Loro - Motivación didáctica*

Carlos A. Rueda
http://loro.sf.net

Índice

 Agradecimientos a Luz Adriana Osorio de la Universidad de los Andes (Colombia) por sus valiosos comentarios.

1. Introducción

Este trabajo surge de la necesidad de disponer de una herramienta de programación diseñada concretamente para apoyar la enseñanza de la programación a nivel introductorio en carreras de computación y afines. Puesto que se trata de una extensa área, que involucra muchos más elementos además de los materiales de apoyo (papel del tutor o educador, pedagogía utilizada, modelo didáctico seguido, interacción grupal, etc.), y en la que abundan diversos planteamientos, enfoques e incluso opiniones encontradas, tal herramienta deberá caracterizarse indispensablemente por ofrecer una arquitectura abierta que posibilite su extensibilidad y adaptación a distintas circunstancias y requerimientos. Este proyecto se enriquece desde luego de la amalgama de propuestas y alternativas existentes pero se busca, al mismo tiempo, alcanzar un cierto grado de autonomía que responda a un compromiso con nuestro propio contexto educacional (Baeza, 2000).

2. El problema

Los lenguajes de tipo industrial comúnmente utilizados para propósitos de enseñanza de la programación (Java, C++, C, Delphi, Basic, etc.), importantes sin duda por su potencia y también altamente atractivos por su mercado laboral, no siempre se constituyen en los recursos adecuados cuando se trata del nivel de principiante (Findler, 2001; Kölling, 1999; Motil y Epstein, 1998). Una situación típica, e incómoda en algunos casos, se presenta cuando el profesor se ve prematuramente comprometido a explicar, o bien forzado a eludir, ciertos aspectos de la programación por imposiciones del lenguaje o del entorno de desarrollo, con el riesgo de confundir al estudiante, lo que entorpece la asimilación del tema de estudio y el avance del curso. En general, estos sistemas asumen ciertos conocimientos y destrezas no formados aún en los principiantes, y, además, su gran tamaño y complejidad característicos tienden a provocar el efecto contraproducente de distraer la atención en aspectos de manejo y a perder de vista los conceptos esenciales que se desean enseñar. “Más es menos” es la frase con la que Motil y Epstein (1998) sintetizan muy acertadamente esta situación.

Existen algunas alternativas de índole más didáctica como son los sistemas MiniJava (Roberts, 2001), BlueJ (Kölling, 2000), JJ–Java Junior (Motil y Epstein, 1998) y Blue (Kölling, 1999). Sin embargo, aunque estos sistemas ofrecen características y enfoques a tener en cuenta como referencia, adolecen de una u otra forma de algunos de los elementos importantes que se buscan suplir integralmente en el producto del presente trabajo. Por ejemplo, algunos de estos sistemas no son “abiertos”, y por lo tanto se hace difícil emprender posibles adaptaciones para atender requerimientos particulares. En un plano menos técnico y más metodológico, el principal defecto general de algunas de estas propuestas es que sus lenguajes de base se prestan a un sobrevaloramiento

temprano de los elementos más mecánicos y sintácticos del lenguaje haciendo poco o menos énfasis en los aspectos más semánticos, como por ejemplo la adecuada representación y definición de problemas antes de pasar a la codificación algorítmica. Por otro lado, la mayoría de las alternativas, tanto en cuanto a entornos de desarrollo como a sus lenguajes de base, agregan al hispanohablante la exigencia de la traducción ya que su idioma es el inglés, aspecto quizá secundario pero que no debe soslayarse en un marco de enseñanza introductorio en el que al mismo tiempo una buena porción de los estudiantes hacen un manejo incipiente de tal idioma, y por lo tanto la traducción de términos puede competir por su atención en un momento en que el objetivo principal es el entendimiento de los conceptos de la programación.

3. La propuesta

Visualizando un marco referencial que abarque la problemática descrita, este proyecto fue emprendido concretamente en un intento por ofrecer una alternativa de sistema de programación que ayude a responder las siguientes preguntas:

- ¿Qué características debe ofrecer una herramienta computacional de programación (tanto lenguaje como entorno de desarrollo) diseñada específicamente para fines didácticos en cursos introductorios a la programación de computadores que apoye al instructor de forma efectiva en la obtención de resultados?
- ¿Contribuye a formar una disciplina más propicia hacia el desarrollo profesional de software el hacer que el lenguaje de programación obligue a una separación explícita y temprana entre el qué debe hacerse y el cómo debe hacerse para abordar problemas computacionalmente?

De un modo u otro, conciente o inconcientemente, todo instructor de introducción a la programación está involucrado con la primera pregunta. Así mismo, todo sistema de naturaleza didáctica representa una propuesta explícita para su solución. Como se verá más adelante, la segunda pregunta tiene una relación más estrecha con el planteamiento particular del presente proyecto.

Este proyecto no pretende dar una respuesta definitiva a ninguna de estas preguntas, como tampoco pretende en absoluto cuestionar a fondo las reales posibilidades didácticas de los sistemas existentes. Lo que sí persigue este trabajo es ofrecer una herramienta que se constituya en un apoyo adicional que permita una aproximación más a las respuestas. En particular, la herramienta privilegiará el aspecto semántico de la programación (haciendo una separación explícita entre el *qué* y el *cómo* sobre un esquema inspirado en el diseño por contrato), y, con un alcance más general, se caracterizará principalmente por ser de naturaleza abierta, lo que en conjunto buscará enriquecer el bagaje de recursos tanto para el profesor como para el principiante que ayude en el proceso de enseñanza-aprendizaje de la programación.

4. Objetivos

4.1. Objetivo general

Diseñar e implementar un lenguaje y un sistema integrado de desarrollo de programas para fines de apoyo en la enseñanza-aprendizaje de la programación de computadores a nivel de principiante en carreras de computación y afines, que apoye al instructor en la presentación de conceptos introductorios, familiarice al estudiante para la transición posterior tanto a lenguajes como a sistemas más sofisticados, e incluya mecanismos flexibles de extensión para enriquecer su funcionalidad.

4.2. Objetivos específicos

1. Definir un lenguaje cuya estructura sintáctica refleje un compromiso explícito con su semántica de tal manera que se reduzca el riesgo de confusión, se facilite la gradual introducción de conceptos, y se fomenten buenas prácticas de diseño, documentación y corrección, en general bajo los lineamientos del diseño por contrato, y en particular como base para abordar el paradigma de orientación a objetos.
2. Permitir de manera integrada y uniforme la codificación, compilación, ejecución y depuración de programas, ofreciendo facilidades de navegación y reutilización de programas existentes.
3. Posibilitar la extensión del repertorio de programas disponibles (bibliotecas) de tal manera que el instructor pueda enriquecer las posibilidades de la herramienta según el tópico en curso.
4. Construir una infraestructura de base para posibilitar la continuación posterior del proyecto en un estilo de “código abierto”, para su mantenimiento y ampliación por parte de una comunidad de interesados como educadores, estudiantes y programadores.

