Propuesta de una Herramienta de Soporte para el Desarrollo de Habilidades de Programación en Estudiantes No Videntes

María Julia Blas
INGAR Instituto de Desarrollo y Diseño
UTN, CONICET
Avellaneda 3657, 3000 Santa Fe, Argentina
mariajuliablas@santafe-conicet.gov.ar

Diego García Lozano
Facultad Regional Santa Fe
Universidad Tecnológica Nacional
Lavaisse 610, (3000) Santa Fe, Argentina
diegogarcialozano95@gmail.com

Marta Castellaro
Facultad Regional Santa Fe
Universidad Tecnológica Nacional
Lavaisse 610, (3000) Santa Fe, Argentina
mcastell@frsf.utn.edu.ar

Abstract

Dentro del ámbito universitario, la accesibilidad a personas con capacidades diferentes es un área de investigación y atención. La existencia de herramientas de software que buscan contribuir a mejorar el aprendizaje en personas no videntes ha crecido, pero su aplicación y grado de apoyo varían según las disciplinas y habilidades que se propone desarrollar. En este sentido, no siempre es posible dar con una aplicación justa para los objetivos perseguidos por una determinada cátedra. Aunque la mayoría de las herramientas disponibles contribuyen a la labor cotidiana, existen aspectos no contemplados en este tipo de aplicaciones. En este trabajo, se presenta una herramienta de soporte a la tarea de enseñanza de programación basada en las experiencias obtenidas a lo largo del primer cuatrimestre del ciclo lectivo 2016 en un curso de primer año de Ingeniería en Sistemas de Información dentro del cual uno de los alumnos es no vidente. Las dificultades detectadas junto con las soluciones adoptadas por los docentes, constituyen las bases de la herramienta propuesta; dando lugar a una automatización de las estrategias utilizadas para analizar códigos C++.

1. Introducción

El desarrollo de habilidades de programación es un tema prioritario en la enseñanza de ciencias de la computación. Son muchos los autores que a lo largo de los años han realizado numerosos esfuerzos para facilitar esta tarea. Desde la programación por demostración [1], [2] hasta la programación visual [3] (pasando por la programación mediante ejemplos [4], programación gráfica [5] y programación física [6]), el número de herramientas de programación preparadas para cubrir las necesidades específicas de los usuarios se ha incrementado.

En la actualidad, la mayoría de las personas puede acceder a herramientas de desarrollo que facilitan el aprendizaje de técnicas de programación en base al análisis de diferentes aspectos y/o propiedades. Esto ha dado lugar a usuarios finales con mejores habilidades para programar. Sin embargo, la mayoría de estas herramientas utiliza interfaces de usuario centradas en la visualización (colores, secciones, indentación, resaltado, imágenes, íconos) para enfatizar los aspectos que el usuario debe tener en cuenta durante el aprendizaje, por lo que no se encuentran orientadas al aprendizaje de programación por parte de personas no videntes.

En este contexto, múltiples trabajos han detallado diferentes estrategias y soluciones aplicables en relación a esta problemática [7]–[14]. Sin embargo, el proceso de enseñanza que da lugar al aprendizaje inicial, no es sencillo de llevar a cabo. Aunque existen muchas herramientas disponibles, la mayoría de ellas apunta a brindar soporte al proceso posterior al aprendizaje inicial (dando por sentado que el usuario -en este caso, estudiante- tiene conocimientos previos en el área). Por lo tanto, todas estas posibles soluciones (o combinaciones de ellas) son aplicables dentro de un contexto controlado, pero la realidad cambia cuando uno de los alumnos del curso tiene dificultades para leer el material proporcionado y/o para resolver los ejercicios

propuestos de la misma forma que sus compañeros. Esta situación se evidenció en 1º año de Ingeniería en Sistemas de Información a inicios del ciclo lectivo 2016, llevando a que los docentes de las diferentes cátedras busquen nuevas estrategias de enseñanza aplicables al caso.

Específicamente, dentro de la cátedra Algoritmos y Estructuras de Datos, la particularidad de enseñar programación en C++ a un alumno no vidente no pareció presentar un obstáculo. Dada la cantidad de herramientas disponibles, parecía factible utilizar una combinación de las mismas a fin de llevar a cabo las clases teórico-prácticas sin afectar el desempeño del alumno. En este punto, es importante destacar que alumno no poseía conocimientos de programación previos al ingreso universitario, por lo que todas las herramientas a utilizar constituían nuevas experiencias dentro del proceso de aprendizaje.

