la implementación desarrollada por los estudiantes Marco Camacho y Daniel Flores, del popular juego Arkanoid.

TANK, es otro juego desarrollado por los hermanos Raúl y Juan Pablo Romero. La idea es dar las instrucciones a un tanque de guerra para que alcance a un blanco.

También los estudiantes Gabriel Belmonte y Walter Guerrero, diseñaron como proyecto final una versión del conocido juego del Buscaminas.

En el curso de Programación II, los estudiantes Huari Delgadillo Barriga y Willie Pozo, desarrollaron una aplicación que permite dibujar figuras geométricas, al más puro estilo de lo que fue el Paintbrush de Microsoft.

También los estudiantes Silvia Soria y Roger Villegas, desarrollaron un sistema que permite Controlar la Producción en una Panadería.

Todos los trabajos vienen acompañados de un Manual Técnico y un Manual de Usuario, dentro de los requerimientos de documentación de la metodología presentada.

Si se toma en cuenta que muchos de estos estudiantes no tenían conocimientos de programación al comenzar el curso, y que tan sólo unas semanas después logran desarrollar ese tipo de aplicaciones, se puede considerar como exitosa la implementación de la metodología mencionada y contar con la seguridad, que en los años que vienen, el aprendizaje correcto de la programación de los estudiantes en la UPB está asegurado.

3. CONCLUSIONES

Partiendo de la importancia que tiene la aplicación de metodologías de ingeniería de software desde los primeros pasos en programación, se ha demostrado que el lenguaje más apropiado para enseñar a programar en nuestro medio es ADA. Se han presentado las ventajas que se tiene al trabajar en un entorno de desarrollo estricto como es el planteado por ADA. Los estudiantes adquieren una disciplina de trabajo que les será de mucha utilidad en el futuro, cuando se vean envueltos en desarrollos mucho más complejos.

Trabajando con un lenguaje de programación que no es de los más difundidos o populares, los estudiantes comprenden la importancia de ampliar sus horizontes y no cerrarse ante una sola alternativa. Esta situación les permitirá en el futuro adaptarse a los cambios tecnológicos que vayan surgiendo.

Esta conclusión en ningún caso debe considerarse como absoluta. Permanentemente deben hacerse nuevos análisis de la realidad en la que nos desenvolvemos para adaptarnos a los cambios de mentalidad, cambios tecnológicos e incluso cambios en las necesidades. Es muy posible que dentro de poco tiempo, así como hoy se propone ADA, se proponga otros lenguajes con otros objetivos y otras metodologías.

La informática es dinámica, la tecnología es dinámica y ese dinamismo debe reflejarse en los programas que sirvan para introducir a la programación a los estudiantes y a todos los interesados en general.

4. REFERENCIAS

[1] AdaIC. *The History of ADA.* Internet: http://archive.adaic.com/docs/flyers/history.html [1998].
[2] Department of Defense, *Requirements for higher order Computer Programming Languages: Steelman*. http://www.adahome.com/History/Steelman/intro.htm [1978].
[3] ISO-IEC, *ISO/IEC 8652: ADA95*, 1995.
[4] ISO-IEC, *ISO/IEC 9899: C*, 1999.
[5] Pascal Leroy. *An invitation to ADA 2005*. Ada Letters, September 2003.
[6] MIT. *Scheme.* Internet: http://www.swiss.ai.mit.edu/projects/scheme/
[7] Major Jeanne Murtagh (USAF), Lt. Col. John Hamilton (US Army), *A Comparison of Ada and Pascal in an Introductory Computer Science Course*, 1998.
[8] Oberon Microsystems, Inc., *A Brief History of Pascal*, 1997.
[9] Ryan Stansifer. *History of the Ada Programming Language.* Internet: http://www.cs.fit.edu/~ryan/ada/ada-hist.html
[10] Alfred Strohmeier, *A side-by-Side Comparison of Exception Handling in Ada and Java*, Ada Letters September 2001.

ANEXO: BREVE HISTORIA DE ADA

ADA es un lenguaje creado a pedido, a partir de un documento de especificaciones emanado del Departamento de Defensa de los Estados Unidos de Norteamérica. El documento en cuestión fue elaborado luego de constatar la diversidad de lenguajes de programación que se utilizaban, en un entorno caótico, muchas veces incompatible y difícil de mantener [1].

En enero de 1975, se creó el HOLWG (*Higher Order Language Working Group*), para preparar un cuaderno de requerimientos para poder seleccionar el lenguaje que sería utilizado en forma oficial en el DoD.

En abril - 1975, aparece la primera versión del documento de requerimientos, bajo el nombre de *Strawman* [1]. Luego de una primera revisión al documento, en agosto del mismo año, salió la segunda versión del documento, bajo el nombre de *Woodenman* y poco después la tercera versión con el nombre de *Tinman*, en enero de 1976.

El HOLWG, verificó con 23 lenguajes de programación de la época, si alguno cumplía con los requerimientos de *Tinman*,

En enero de 1977, ninguno de los lenguajes verificados satisfacía completamente; sin embargo, Pascal, Algol y PL/1 eran considerados como un buen inicio [9]. Con estos resultados, publicaron una nueva versión del documento, consideradas como las especificaciones del lenguaje ideal. Una vez publicado el documento, se recibieron 17 propuestas de diferentes partes del mundo. Se seleccionaron cuatro propuestas, identificándolas cada una con un color diferente:

- Verde: La Propuesta de CII Honeywell Bull. Grupo dirigido por Jean Ichbiah.
- Rojo: La Propuesta de Intermetrics. Grupo dirigido por Benjamin M. Brosgol.
- Azul: La Propuesta de SofTech. Grupo dirigido por John Goodenough.
- Amarillo: La Propuesta de SRI International. Grupo dirigido por Jay Spitzen.

De las cuatro, las dos primeras (la propuesta Verde y la propuesta Roja), fueron seleccionadas finalistas en abril de 1978. Dos meses después, en junio, se publicó el documento con las especificaciones definitivas. El documento final, recibe el nombre de *Steelman* [2].

En mayo de 1979, la propuesta Verde (del Grupo Francés CII Honeywell Bull), fue seleccionada ganadora. ADA había nacido. El nombre se debe a *Augusta Ada Byron, Condesa de Lovelace* (hija de célebre poeta Lord Byron), quien fue asistente del científico *Charles Babbage*. Por los trabajos realizados para la Máquina de Diferencias de Babbage, ADA es considerada la Primera Programadora de la Historia.

Entre abril y octubre de 1981, se puso a disposición de la comunidad el estándar ANSI propuesto por el DoD. Fruto de esta propuesta, se procedió con la estandarización del lenguaje, resultando en el estándar militar 1815A (enero de 1983), ANSI un mes después e ISO-8652 en 1987. En febrero de 1995, fue aceptada una modificación al estándar, es el lenguaje tal y como se lo conoce en este momento: ADA95. Este mes acaba de salir la convocatoria para ADA 2005, una futura y nueva revisión del lenguaje [5].

[1] Los nombres para los documentos, fueron tomados del libro *El Mago de Oz*, de L. Frank Baum.