5. Lo que el proyecto es

Los componentes básicos de este proyecto son, por una parte, la definición de un lenguaje de programación propio, y, por la otra, el diseño e implementación de un ambiente de desarrollo de programas que dé soporte a la codificación en dicho lenguaje. Ambos componentes deben integrarse íntimamente, incluyendo mecanismos de manejo que representen menos compromisos para el instructor de tal manera que se facilite llevar a cabo un trabajo más progresivo de acuerdo con los tópicos de enseñanza.

Tanto el lenguaje como el ambiente de trabajo deben fomentar algunas de las prácticas ampliamente aceptadas de diseño y construcción de software para un adecuado acercamiento al paradigma de la orientación a objetos, partiendo de un esquema basado en contratos (Meyer, 1997; Blom et al, 2001), en el que se haga una separación expresa entre el qué debe hacerse y el cómo debe hacerse. En cuanto a lenguaje, Brosgol (2000) se expresa así: “Si bien el papel relativo que juega un lenguaje de programación en contraste con otros factores (calidad del compilador y otras herramientas de soporte, la metodología de desarrollo, el talento personal) ha sido por mucho tiempo motivo de debate, es claro que las características de un lenguaje pueden o bien inhibir o bien promover una buena ingeniería de software.” En términos generales, el sistema está dirigido a fomentar sólidas bases hacia la ingeniería de software, entendida ésta como el conjunto de principios y técnicas de soporte para el desarrollo predecible de sistemas correctos, adaptables y eficientes.

Puesto que es imposible prever la variedad de contextos de aplicación y de requerimientos particulares, la herramienta debe ser fundamentalmente un producto abierto, no sólo en cuanto a caracteristicas de la interfaz gráfica de usuario, sino también a nivel del mismo lenguaje, que sea susceptible de adaptarse a diversas circunstancias y necesidades. Para posibilitar esto, el proyecto se encuentra montado en una infraestructura de gestión apropiada en Internet, la cual provée facilidades de comunicación (entre desarrolladores y otros interesados), de actualización de software y documentación, descarga de versiones, noticias, seguimiento a anomalías (bugs), control de versiones (estilo CVS), y el mantenimiento de listas de correo, entre otros servicios.

Si bien la idea motivadora de este trabajo ha sido la de disponer de una herramienta de programación diseñada específicamente para servir de apoyo en la enseñanza y el aprendizaje a nivel de principiante, también se busca que el proyecto sea aprovechable en estudios más avanzados de computación, en donde los estudiantes puedan involucrarse con los propios mecanismos internos del sistema, como puede ser en cursos sobre diseño e implementación de lenguajes, programación orientada a objetos, patrones de diseño, procesadores de lenguajes, en todo caso con la posibilidad de modificar y complementar el sistema según convenga.

6. Lo que el proyecto no es

- No es un tutor. Aunque los elementos de programación incorporados buscan fomentar algunas prácticas que se consideran beneficiosas en desarrollo de software tanto a nivel general, como a nivel de principiante, el proyecto no contempla incluir características o funcionalidades asociadas típicamente a los sistemas tutores, en los que, por ejemplo, se siguen ciertas secuencias preestablecidas de acción para propósitos específicos, o el seguimiento de ciertas metodologías.

- No es un sistema de ejercitación y práctica, pues no supone los conocimientos previos ni presenta al usuario escenarios de entrenamiento de acuerdo con sus capacidades.
- No es un simulador o juego educativo, pues no representa una realidad reducida del problema en la que el usuario controle variables de ambiente o haga pruebas del estilo “que pasaría si…”
- No es un sistema experto, pues no presenta el conocimiento de un experto ni se construye con base en éste.
- No es una propuesta de modelo didáctico. Un modelo didáctico está más involucrado con la forma en que se enseña o se debe enseñar una cierta área de estudio, en donde pueda verbalizarse sobre el mismo aprendizaje y se formulen preguntas de investigación, en este caso en estudios sobre la enseñanza de la programación (Kaasbøll, 1999).
- No es una propuesta de método de solución de problemas. Es el instructor quien decide en este aspecto aplicando, en donde considere conveniente, el diseño por contrato fomentado por la herramienta y su lenguaje.
- No incluye una evaluación de tipo educativo. Aunque se ha mantenido un acompañamiento de tipo apreciativo de parte de algunos profesores y estudiantes, que ha servido para guiar la revisión y mejoramiento de la herramienta durante su mismo desarrollo, este trabajo no incluye la realización de una investigación cuantitativa y rigurosa sobre el aporte educativo propiamente dicho.

7. Descripción general del sistema

En esta sección se hace una descripción sintetizada de las principales características de los dos componentes centrales del sistema construido. El resto del documento se dedica a exponer los elementos que han servido de base para dicha construcción.

7.1. El lenguaje

En este proyecto se define un lenguaje de programación de carácter didáctico con revisión estricta de tipos y reciclaje automático de memoria. Su soporte central se inspira en el diseño por contrato aplicado a la programación imperativa, buscándose una fundamentación conceptual que sea, en primer término, extensible hacia la programación orientada a objetos y, a mayor alcance, aprovechable en la práctica de la programación en general.

Las tareas del mundo real se enfocan en el lenguaje como procesos que reciben, manipulan y producen información. En este sentido hay dos enfoques bien

demarcados pero complementarios. Cada problema se describe como un proceso cuya relación entrada/salida satisface los requerimientos correspondientes, todo esto desde un punto de vista externo; pero también un proceso involucra un mecanismo interno, el cual debe activarse para que se logre la solución efectiva al problema. El lenguaje incluye una construcción particular, llamada `especificación`, para representar los problemas desde el punto de vista externo, y una construcción particular, llamada `algoritmo`, para representar las posibles soluciones a tales problemas. En el lenguaje definido no es posible escribir un algoritmo sin antes especificar el problema que se desea solucionar. El esquema "primero especificar, luego implementar", da mayor énfasis desde el principio al carácter semántico de la programación, circunscribiendo los detalles sintácticos y algorítmicos del lenguaje en una categoría bien definida, más fácil de identificar y diferenciar para efectos de enseñanza.