Aunque la utilización de las herramientas disponibles sentó una base sólida para el trabajo cotidiano, muchos aspectos importantes quedaban relegados al entendimiento del alumno. Dentro del aula, tales aspectos podían ser fácilmente solucionados por medio de una explicación del docente, pero el problema se acrecentaba cuando estos aspectos eran evidenciados por fuera de la clase. Además, al profundizar los temas curriculares y plantear el desarrollo de programas más extensos de forma colaborativa, la lectura de códigos desarrollados por otros compañeros constituía para el alumno una tarea altamente difícil y tediosa. Por estos motivos, desde la cátedra se impulsó el diseño (y posterior implementación) de una herramienta de soporte a la tarea de programación que facilite el entendimiento de todos los problemas detectados hasta el momento y, al mismo tiempo, posibilite la incorporación de nuevas características a partir de problemas futuros. Es decir, la herramienta propuesta en este trabajo posibilita al usuario incorporar marcas y sugerencias dentro de un código ya desarrollado; permitiéndole al alumno leer e interpretar desarrollos realizados por otras personas, como así también depurar sus propios códigos. En este trabajo, se presenta esta propuesta a fin de difundirla en la comunidad educativa no sólo para divulgar la herramienta sino también para enriquecer los aspectos trabajados con nuevas perspectivas.

El resto del trabajo se encuentra estructurado de la siguiente manera. La sección 2 presenta el conjunto de herramientas más comunes que utilizan los programadores no videntes. Un alumno con discapacidad visual que quiere aprender a programar puede utilizar múltiples herramientas durante el aprendizaje, por lo que esta sección resume y analiza las herramientas básicas requeridas para tal fin [15]. La sección 3 detalla el contexto de la herramienta propuesta, poniendo énfasis en los problemas detectados (en relación al alumno no vidente) durante el cursado del primer cuatrimestre. La sección 4 conceptualiza el diseño de esta herramienta a fin de sentar las bases de diseño necesarias para solucionar los problemas detectados durante el aprendizaje de forma automática. La sección 5 introduce los resultados obtenidos hasta el momento. Finalmente las secciones 6 y 7 sintetizan, respectivamente, los trabajos futuros y las conclusiones del trabajo desarrollado.

2. Herramientas de Utilidad para Desarrolladores No Videntes

2.1. Sistemas Operativos

En la actualidad, existe una innumerable cantidad de sistemas operativos disponibles. Sin embargo, el nivel de adecuación de estos sistemas para con las personas no videntes es bajo, ya que no existen suficientes usuarios ciegos como para garantizar una mejor accesibilidad de las aplicaciones.

Sólo el 1% de las personas ciegas utiliza *Linux*. Esto se debe a que, entre otras cosas, en este sistema operativo los lectores de pantalla no son tan avanzados y la accesibilidad del escritorio no es tan simple como en otros casos. En este sentido, aunque es un sistema operativo factible de ser utilizado no presenta un nivel de accesibilidad distintivo respecto de *Windows* o *Mac OS X*. Aunque en un principio puede parecer el sistema operativo más apropiado (debido a que es sólo texto), presenta muchas dificultades al tener que trabajar con navegadores de Internet. Esta es una de sus principales desventajas, ya que al cambiar de perspectiva (de escritorio a Internet) dificulta el entendimiento del contenido actual por parte de los usuarios no videntes. Aun así, es importante destacar que *Linux* es una parte integral del conjunto de herramientas que todo desarrollador (vidente o no vidente) debe conocer y manejar.

En este contexto, es posible pensar que *Mac OS X* es uno de los sistemas operativos que brinda mayor accesibilidad a las personas no videntes, ya que su objetivo es proporcionar una interfaz de usuario agradable y simple de usar, manteniendo (de igual forma) la posibilidad de acceder a una terminal. Aunque esto es cierto, el mayor inconveniente que posee este sistema operativo para con los usuarios ciegos es que es cerrado. Mientras que en *Linux* y *Windows* existen múltiples herramientas de accesibilidad disponibles para ser instaladas (lectores de pantalla, sintetizadores de voz, etc.), en *Mac OS X* sólo pueden instalarse/utilizarse herramientas propietarias. De esta manera, la falta de libertad de los usuarios para elegir las herramientas de accesibilidad más convenientes de acuerdo a sus necesidades se transforma en su principal desventaja.