7.2. El entorno

La integración del producto abarca las distintas fases del desarrollo de programas, desde el procesamiento del código fuente, pasando por la compilación, la generación de documentación, la explicación de errores, hasta la ejecución (completa o paso-a-paso) y la interpretación interactiva, entre otros. Aunque estas características se asocian generalmente con sistemas profesionales de desarrollo, este proyecto las contempla desde una perspectiva introductoria más apta para el principiante, concretamente, mediante una interfaz más sencilla, es decir, con un menor número de elementos gráficos y menos opciones y comandos, en todo caso ofreciéndose una familiarización que facilite la transición posterior a herramientas más sofisticadas.

La organización del trabajo en el entorno se hace con base en proyectos. Un proyecto es un conjunto relacionado de elementos que conforman un desarrollo, un tema o un problema particular. El elemento central es un diagrama estilo UML que muestra los componentes del proyecto. Se destina asímismo una ventana de edición para cada una unidad particular. La documentación de cada unidad se genera automáticamente en formato HTML como acción complementaria en cada compilación exitosa, y puede visualizarse en una ventana para ello. Cada sesión de ejecución se lleva a cabo en una ventana que puede ser lanzada desde la ventana principal del proyecto o desde la ventana de edición del algoritmo respectivo. Se dispone también de una ventana para el intérprete interactivo en donde puede lanzarse la ejecución de algoritmos e incluso ejecutarse fragmentos de código, o hacerse la evaluación de expresiones individuales. Las variables declaradas desde el intérprete interactivo, que pueden desplegarse convenientemente en estilo tabular, son accesibles desde otras ventanas de ejecución como posibilidad para suministrar valores de entrada y/o recibir valores de salida. Todo lo anterior conlleva facilidades para la interacción con el sistema, además de facilitar una pronta aproximización al mismo lenguaje para fines introductorios.

El esquema conceptual de separación especificación-algoritmo que se refleja en el lenguaje, resulta especialmente apto para incluir la posibilidad de llevar a cabo un *desarrollo basado en pruebas*. En efecto, esta característica está incluida en el entorno integrado y básicamente se trata de llevar a cabo los siguientes pasos: i) escribir una especificación; ii) generar un esquema de pruebas para la especificación; iii) escribir código de puebas; iv) escribir uno o más algoritmos para la especificación, y v) ejecutar las pruebas. El entorno ayuda en la realización automática de los pasos ii) y v).

8. Bases del sistema

Los tópicos esenciales para los objetivos de este proyecto se pueden agrupar en dos temas principales: el primero relacionado con las características de uso deseables para una herramienta de apoyo a la enseñanza de la programación, es decir, desde el punto de vista de su utilización por parte de estudiantes y profesores en cursos introductorios; y el segundo en relación con los elementos técnicos y de diseño que permiten la construcción de la herramienta como tal. Este documento se centra en el primer aspecto.

8.1. Patrones de diseño

Los denominados “patrones de diseño” (Gamma et al, 1994) conforman el marco conceptual general para este trabajo. Se trata de una refinada disciplina tanto de desarrollo de software como de comunicación profesional que se constituye no sólo en el soporte de diseño más importante para la construcción de la herramienta, sino que también incluye algunos de los elementos conceptuales que se desean reflejar para fines de enseñanza.

Un patrón de diseño es una descripción de objetos y clases interactuantes que se acondicionan para resolver un problema de diseño general en un contexto particular (Gamma et al, 1994). En este sentido, los patrones identifican y especifican abstracciones que están por encima del nivel de las clases y objetos como tales.

El campo de los patrones de diseño se remonta por lo menos a comienzos de los años 1980, momento en que era común hablar de programación estructurada pero aún no estaba difundida la programación orientada a objetos, en donde Smalltalk era el lenguaje más representativo mientras que C++ estaba en sus inicios. Sin embargo, algunas ideas contaban con algún nivel importante de elaboración, como es el caso del conocido esquema de diseño “Modelo-Vista-Control” (MVC) que se desarrolló para Smalltalk. En ella se divide la interfaz de usuario en tres partes: el modelo de datos, que contiene la lógica de la aplicación, la vista, que se encarga de la interfaz al usuario propiamente, y el control, que media entre las dos primeras. En este proyecto se sigue el patrón

MVC como estructura de diseño para el entorno integrado de programación (sección 3.3.2).

Los patrones de diseño se constituyen en un aporte fundamental para este proyecto no sólo en cuanto a estructura medular de construcción, sino también porque ellos mismos están basados en la aplicación de algunos principios conceptuales que precisamente deberían presentarse y fomentarse a los estudiantes desde el nivel de principiante. Entre tales principios hay que destacar en especial el siguiente (Gamma et al, 1994):

Programar para una interfaz, no para una implementación

Este principio significa hacer que los objetos en un sistema computacional interactúen a través de las operaciones declaradas en las correspondientes interfaces, sin conocimiento previo de las clases de implementación particulares. Este principio repercute en mayor flexibilidad y modularidad.

En el sentido propiamente educativo, es pertinente señalar que algunos autores (por ejemplo, Nguyem y Wong, 2001) proponen también involucrar los patrones de diseño de forma explícita desde el principio. Su argumento es que justo de esto se trata la ciencia de la computación, esto es, del entrenamiento del pensamiento abstracto, del cual los patrones de diseño son un magnífico ejemplo.

8.2. Características didácticas

En esta sección se describen los rasgos deseables tanto para un lenguaje como para un entorno de programación desde un punto de vista didáctico. También se hace una breve exposición de algunas de las principales posturas en esta temática. En especial, se destacan aquellas propuestas que se enfocan principalmente en el aspecto semántico de la programación. Para más profundización en el tema, consúltense las referencias.

8.2.1. Características de un lenguaje didáctico

La siguiente es una lista seleccionada de características deseables para los lenguajes de programación en general (Finkel, 1996), concretamente con aquéllas que favorecen fines didácticos (Kolling, 1999), según los propósitos de este trabajo:

- Conceptos bien definidos. Es importante que el lenguaje refleje un compromiso directo con los conceptos básicos que se desean enseñar. Esto se logra en parte haciendo que el lenguaje involucre construcciones sintácticas que representen de manera directa y expresa los conceptos correspondientes.
- Diseño por contrato. El lenguaje debe promover la documentación de los procesos (sus entradas y salidas), así como el uso de precondiciones,

poscondiciones, y afirmaciones dentro del código, de tal manera que se propicie la corrección de los programas y se enfaticen los aspectos semánticos de la programación.

- Consistencia sintáctico-semántica. Es importante que cada construcción sintáctica se asocie con un sentido específico para evitar confusiones y redundancias. Esto permite que el lenguaje sea más compacto, claro y fácil de aprender.
- Alto nivel. Esta es una característica importante que permite concentrar esfuerzos en los conceptos fundamentales, y no distraerse en aspectos internos demasiado complejos relacionados con la máquina subyacente o con la organización y manipulación de la memoria.
- Seguridad. El lenguaje debe contar con un fuerte sistema de tipos en tiempo de compilación, detección de variables no inicializadas y no incluir operaciones sobre apuntadores (estilo C) reconocidas como peligrosas. Cualquier tipo de error, sea en compilación o ejecución, debe ser detectado y reportado con el mayor detalle posible tan pronto se presente.
- Fácil transición. Es importante que los conceptos fomentados por el lenguaje tengan una manifestación real y palpable en lenguajes subsecuentes. En cuanto a eficiencia es de anotar que si bien es considerada como una de las características críticas de los lenguajes de programación en general, ésta no hace parte de las prioridades en el caso de un lenguaje didáctico. Desde luego, debe lograrse por lo menos un desempeño aceptable que haga práctica la utilización de la herramienta.

8.2.2. Características del entorno

También es fundamental el disponer de un sistema de trabajo integrado apto para el perfil de un principiante (Findler et al, 2001; Motil y Epstein, 1998; Kölling, 1999). Las características deseables en este caso son:

- Facilidad de uso. El entorno debe permitir una interacción inmediata y transparente con el lenguaje, reduciendo en lo posible cualquier distracción innecesaria con respecto al manejo del entorno como tal.
- Integración. El entorno debe integrar las diversas actividades típicamente asociadas con el desarrollo de programas como la edición, compilación, ejecución, depuración y visualización de documentación.
- Reutilización de código. Se debe permitir la visualización cómoda de los elementos de código disponibles en bancos existentes para su eventual referencia y reutilización en programas nuevos, así como la creación y actualización de nuevos bancos de código, buscando que el estudiante se familiarice prontamente con este importante rasgo de la práctica de la programación.

- Modo intérprete. El entorno debe ofrecer un modo intérprete para permitir la ejecución de fragmentos de código de manera interactiva como facilidad para hacer pruebas y experimentación.
- Fácil transición. De forma similar a como se dijo sobre el lenguaje, es importante que los diferentes componentes y teminología utilizados en el entorno de desarrollo tengan una manifestación similar y coherente en sistemas subsecuentes.

8.3. El primer lenguaje

Este trabajo se enfoca en estudiantes que se desempeñarán profesionalmente en áreas relacionadas estrechamente con la programación, como puede ser el caso de los programas de tecnologías o ingenierías de sistemas o software y en general cualquier estudio formal relacionado con las ciencias de la computación.

En este contexto, Robinson (1994) presenta una serie de criterios para la elección del lenguaje a ser impartido inicialmente a estudiantes de pregrado en ciencias de la computación, quienes deben comenzar con una sólida fundamentación en programación sobre la cual establecer los principios que serán aplicados subsecuentemente en muchos lenguajes diferentes. Robinson menciona tres objetivos principales:

- Base matemática: Los estudiantes necesitan ver los programas como descripciones formales de algoritmos abstractos.
- Uso estricto de tipos: El gran valor del manejo de tipos en la escritura de programas correctos y mantenibles está bien establecido. Esto es particularlmente importante en la evolución de grandes sistemas en donde un equipo de programadores puede estar trabajando por varios años.
- Enfasis funcional: Un estilo funcional conduce a una programación correcta, además lleva así mismo a un análisis matemático de algoritmos.

  Además comenta: “También debe decirse que un entorno amigable para la experimentación es una gran virtud, lo que probablemente implica el uso de un lenguaje interpretado.” En este proyecto en particular, se hace énfasis en el manejo estricto de tipos, el seguimiento en parte de un estilo funcional y la posibilidad de la modalidad intérprete para la ejecución de fragmentos de código y la evaluación de expresiones.

8.4. Modelo de programación

¿Hay un modelo de programación idóneo para enseñar a programar? En principio, esta pregunta podría responderse con programación “orientada a objetos” (OO) teniendo en cuenta que tal paradigma no sólo ha sido adoptado casi

como norma en términos industriales, sino que incluso ha ganado una amplia aceptación a nivel de enseñanza (Kölling, 2001).

Sin embargo, el hecho es que no hay un consenso general sobre sus verdaderas bondades educativas o sobre métodos efectivos para impartir dicho paradigma en el aula. Algunos autores se adhieren a otras alternativas, como es el caso de la programación funcional (Findler et al, 2001; Felleisen et al, 2001; Giegerich et al, 1999). En particular, Findler et al (2002) afirman que: "... Scheme, adecuadamente apoyado, es casi ideal para la enseñanza de los principios de la programación y la computación." En Giegerich et al (1999) se hace una apreciación similar en el contexto del lenguaje Haskell. Otros autores además presentan sus propias reparos a la OO (Johnson, 1994). Incluso el mismo Stroustrup, creador del lenguaje C++, si bien promueve decididamente un alto nivel para la enseñanza a nivel introductorio, considera que enseñar programación OO pura es otro extremo que deber evitarse (Stroustrup, 1999). En general, Stroustrup ofrece más bien una receta de "sentido común" en la que se haga énfasis sobre las técnicas: "Para los aprendices de la programación, el aprendizaje del lenguaje de programación debe dar soporte al aprendizaje de técnicas de programación efectivas."

Por su parte, Brosgol (2000) plantea: "Una de las razones por las que los conceptos OO pueden ser difíciles en un curso de primer nivel, es que hay una gran distancia conceptual entre un registro que contiene campos de datos y una clase que contiene tanto datos como métodos." En su comparación, Brosgol explica que el lenguaje Ada, a diferencia de Java, no requiere familiaridad con la OO para hacer un uso efectivo del lenguaje y, por lo tanto, "le permite al instructor posponer el estudio de conceptos OO hasta que se hayan introducido las construcciones más básicas."

Ahora bien, buscando un balance, hay que tener en cuenta también el papel que juega el entorno operativo de interacción con el lenguaje. Kölling (1999), por ejemplo, argumenta que los limitados logros en la enseñanza de la programación OO no se han debido a que tal modelo sea didácticamente inadecuado, sino que han sido consecuencias de las herramientas comúnmente utilizadas como apoyo. Esta argumentación puede extenderse de manera más general, no solamente a la programación OO.