Desde este punto de vista, *Windows* es uno de los sistemas operativos más simples de ser utilizados por personas no videntes ya que flexibiliza algunos de los aspectos mencionados en los casos precedentes. De todas maneras, es importante destacar que cualquiera de los sistemas operativos mencionados puede ser utilizado por usuarios no videntes (todo depende del objetivo que se tenga en mente al momento de instalarlo en el equipo).

2.2. Lectores de Pantalla (Screen Readers)

Un número importante de estudios muestra que cuando se utilizan aplicaciones basadas en audio las personas ciegas pueden desarrollar y ensayar de mejor forma su cognición [16]–[18]. Los lectores de pantalla utilizan esta estrategia de forma tal de reproducir la información visualizada en la pantalla a personas no videntes por medio de una lectura textual del contenido (es decir, haciendo uso de audio).

En términos generales estas herramientas son programas de software que proveen una interfaz entre el sistema operativo, las aplicaciones y el usuario [19]. Para su funcionamiento, el usuario envía comandos por medio de diferentes combinaciones de teclas a fin de indicar al lector que informe los cambios de texto que tienen lugar dentro de la pantalla. Cada uno de los comandos indica diferentes acciones al lector, a saber: deletrear una palabra, leer una línea completa, leer un texto completo, encontrar una cadena de caracteres dentro del texto desplegado en pantalla, localizar el cursor o seleccionar un ítem. Algunos lectores más avanzados poseen una mayor cantidad de funciones, como ser: localizar texto de un color específico, leer partes preseleccionadas de la pantalla, leer únicamente texto resaltado, identificar la opción activa dentro de un menú, entre otras.

Entre las herramientas más populares de este tipo de software se destacan *JAWS* (Job Access With Speech) [20], *NVDA* (NonVisual Desktop Access) [21], *Window-Eyes* [22], *VoiceOver* (solo en Mac e iOS) [23], *Orca* (solo en Linux sobre Gnome) [24], *Speakup Project* (solo en Linux sin ambiente de escritorio) [25], *Google Talkback* (solo en Android) y *ChromeVox* (solo en Chrome) [26].

2.3. Entornos de Desarrollo Integrado (IDEs)

Un entorno de desarrollo integrado (*Integrated Development Environment*, en inglés) es una herramienta informática que proporciona servicios integrales para el desarrollo de software [27]. En términos generales, un IDE incluye un editor de código fuente, herramientas de construcción automática y un depurador. Además, estos entornos suelen incluir un compilador y/o un intérprete.

Para un desarrollador no vidente, la mayoría de las características incluidas en los IDEs actuales son de utilidad. Desde el auto-completado inteligente de código hasta la navegación por bloques, el uso de estas herramientas como parte del proceso de desarrollo facilita la tarea de programación para estos usuarios. Es importante destacar que una de las características más requeridas en estos casos es la posibilidad de utilizar comandos de teclado (por ejemplo, combinaciones de teclas) para acceder a las diferentes operaciones.

3. Dificultades Encontradas en la Enseñanza de Programación a un Alumno No Vidente

A lo largo del cursado del 1º cuatrimestre, el programa de la cátedra *Algoritmos y Estructuras de Datos* propone desarrollar los contenidos detallados en la Tabla 1. Como puede observarse estos contenidos buscan introducir al alumno en las nociones básicas de programación, utilizando primero un pseudo-interprete en lenguaje natural (que posibilita la adaptación del alumno al pensamiento algorítmico), para luego pasar al paradigma de programación estructurada en C++. De esta manera, los alumnos realizan una transición paulatina entre el lenguaje natural y los lenguajes de programación.

Para el desarrollo de estos temas, en las clases de teoría se utilizan ejemplos motivadores como parte del proceso de enseñanza. Así, mientras el docente desarrolla los conceptos teóricos, el alumno puede asimilar tales conceptos de forma clara y ejemplificadora dentro de un caso práctico específico. Por su parte, las clases de práctica se dividen en dos tipos de actividades: actividades en aula y actividades en laboratorio. En el ámbito del aula, las prácticas se llevan a cabo en papel a fin de que el alumno plasme sus ideas de forma individual. De esta manera, se fomenta el desarrollo de estrategias de solución y no la construcción de soluciones por prueba y error. En el ámbito del laboratorio, los alumnos trabajan de forma cooperativa y colaborativa haciendo uso de herramientas de software que les facilitan la codificación de sus soluciones. Además, en este último caso, utilizan *URI Online Judge*[1] como mecanismo de auto-evaluación para controlar un subconjunto de ejercicios previamente definidos por los docentes. Así, el alumno puede probar sus desarrollos a fin de corroborar su correctitud. En ambos casos (tanto en aula como en laboratorio), los alumnos llevan a cabo guías prácticas pensadas para trabajar con un conjunto de contenidos específicos.

3.1. Dificultades y Problemas Detectados (Primer Cuatrimestre)

Durante el desarrollo de las clases teóricas, la utilización del lector JAWS [20] permitió al alumno no

[1] *URI Online Judge*: https://www.urionlinejudge.com.br/judge/en/login

vidente acceder a todo el material bibliográfico (tanto libros como transparencias) proporcionado por la cátedra. La lectura de los códigos de ejemplo se desarrolló combinando el uso de este lector con la visualización del código fuente en el entorno de desarrollo propuesto, lo que facilitó el seguimiento de las estrategias de solución aplicadas. Aunque se dieron algunas dificultades, las mismas pudieron resolverse sin mayores inconvenientes.

Tabla 1. Resumen de los contenidos curriculares de la cátedra *Algoritmos y Estructuras de Datos* desarrollados durante el 1º cuatrimestre de 2016.

Semana	*Presentación*
1	• Presentación de la materia. • Introducción.
2	• Resolución de problemas generales. • Algoritmos. • Expresiones.
3	• PSeInt[2]. • Estructuras de control condicionales.
4	• Estructuras de control anidadas. • Problemas de algoritmia.
5	• Computadoras. • Paradigmas. • Lenguajes de programación.
6	• Proceso de programación. • Lenguaje C++.
7	• Datos de entrada/salida. • Tipos de datos simples. • Estructuras de control condicionales.
8	• Estructuras de control repetitivas C++.
9	• Problemas de secuencias.
10	• Descomposición modular. • Funciones.
11	• Funciones. • Generación de números aleatorios.
12	• Paso de parámetros por copia.
13	• Paso de parámetros por referencia. • Recursividad.
14	• Caracteres. • Cadenas de caracteres.
15	• Arreglos. • Operaciones sobre arreglos.

[2] *PSeInt* es una herramienta que asiste al estudiante en sus primeros pasos en programación. Mediante un simple e intuitivo pseudolenguaje en español (complementado con un editor de diagramas de flujo), permite al alumno centrar su atención en los conceptos fundamentales de la algoritmia computacional, minimizando las dificultades propias de un lenguaje y proporcionando un entorno de trabajo con numerosas ayudas y recursos didácticos [28].

En contraposición, durante las clases prácticas se evidenció una mayor cantidad de inconvenientes para comprender las soluciones de los ejercicios planteados. Tales dificultades se relacionaron con aspectos propios de ubicación, localización y especificación de los códigos desarrollados (no con la elaboración de los algoritmos).

Al iniciar el cursado, la habilidad del alumno no vidente para codificar ejercicios simples en C++ (20 a 30 líneas) era similar a la de sus compañeros de curso. Sin embargo, al avanzar en los contenidos y dirigir la programación hacia ejercicios más complejos y extensos, se comenzó a evidenciar la falta de mecanismos de soporte en los entornos de desarrollo para ubicar al usuario dentro de un archivo fuente y poder ayudarlo a comprender la estructura del programa desarrollado. En las siguientes secciones se describen las dificultades detectadas hasta el momento en relación a esta problemática.

Ubicación en el Código Fuente. La identificación por número de línea es una de las principales formas de ubicación dentro del código fuente de un programa. Este tipo de localización es de utilidad al momento de corregir los errores de sintaxis y depurar el código en busca de otro tipo de errores. Sin embargo, en el caso de los programadores ciegos, sirve además como soporte para comprender el contenido del programa, permitiéndoles analizar las expresiones utilizadas línea a línea y contextualizarlas dentro del programa remanente. Esta característica es de utilidad en códigos reducidos, pero en códigos extensos la simple numeración de las líneas puede que no aporte demasiada información acerca del contenido del programa. Por ejemplo, la expresión *(1)* visualiza el contenido que obtendrá un programador ciego del lector de pantalla al posicionarse con el cursor del IDE en la línea 10 de un código fuente.

```
10 if(cordenadax>0 && cordenaday>0){ (1)
```

Como puede observase, es verdaderamente difícil determinar la correctitud de esta línea sin conocer el contexto dentro del cual se encuentra inserta. Claro está que este problema no se limita a programadores no videntes, ya que para comprender una expresión debe analizarse su contexto. Sin embargo, la lectura del contexto en el caso de personas no videntes es una tarea altamente costosa ya que implica desplazar el cursor hacia las líneas precedentes y consecuentes en espera de que el lector de pantalla las traduzca. En este sentido, proveer (al menos) la información asociada al método, función o estructura dentro del cual se encuentra ubicada la línea de código actualmente seleccionada es información de valor para los usuarios no videntes (dando como consecuencia un problema que debe ser tratado por los docentes).