8.5. Disciplina más que paradigmas

Hay que advertir también que el programador objetivo de este proyecto se verá en cursos posteriores enfrentado a tareas de programación de características tales que no sea viable, práctico o apropiado abordarlas con estrategias OO. De hecho, otros paradigmas son más idóneos para cierta clase de problemas. Por ejemplo, la programación concurrente puede resultar bastante compleja en un lenguaje OO representativo como Java, pero resultar bastante sencilla si se usa el modelo de computación adecuado (van Roy y Haridi, 2002a, 2002b). Estas consideraciones hacen manifiesta la necesidad de contar con una infraestructura conceptual

básica sobre la cual apoyar la posterior construcción de los modelos y paradigmas de programación. Con base en una fundamentación científica rigurosa, éste es, en efecto, el esquema utilizado por el elaborado trabajo de van Roy y Haridi (2002b). En comparación con el trabajo de van Roy y Haridi (que no se enfoca concretamente a un primer curso en programación, y de hecho utiliza un estilo formal), el presente proyecto ofrece un planteamiento más introductorio y práctico hacia la ingeniería de software, en donde la pregunta que surge, como se mencionó al comienzo de este documento, es: ¿Qué conceptos introducir y enfatizar a principiantes de la programación que contribuyan a la formación de una buena displina de desarrollo de software?

8.5.1. Programación por contrato

La programación basada en contratos (Blom et al, 2001; Meyer, 1997) destaca las implicaciones semánticas de la programación, dejando en un segundo plano los detalles sintácticos y mecánicos del lenguaje utilizado. Los métodos basados en contratos se constituyen en la disciplina medular cuyas bondades pueden aprovecharse en una posterior orientación sea ésta funcional, imperativa, lógica o a objetos. Como se mencionó anteriormente al enumerar las características deseables para los lenguajes de programación (en particular, los didácticos), estos detalles deben definirse de forma que los aspectos semánticos se vean reflejados unívoca y consistentemente.

El diseño por contrato se refiere a visualizar las interfaces entre los componentes de un sistema como contratos que se especifican integralmente dentro del código fuente en el lenguaje de implementación, con el importante beneficio de que tanto clientes como proveedores logran un entendimiento preciso de sus responsabilidades. Más que pretender un seguimiento rigurosamente formal, lo que se persigue en este proyecto es sensibilizar al estudiante principiante en este sentido de tal manera que se forme una actitud que aprecie el valor de la claridad en la separación de responsabilidades para la construcción de software.

8.5.2. Desarrollo dirigido por pruebas

Una práctica ampliamente promovida en los últimos años, es la de dirigir todo desarrollo de software con base en pruebas. (Hammel y Nettleton, 2001; Astes, 2002; Marick, 2000; Gamma y Beck, 1999). El desarrollo dirigido por pruebas es un estilo de desarrollo en donde:

- Se mantiene un conjunto exhaustivo de pruebas unitarias.
- Ningún código entra a producirse a menos que cuente con pruebas asociadas.
- Las pruebas se escriben primero.

- Las pruebas determinan el código que se necesita escribir.

Un entorno de programación que contemple algunas facilidades para favorecer la aplicación de esta disciplina, resultaría altamente conveniente para fines didácticos. Por ejemplo, para permitir la ejecución automática de un conjunto de pruebas que se hayan definido previamente sobre uno o varios programas, o incluso ayudar en la generación automática de esquemas para la codificación de las pruebas.

9. Lenguajes y entornos existentes

En esta sección se describen brevemente algunas de las opciones existentes en cuanto a lenguajes y entornos desde una perspectiva de evaluación para fines de enseñanza de la programación. Se trata de descripciones generales, que no pretenden ningún tipo de revisión exhaustiva ni tampoco sentar ninguna posición crítica sobre sus verdaderas bondades educativas, pero que proporcionan un contexto adecuado con miras al lenguaje definido según los propósitos de este proyecto.

9.1. Sistemas industriales

Smalltalk

Smalltalk ha sido considerado el lenguaje orientado a objetos por excelencia de acuerdo con la concepción más pura de tal paradigma. Todo en el lenguaje es un objeto, incluso las estructuras de control. A primera vista esta pureza podría considerarse como supremamente ventajosa para fines didácticos puesto que unifica y, aparentemente, simplifica conceptos que son completamente distintos en otros lenguajes. Sin embargo, esta sobresimplificación hace que el lenguaje se vuelva oscuro para los principiantes (Kolling, 1999), es decir, repercute en poca legibilidad de los programas. En este sentido, conviene mantener la tradición de lenguajes imperativos en donde las estructuras de control de flujo (ciclos, condiciones, saltos) se ubican en una categoría totalmente distinta a la de los objetos.

C++

Por su carácter híbrido, que busca dar soporte simultáneo a los estilos procedimental y por objetos, C++ resulta ser una opción inconveniente para fines didácticos. Su diseño, en la tradición de C, está fuertemente influenciado por consideraciones de eficiencia que repercuten en construcciones de bajo nivel muy riesgosas para los estudiantes. Aunque el mismo Stroustrup (1999) viene al rescate proponiendo un enfoque apropiado para fines de la enseñanza de C++ a

estudiantes tanto principiantes como experimentados, el hecho es que este lenguaje resulta ser exageradamente complejo, permisivo, redundante y de una sintaxis bastante ilegible para los principiantes, quienes son los estudiantes objetivo de este proyecto.

Java

En comparación con C/C++, Java resulta ser una alternativa significativamente superior para eventuales fines didácticos puesto que muchos de los rasgos negativos de aquéllos han sido evitados. Sin embargo, adolece de algunos inconvenientes importantes en lo que al lenguaje como tal se refiere. Roberts (2001) lo sintetiza así: "En general, el problema surge del hecho que Java fuerza a los estudiantes a entender una serie de elementos conceptualmente sofisticados para escribir aún los programas más simples." Por ejemplo, el método que sirve de punto de entrada para lanzar la ejecución de una aplicación debe tener el siguiente encabezado:

```
public static void main(String[] args)
```

De manera prematura, esta sóla línea ya está demandando demasiadas explicaciones por parte del profesor, explicaciones que deberá aplazar necesariamente quizá generando desmotivación en algunos estudiantes.

Estas "debilidades" podrían superarse, no obstante, a través de algunos mecanismos sofisticados que ofrece el mismo lenguaje y bibliotecas de soporte (en particular, los mecanismos de "reflexión" –reflection), canalizados por un "entorno" de ejecución que los haga transparentes para el estudiante. Esto puede posibilitar, por ejemplo, la ejecución directa de cualquier método en un programa. En efecto, éste es el esquema sobre el que se apoyan muchas propuestas para enseñanza como lo son el Problem Set Framework (Proulx et al, 2002), y el sistema Bluej discutido más adelante.

En todo caso, Java resulta aún inconveniente puesto que carece de soporte a la programación por contrato, y además, aunque ofrece el poderoso mecanismo de las "interfaces," no promueve que se haga una separación sistemática entre éstas y las clases de implementación, lo cual resultaría muy favorable para que los estudiantes comiencen desde temprano a entrenarse en la filosofía "programar para una interfaz" (sección 2.1).