Delimitación de Bloques. Muchos lenguajes de programación utilizan llaves para delimitar bloques de código. Cuando los estudiantes aprenden a programar, es frecuente que confundan la delimitación de bloques por medio de llaves con otro tipo de estrategias (por ejemplo, uso de sangrías). Aún más, es común que los alumnos de los niveles iniciales no tengan en claro los objetivos de los bloques que definen, por lo que suelen trabajar cerrando llaves en lugares incorrectos (lo que da lugar a bloques de códigos mal definidos).

Para facilitar la especificación de este tipo de elementos, los entornos de desarrollo suelen utilizar colores a fin de indicar el par de llaves que define un bloque de código. En la Figura 1 se visualiza un programa de ejemplo codificado en C++ haciendo uso del IDE ZinjaI[3]. Como puede observarse, el cursor se encuentra posicionado en una de las llaves de finalización de bloque localizada en la línea 23. Este posicionamiento da como consecuencia el resaltado en color rojo del par de llaves que define el bloque seleccionado (esto es, llaves de las líneas 12 y 23). Aunque esta información es útil para la mayoría de los programadores, las personas no videntes no poseen ningún tipo de mecanismo que les permita evidenciar este comportamiento. Entonces, la delimitación de bloques de código se transforma en un problema susceptible de ser trabajado por medio de otro tipo de estrategias durante el aprendizaje.

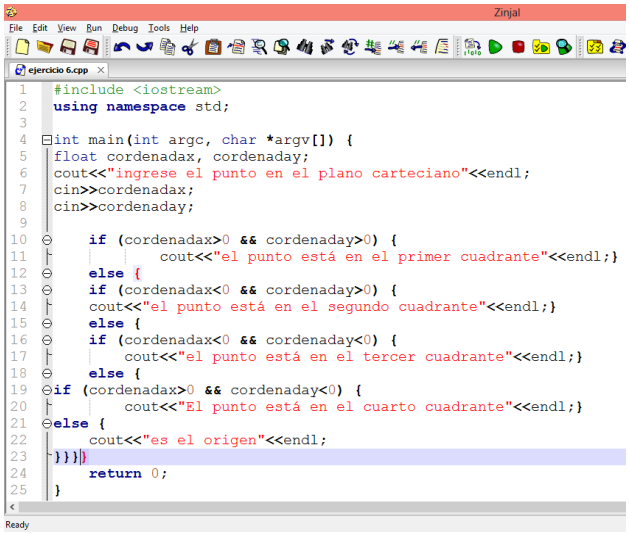

Figura 1. Ejemplo en C++ de mecanismos de delimitación de bloques de código en el IDE Zinjal.

Separación de Líneas. Dentro de un programa C++ todas las sentencias que no se corresponden con la especificación de estructuras de control deben finalizar con punto y coma. En un caso ideal, cada una de las líneas de un archivo de código fuente debería contener una (y solo una) sentencia. Sin embargo, los lenguajes de programación no suelen restringir la estructuración de sus códigos a un formato específico (ya que esto depende del editor utilizado para dar soporte al lenguaje y no del lenguaje en sí mismo). Por este motivo, es normal encontrar códigos C++ que incluyan expresiones similares a las expuestas en *(2)*; donde se visualizan dos sentencias diferentes (una de salida estándar y otra de entrada estándar) dentro de una misma línea (en este caso, la línea 5).

```
5 cout<<"Ingrese un dato"; cin>>x;    (2)
```

Para las personas no videntes (sobre todo para aquellas que recién comienzan a programar), la lectura organizada del código fuente es de mucha utilidad ya que les brinda la posibilidad de analizar el contenido de un programa de forma sectorizada. De esta manera, les permite comprender con mayor nivel de certeza las instrucciones, ya que focalizan su atención en la lectura de una sentencia por vez (es decir, una traducción del lector de pantalla por línea). Además, les facilita la lectura carácter a carácter debido a que reconocen cada punto y coma como la marca de fin de una sentencia (ya que al encontrar un punto y coma, el programador sabe que ha finalizado la lectura y puede pasar a leer la línea siguiente). En este contexto, la individualización de las sentencias en una única línea de texto constituye un problema a trabajar.

Uso de acentos y Distinción entre Letras Mayúsculas y Minúsculas. Es sabido que durante el proceso de codificación el programador no puede utilizar caracteres acentuados. Además, habitualmente los lenguajes de programación estructurados son *case sensitive*[4]. En general, cuando los alumnos de los niveles iniciales comienzan a programar solo utilizan letras minúsculas en sus definiciones (sin incluir acentos en la mayoría de los casos). Esto da como resultado que no evidencien las particularidades del leguaje hasta encontrarse en etapas más avanzadas del aprendizaje.

Sin embargo, en el caso de los programadores no videntes, su propia formación los lleva a programar sus códigos siguiendo tanto las normas de escritura tradicionales como las reglas de ortografía. Esta situación se pone en evidencia al dar los primeros pasos en programación, dando como resultado la aparición de múltiples errores de sintaxis al momento de compilar los archivos de código desarrollados. Ante la ocurrencia de

[3] *ZinjaI*: Disponible en http://zinjai.sourceforge.net/.

[4] *Case sensitive*: Expresión usada en informática que se aplica a los textos en los que tiene alguna relevancia escribir un carácter en mayúsculas o minúsculas.

este tipo de errores, el lector de pantalla no brinda una ayuda simple y clara. Mientras que la lectura palabra por palabra omite la distinción de caracteres especiales, la lectura carácter a carácter (en su forma estándar) no suele estar configurada para realizar esta diferenciación. Luego, la posibilidad de evitar la ocurrencia de este tipo de errores es un problema de interés que puede ser trabajado de diferentes maneras.

3.2. Dificultades y Problemas a Detectar (Segundo Cuatrimestre)

Durante el transcurso del segundo cuatrimestre es probable que se evidencien nuevas dificultades a ser afrontadas (tanto por el alumno no vidente como así también por los docentes asociados a su comisión). Aunque aún no se ha comenzado a trabajar en los temas relacionados con la segunda parte de la materia, es importante conceptualizar una solución automatizada que permita incorporar nuevas soluciones a nuevos problemas a medida que se avanza con el cursado. Entre los temas a trabajar durante este período del ciclo lectivo se destacan tipos de datos abstractos, archivos y estructuras de datos dinámicas.

4. Automatización: Herramienta de Soporte para “hacer explícito lo implícito”

Tomando como base las dificultades detectadas hasta el momento, desde la cátedra se impulsó la idea colaborativa de desarrollar una herramienta de software que posibilite la resolución de estas dificultades de forma automática.

Conceptualmente, la herramienta se encuentra diseñada como software de soporte al programador no vidente a fin de simplificar las tareas de lectura y análisis de código fuente. Para esto, el programador elegirá un código C++ implementado en cualquier entorno de desarrollo y, luego, generará un código equivalente que mantendrá el contenido original pero que, al mismo tiempo, incorporará un conjunto de marcas que agilizarán su interpretación. Cada tipo de marca actuará como mecanismo de solución de una (o más) de las dificultades detectadas. El código resultante de este proceso de marcado no será más que un nuevo código fuente C++ que podrá abrirse y editarse con cualquier entorno de desarrollo. Bajo esta conceptualización, no se pierden las bondades de los IDE ni se imposibilita el uso de herramientas complementarias por parte de los usuarios no videntes (como ser el lector de pantalla).

4.1. Desarrollo basado en Prototipos Evolutivos

El desarrollo basado en prototipos posibilita la construcción de modelos de aplicaciones de software sobre los cuales es posible analizar las funcionalidades básicas requeridas por el cliente (sin necesidad de incluir toda la lógica o todas las características del modelo final). De esta manera, permite al cliente evaluar en forma temprana el producto e interactuar con diseñadores y desarrolladores para determinar si el desarrollo actual cumple con las expectativas [29].

El modelo basado en prototipos pertenece al conjunto de modelos de desarrollo evolutivos, ya que cada prototipo es evaluado por el cliente a fin de lograr una retroalimentación con la que se refinan los requisitos del software y se ajusta el prototipo actual. Esto permite que, al mismo tiempo, el desarrollador entienda que es lo que se debe construir y el cliente tenga resultados a corto plazo. Si sobre el modelo evolutivo se incorpora una estrategia de desarrollo incremental, se da lugar a un modelo de desarrollo evolutivo e incremental. Este tipo de modelos brinda la posibilidad de controlar la complejidad y los riesgos, desarrollando una parte del producto software en una etapa y reservando el resto de los aspectos requeridos para el desarrollo futuro. De esta manera, los prototipos desarrollados son mecanismos de prueba de un conjunto de funcionalidades que condensan la idea principal del sistema en desarrollo, aumentando su funcionalidad con el paso del tiempo.