9.2. Propuestas didácticas

Java-Junior, JJ

JJ es un interesante lenguaje para principiantes cuya motivación, según lo anuncian sus autores (Motil y Epstein, 1999), se refleja en el lema "menos es más". En JJ sólo haya una forma para hacer cada cosa: sólo un constructor, sólo un tipo entero, sólo un tipo real, sólo un estilo de comentario, sólo una estructura

de decisión y sólo una estructura de repetición. De esta manera, los autores aseguran evitar muchas confusiones y ahorrar una gran cantidad de tiempo. JJ incluye también funciones "puras" (sin efectos colaterales), afirmaciones (precondiciones, poscondiciones, invariantes de clase) y manejo de excepciones.

En el aspecto instruccional, es ilustrativo comentar brevemente la forma cómo los autores proceden en sus cursos. Comienzan con el estilo procedimental explicando las estructuras de control de decisión y de iteración. Luego, cuidando de no dedicar demasiado tiempo en algoritmos con anidamientos complejos de decisión e iteración, pasan rápidamente a presentar algunos conceptos de flujos de datos e introducen los métodos (funciones y rutinas), así como el encapsulamiento antes de introducir la programación orientada a objetos. Los autores afirman que de esta manera reducen el riesgo de que se den preferencias tempranas y mal fundadas por algún paradigma particular.

Como uno de los objetivos centrales de JJ es servir de transición hacia Java, ellos han fijado algunas limitaciones; por ejemplo, sólo hay arreglos de una dimensión, no hay herencia ("...parece un concepto demasiado complejo para ser introducido en una etapa tan temprana"), y las excepciones se enfocan a detectar posibles errores en la invocación de ciertas rutinas incorporadas en el sistema.

Lamentablemente, JJ no es un sistema abierto; pero en todo caso es una propuesta que enriquece el panorama para los fines del presente proyecto.

Bluej

Bluej (Kolling, 2000) es una herramienta para la enseñanza de la programación a nivel de principiante en el que se utiliza Java como lenguaje. Aunque es gratuito y al parecer goza de gran aceptación, algunas veces se critica su carácter "cerrado" que dificulta la adición o modificación de sus servicios, o su acondicionamiento a requerimientos particulares.

La hipótesis básica que motivó el desarrollo de Bluej es que "la enseñanza de la orientación a objetos no es intrínsicamente más compleja que con otros paradigmas alternativos, sino que se complica por una ausencia marcada de herramientas apropiadas y de experiencia pedagógica con este paradigma" (Kolling et al, 2001).

Las debilidades pedagógicas per se de Java (comentadas anteriormente) son en parte superadas en Bluej a través de opciones de alto nivel en el momento de ejecutar programas. Esto se logra mediante una interfaz gráfica de usuario que le permite al estudiante ejecutar métodos de clase y de instancia mediante ventanas de diálogos. En particular, esto evita enfrentar a los estudiantes con el comprometedor "`public static void main(String[])`", desafortunadamente tan abusado en muchos libros de introducción a la programación por objetos.

Aunque Bluej se constituye sin duda en una referencia muy valiosa a tener en cuenta, es importante destacar también que sufre de algunas de las desventajas ya mencionadas con respecto al lenguaje Java que le sirve de base, a saber,

carencia de soporte a la programación por contrato, falta de obligatoriedad de comentarios de documentación y falta de separación sistemática de interfaces y clases de implementación.

MiniJava

MiniJava (Roberts, 2001) es un lenguaje de programación orientado a enseñanza basado en Java, que ha sido diseñado para reducir el factor de "intimidación" que los estudiantes principiantes experimentan cuando se encuentran con un sistema de la magnitud del ambiente Java. Su autor explica que a través de un entorno más simplificado como MiniJava, los estudiantes podrán enfocarse en los aspectos conceptuales de la programación sin verse acosados en demasiados detalles. En MiniJava se incluyen también algunas adiciones con el fin de hacer más fácil la enseñanza a los novatos. Por ejemplo, incluye una ventana de interfaz habilitada para el típico ciclo "leer-evaluar-imprimir" encontrado principalmente en entornos de lenguajes interpretados. Citando a Roberts: "Tener la posibilidad de evaluar simples expresiones de manera inmediata es una herramienta maravillosa para estudiantes que tienen problemas en entender cómo operan los conceptos subyacentes."

DrScheme

DrScheme (Findler et al, 2001) es una propuesta pedagógica basada en el lenguaje Scheme. Según las mismas palabras de sus creadores, "DrScheme es un ambiente de programación integrado e interactivo diseñado específicamente teniendo en mente las necesidades de los principiantes." Ofrece un muy sofisticado entorno de programación en el que se incluyen diferentes modos operativos de acuerdo con el nivel del estudiante (principiante, intermedio, avanzado). También incluye el manejo de "paquetes de enseñanza" (teachpacks), que vienen a ser proyectos en áreas específicas con el fin de "conectar el código de los estudiantes con paquetes de rutinas avanzadas en gráficas, redes y otros."

Python

Python es un lenguaje para guiones (scripting) no sólo utilizado extensamente a nivel profesional, sino también, de acuerdo con su creador (Rossum, 1999), enfocado a propósitos de enseñanza. Stajano (1999) lo explica así: "La principal ventaja de Python como lenguaje introductorio es su alto nivel de abstracción, apropiado para introducir los conceptos fundamentales de los algoritmos sin distraerse en detalles irrelevantes como las micro- optimizaciones al nivel de máquina y los enredos de la asignación de memoria. Naturalmente, los futuros científicos de la computación tendrán que introducirse en tales detalles en algún momento, pero esto puede suceder después, en el contexto de explicarles cómo se hicieron los propios bloques constructivos de Python."

Python es sin duda una referencia a tener en cuenta, pero, sin pretender ningún cuestionamiento de fondo, es de anotar que adolece de una de las características mencionadas como deseables para un lenguaje de enseñanza, a saber, la revisión estricta de tipos en tiempo de compilación. En general, la ausencia de revisión de tipos es un rasgo típico de los lenguajes interpretados, en parte porque están enfocados a servir de herramientas de desarrollo rápido bajo el control de programadores experimentados. Esto de por sí no descarta a Python (ni a alternativas similares) como un buen candidato para primer lenguaje, pero deja por entero en manos del instructor la explicación y promoción de ciertas disciplinas de desarrollo de software como las relacionadas con el concepto de tipos de datos.