En el caso de la herramienta propuesta, las ventajas de utilizar un desarrollo basado en prototipos son evidentes. No solo permitirá evaluar la adecuación de las soluciones especificadas para cada una de las dificultades detectadas, sino que además posibilitará la incorporación de nuevas funcionalidades a medida que se avance en el desarrollo. Debido a que existe solo un usuario con la capacidad de evaluar las soluciones propuestas, es altamente provechoso lograr una retroalimentación inmediata de forma tal que se agilice el proceso de desarrollo en base a un proceso de mejora continua del producto de software resultante.

4.2. Estilo Arquitectónico Propuesto: Pipeline

Los sistemas de flujo de datos se caracterizan por la forma en la cual la información se mueve a través del sistema. En general, su arquitectura tiene dos o más componentes de procesamiento que mapean de diferente forma los datos de entrada en datos de salida. La naturaleza de los datos a ser mapeados no queda restringida por el estilo arquitectónico [30].

Si los vínculos entre componentes se dan de forma secuencial, la arquitectura corresponde a un modelo *pipeline*. En estos casos, el flujo de datos de salida de un componente se transforma en el flujo de datos de entrada del siguiente. Pensando que cada componente de procesamiento puede trabajar en la solución de uno de los problemas detectados, el diseño de la herramienta propuesta se corresponde con una arquitectura pipeline.

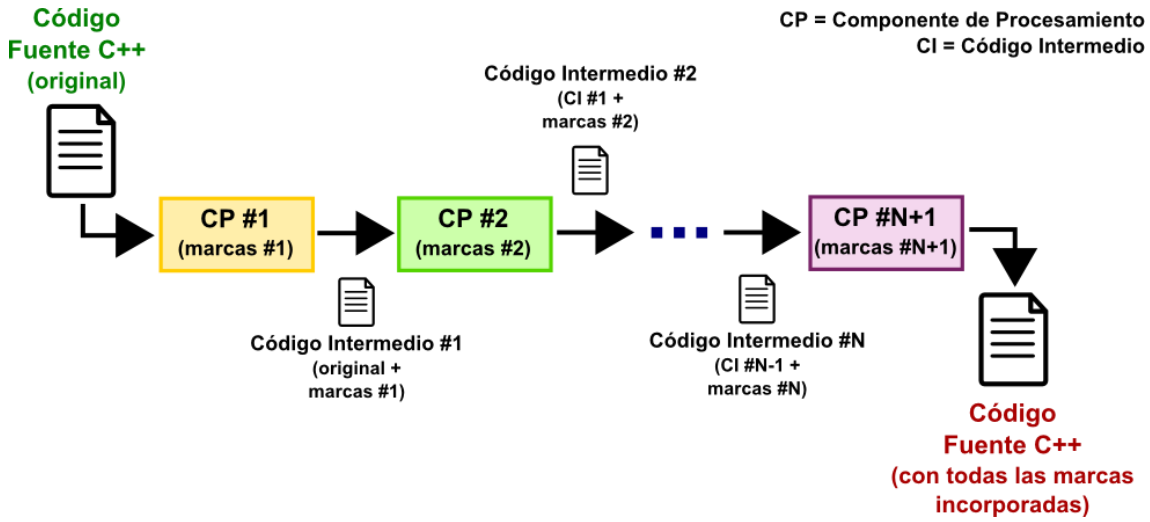

Figura 2. Arquitectura pipeline de la herramienta propuesta.

La Figura 2 esquematiza el diseño arquitectónico propuesto. De acuerdo con este esquema, cada uno de los componentes de procesamiento (CP) actúa sobre una versión del código fuente a fin de modificarla (introduciendo un conjunto de marcas que buscan solucionar uno de los problemas detectados) y retransmitirla como flujo de salida hacia otro componente.