10. Estado actual

Se puede hacer el siguiente balance de resultados con respecto a los aspectos didácticos presentes en la actual versión 0.8 del sistema:

1. Se ha hecho la definición de un lenguaje de programación en donde el criterio de la consistencia ha jugado un papel fundamental. Contrario a pretender un lenguaje "breve", es decir, de pocas palabras reservadas, o de palabras o frases abreviadas y demasiado uso de símbolos no alfabéticos, se dio mayor importancia a la claridad expresiva y verbal. En particular, toda construcción de anidamiento (acción compuesta que incluye a su vez acciones) se establece con la debida sintaxis de inicio y final del bloque. A un nivel más semántico, el lenguaje se fundamenta en una estructura conceptual inspirada en el diseño por contrato en donde se debe comenzar con la debida especificación de los problemas antes de pasar a su solución computacional algorítmica. Este esquema de programación basada en contratos se complementa, por una parte, con los elementos léxicos de documentación que hacen parte integral de las principales construcciones del lenguaje, y por otra, con opciones desde el entorno integrado que facilitan la práctica de un desarrollo de programas dirigido por pruebas.

2. En cuanto a entorno de programación, se diseñó y construyó un sistema centrado en proyectos para el desarrollo de programas incluyendo las diversas fases que típicamente se asocian en la práctica, además de otras características más novedosas como el mencionado manejo de aplicación de pruebas unitarias (*unit testing*), edición y ejecución de guiones de demostración y seguimiento explicativo paso a paso en la ejecución de programas. Todo esto se lleva a cabo de una manera integrada y consistente. En particular, es de mencionar el caso de la tabla de declaraciones globales, la cual es persistente y accesible desde cualquiera de las sesiones iniciales de ejecución para efectos de permitir el suministro de argumentos y las asignaciones de valores de resultado.

3. Para permitir de manera cómoda y organizada la extensión del repertorio de programas disponibles, se incluyó, por una parte, el concepto de paquetes para la visibilidad de los elementos de software en el lenguaje, y, por la otra, el esquema basado en proyectos para el entorno de desarrollo; todo lo cual trae ventajas importantes de modularidad, claridad y flexibilidad. Además, se construyó un mecanismo de implementación de algoritmos en Java (a través de BeanShell) que permite el desarrollo de bibliotecas sofisticadas aprovechando la infraestructura del lenguaje huésped sin abandonar la disciplina de especificaciones para la debida interfaz con el ambiente Loro.

4. El sitio web del proyecto se ha constituido efectivamente en el canal de base tanto para el desarrollo como para la comunicación con los usuarios. En cuanto a desarrollo, el servicio prestado por SourceForge ha facilitado el seguimiento de procedimientos para la actualización y control de versiones, mantenimiento de listas de correo y el registro tanto de anomalías como de peticiones de extensiones y nuevas características.

En general, la herramienta construida maneja, a un nivel comparativamente más simple, elementos típicamente encontrados en sistemas más sofisticados como el de la organización por proyectos, edición con coloreamiento de sintaxis, visualización y navegación de documentación en formato HTML, y ejecución de programas con seguimiento paso a paso, entre otros. Esto contribuye a facilitar la posterior transición que necesariamente el estudiante habrá de dar hacia sistemas industriales de desarrollo.

11. Conclusión y trabajo futuro

Como se desprende de la revisión bibliográfica realizada en este trabajo, no se cuenta con un panorama de consenso general sobre cómo atacar efectivamente la problemática de la enseñanza de la programación. Por el contrario, se presentan posiciones diversas y hasta encontradas, situación que sigue dejando en manos del profesor la decisión de cuál metodología y cuáles recursos utilizar en el aula de clase. En este proyecto se partió de un reconocimiento claro de esta situación, de ahí que, en un afán por hacer una delimitación adecuada, se haya enfocado concretamente al campo de los materiales de apoyo para el aprendizaje de la programación a nivel de principiante. Aunque existen algunas alternativas disponibles en dicho campo, este proyecto se realizó con el propósito de ofrecer un recurso adicional, fundamentalmente de carácter abierto, que apoye al instructor en la enseñanza de la programación introductoria fomentando buenas prácticas de ingeniería de software.

De acuerdo con el balance de resultados descrito anteriormente, el proyecto ha alcanzado un nivel de desarrollo que en buena medida rebasa los propósitos que se trazaron en un principio. Esto no significa que ya se cuente con un producto definitivo y, menos aún, con una solución probada para apoyar la enseñanza de la

programación. Más bien, el principal resultado ha sido el de ofrecer una herramienta que puede constituirse en un elemento más para ayudar a un acercamiento a esta problemática, por lo menos ayudando a clarificar las preguntas, y, quizá más ambiciosamente, permitiendo emprender otros trabajos aun más involucrados con la dimensión educativa, en donde, ciertamente, este desarrollo computacional debe encontrar el necesario complemento.

Trabajo futuro

Una de las primera alternativas de continuación es la realización de estudios enfocados concretamente a evaluar la herramienta en cuanto a sus reales ventajas y desventajas frente a otros materiales para la enseñanza de la programación. Un estudio de este tipo ayudaría no sólo a clarificar los aspectos operativos que ameritan mejorarse, sino también a precisar en general las fortalezas y debilidades didácticas para una extensión futura del sistema a más largo plazo.

Reconociendo que aún en el mismo campo de los materiales de apoyo es mucho lo que aún hay por hacer y entender, uno de los caminos más interesantes de continuación a más largo plazo, es hacer las modificaciones necesarias orientadas a construir una infraestructura más amplia y flexible para que se disponga de más opciones de configuración y operación del sistema de programación. Podría explorarse la posibilidad de diseñar un "*framework*" para construir herramientas de apoyo en la enseñanza de la programación con algunas características como las siguientes:

- Multi-paradigma. El proyecto se ubica actualmente en el modelo imperativo con extensión a la orientación a objetos, pero sería interesante estudiar la forma de posibilitar otros paradigmas. Roy y Haridi (2002) ofrecen un planteamiento que podría servir para definir un posible camino en este sentido.
- Multi-lenguaje: Actualmente, el sistema ofrece un único lenguaje para fines de especificación, permitiendo que la implementación se haga bien sea en Loro o en Java. Podría explorarse la posibilidad y pertinencia de incluir otras alternativas, no sólo con lenguajes existentes, sino incluso la definición de nuevos lenguajes o variantes a uno ya incorporado.
- Multi-idioma. Se trata de habilitar el sistema para que pueda ofrecerse sobre diferentes idiomas naturales.
- Ejecución distribuida. Una posible aproximación al campo de la ejecución distribuida es la de implementar un mecanismo de ejecución remota de algoritmos, construido sobre el esquema de invocación de métodos remotos de Java (RMI).

- Trabajo colaborativo y gestión de proyectos. En un contexto de administración de cursos, una posibilidad es la de implementar un sistema de gestión de proyectos, quizá interconectado con un sistema existente de manejo y comunicación de usuarios-grupos, para la asignación y entrega de tareas, utilizando protocolos de transporte estándares como HTTP, FTP, o email, entre otros, y complementada con los mecanismos de seguridad (encriptamiento, comunicación segura) que se consideren necesarios.

Referencias

ASTELS, Dave. What is Test-driven Development?. The Coad Letter. Issue 93. 2002.

BAEZA YATES, Ricardo. Diseñemos todo de nuevo: Reflexiones sobre la computación y su enseñanza. En: Revista Colombiana de Computación. Vol. 1. No. 1. Diciembre de 2000. ISSN 1657-2831.

BLOM, Martin; NORDBY, Eivind; BRUNSTROM, Anna. Teaching Semantic Aspects of OOP. Fifth Workshop on Pedagogies and Tools for Assimilating Object Oriented Concepts OOPSLA'01, Octubre, 2001. http://www.cs.umu.se/ jubo/Meetings/OOPSLA01/Program.html

BROSGOL, Benjamin M. A Comparison of Ada and Java as a Foundation Teaching Language. Ada Core Technologies. 2000.

FELLEISEN, M., Findler, R. B., Flatt M., Krishnamurthi, S. How to Design Programs: An Introduction to Computing and Programming. The MIT Press. 2001. http://www.htdp.org/2002-09-22/Book/

FELLEISEN, M., Findler, R. B., Flatt M., Krishnamurthi, S. The Structure and Interpretation of the Computer Science Curriculum. Functional and Declarative Programming in Education (FDPE02). 2002. http://people.cs.uchicago.edu/ robby/publications/papers/htdp-sicp-fdpe2002.pdf

FINDLER et al. DrScheme: A Pedagogic Programming Environment for Scheme. Department of Computer Science. Rice University. 2001. http://www.ccs.neu.edu/scheme/pubs/

GAMMA, Erich y BECK, Kent. JUnit Test Infected: Programmers Love Writing Tests. 1999. http://junit.sf.net/doc/testinfected/testing.htm.

GAMMA, Erich, HELM, Richard, JOHNSON, Ralph y VLISSIDES, John. Design Patterns: Elements of Reusable Object Oriented Software. Addison-Wesley. 1994.

GIEGERICH, R., HINZE, R., y KURTZ, S. Straight to the Heart of Computer Science via Functional Programming. Proceedings of the Workshop on Functional and Declarative Programming in Education. Paris, 29 September 1999. http://www.cs.rice.edu/ matthias/FDPE99/

HAMMEL, Thomas y NETTLETON, Robert. Test First, Code Later. Java Developer's Journal. Volumen 7, No. 2. 2001.

JOHNSON, Stephen C. Objecting to Objects. USENIX Technical Conference. San Francisco, CA. Enero, 1994.

KAASBøLL, Jens. Exploring Didactic Models for Programming. Department of Informatics. University of Oslo. Norway. 1999. http://citeseer.nj.nec.com/330187.html

KAASBOLL, Jens. Learning and Teaching Programming. Department of Informatics. University of Oslo. Norway. 2000.

KÖLLING, Michael. The Design of an Object-Oriented Environment and Language for Teaching. PhD. Thesis. Basser Department of Computer Science. University of Sidney. 1999. http://www.mip.sdu.dk/ mik/blue/

KÖLLING, Michael. The BlueJ Tutorial. 2000. http://www.bluej.org/tutorial/tutorial.pdf

KÖLLING, Michael, QUIG, Bruce, ROSENBERG, John. The BlueJ System and its Pedagogy. Position Paper. Fifth Workshop on Pedagogies and Tools for Assimilating Object Oriented Concepts OOPSLA'01, Octubre, 2001. http://www.cs.umu.se/ jubo/Meetings/OOPSLA01/Program.html

MARICK, Brian. Testing for Programmers. Testing Foundations. 2000. http://www.testing.com/http://www.testing.com/writings/half-day-programmer.pdf

MEYER, Bertrand. Object-Oriented Software Construction. Prentice Hall. 1997.

MOTIL, John, y EPSTEIN, David. JJ: A Language Designed for Beginners (Less Is More). 1998. http://www.publicstaticvoidmain.com

PROULX, Viera K., RASALA, Richard, y RODRIGUES, Jason Jay. Simple Problem Solving in Java: A Problem Set Framework. CCSC-NE 2002. en: Computing in Small Colleges. Abril 2002. http://www.ccs.neu.edu/jpt/ProblemSetFrame/

REINFELDS, Juris. Programming as an Engineering Discipline. 32nd ASEE/IEEE Frontiers in Education Conference (FIE 2002). 2002. http://www.info.ucl.ac.be/people/PVR/FIE2002paperFin.pdf

ROBERTS. Eric. An Overview of MiniJava. Proceedings of the 32nd SIGCSE Techical Symposium on Computer Science Education, pp 1 – 5. 2001. http://excell.stanford.edu/Excell_Archives/MiniJava.pdf

ROBINSON, Peter. From ML to C via Modula-3 an approach to teaching programming. Dic. 1994. http://www.cl.cam.ac.uk/~pr/mlm3/mlm3.html.

ROSSUM, Guido van. Computer Programming for Everybody: A Scouting Expedition for the Programmers of Tomorrow. Funding proposal sent to DARPA. CNRI Proposal #90120. July 1999. http://www.python.org/doc/essays/cp4e.html Página web del proyecto: http://www.python.org/cp4e

ROY, Peter Van, y HARIDI, Seif. Teaching computer programming as unified discipline with a practical scientific foundation. 2002a. http://www.info.ucl.ac.be/people/PVR

ROY, Peter Van, y HARIDI, Seif. Concepts, Techniques, and Models of Computer Programming (Borrador, Agosto 26/2002). 2002b. http://www.info.ucl.ac.be/people/PVR/book.pdf

ROY, Peter Van, y HARIDI, Seif. Teaching Programming with the Kernel Language Approach. Workshop on Functional and Declarative Programming in Education (FDPE02). Octubre, 2002c.

RUEDA, Carlos A. LORO - Lenguaje Orientado a Objetos, Nivel I. Especificación 0.97. Informe Técnico, Área de Computación. Facultad de Ingeniería de Sistemas. Universidad Autónoma de Manizales. Manizales, mayo 1997.

STAJANO, Frank. Python in Education: Raising a Generation of Native Speakers. University of Cambridge Computer Laboratory. Python 8 Conference Proceedings, Octubre 1999. http://www.python.org/workshops/2000-01/proceedings/papers/stajano/stajano.html

STROUSTRUP, Bjarne. Learning Standard C++ as a New Language. The C/C++ Users Journal. Mayo, 1999.