4.2.1. Responsabilidades de los Componentes de Procesamiento. En base a las dificultades y problemas descriptos en la sección 3, se plantearon las siguientes soluciones automatizadas a ser implementadas en los primeros cuatro componentes de procesamiento:

1) *CP #1 - Dar un formato legible al código fuente*: Aunque la legibilidad es una propiedad del código fuente a nivel visual, su aplicación da lugar a una mejor comprensión y entendimiento por parte de cualquier lector (ya sea humano o artificial). En este sentido, la organización de la información que se provee al darle legibilidad al código es de mucha ayuda para los programadores no videntes. Por este motivo, se propone que el primer componente de procesamiento de la herramienta a implementar tenga como objetivo dar formato legible al código fuente. Teniendo en cuenta que existen múltiples herramientas desarrolladas específicamente para estos fines (embellecedores, beautifiers, formateadores, entre otros), se propone realizar un análisis de las herramientas de código abierto existentes a fin de seleccionar una de ellas y utilizarla como primer módulo de la herramienta en desarrollo. En este punto es importante destacar que el formateador de código debe ser el primer módulo de procesamiento de la herramienta ya que la información que produce como resultado de su ejecución simplifica el trabajo a realizar en los siguientes componentes (es decir, ayuda a examinar de forma más eficiente el código bajo análisis).

2) *CP #2 - Incorporación de comentarios descriptivos para especificar el cierre de bloques*: A fin de delimitar explícitamente los bloques de código, se propone la inserción de comentarios descriptivos luego de la llave de cierre. Específicamente, el comentario a insertar deberá indicar el contexto en el cual está actuando el contenido del bloque. Esta responsabilidad estará asociada al segundo componente de procesamiento a incorporar en la herramienta.

3) *CP #3 - Modificación de caracteres alfabéticos con acentos y letras mayúsculas*: A fin de solucionar los problemas asociados con los caracteres acentuados y letras mayúsculas, se propone modificar el código entrante reemplazando estos caracteres por los caracteres en minúsculas no acentuados equivalentes. Esta transformación será incorporada a la herramienta propuesta en el tercer componente de procesamiento. Como resultado de su ejecución se obtendrá un código C++ equivalente al ingresado cuyos caracteres alfabéticos serán únicamente letras minúsculas.

4) *CP #4 - Descriptor de la cantidad de líneas incluidas dentro de una función*: Con el objetivo de indicar el tamaño de las diferentes funciones, se propone incorporar un comentario (luego de la especificación del encabezado) que indique no sólo la cantidad de líneas involucradas en la función sino

también el número de línea de inicio y finalización de la misma. De esta manera, el lector puede (sin necesidad de conocer el desarrollo de la función) dimensionar su extensión. La implementación de esta responsabilidad quedará asociada al cuarto componente de procesamiento de la herramienta.

4.2.2. Componentes de Procesamiento No Especificados. Las responsabilidades de los componentes de procesamiento no especificados hasta el momento deberán ser detalladas a medida que se detecten nuevas dificultades y/o problemas. Cada una de estas especificaciones dará lugar a la implementación de un nuevo módulo de la herramienta.

5. Resultados Preliminares

Tal como se ha mencionado con anterioridad, la herramienta propuesta en este trabajo se encuentra actualmente en proceso de desarrollo. Un becario de grado ha sido asignado a esta tarea, trabajado en estrecha relación con los docentes y auxiliares de la cátedra. El lenguaje de programación Java fue elegido para la implementación debido a las ventajas que presenta en lo referente a portabilidad y accesibilidad.

Teniendo en cuenta la finalidad descripta para el primer componente de procesamiento, se realizó un estudio de los embellecedores existentes. Se decidió trabajar con Uncrustify debido a que (además de ser una herramienta de código abierto) es lo suficientemente completa como para abarcar todos los posibles problemas de formato asociados a un código C++. Esta herramienta se ejecuta por consola y modifica el archivo fuente original, por lo que provee además las interfaces necesarias para trabajar en conjunto con el resto de los componentes propuestos.

De forma adicional se implementó una primera aproximación del componente de procesamiento asociado a la delimitación de bloques por medio del uso de comentarios (CP #2). La Figura 3 muestra como ejemplo un código desarrollado por el alumno no vidente haciendo uso de ZinjaI, mientras que la Figura 4 visualiza el mismo código luego de haberse realizado la ejecución del componente implementado. Como puede observarse, el código resultante incorpora un conjunto de comentarios descriptivos que (al ser traducidos por el lector de pantalla) permiten al programador no vidente interpretar su significado. Aunque el código obtenido no incorpora indentaciones, es importante destacar que la finalidad del mismo es facilitar la lectura a personas no videntes. En este contexto, el uso de indentaciones no es relevante ya que los lectores de pantalla sólo tienen la capacidad de leer contenido (es decir, no leen espacios en blanco y/o tabulaciones en el modo de lectura tradicional). También es importante destacar que la cantidad de líneas de código se ha incrementado (de 25 a 30) como resultado del proceso de transformación. Esto no es un problema ya que la interpretación del código final es mucho más sencilla que la del código original.

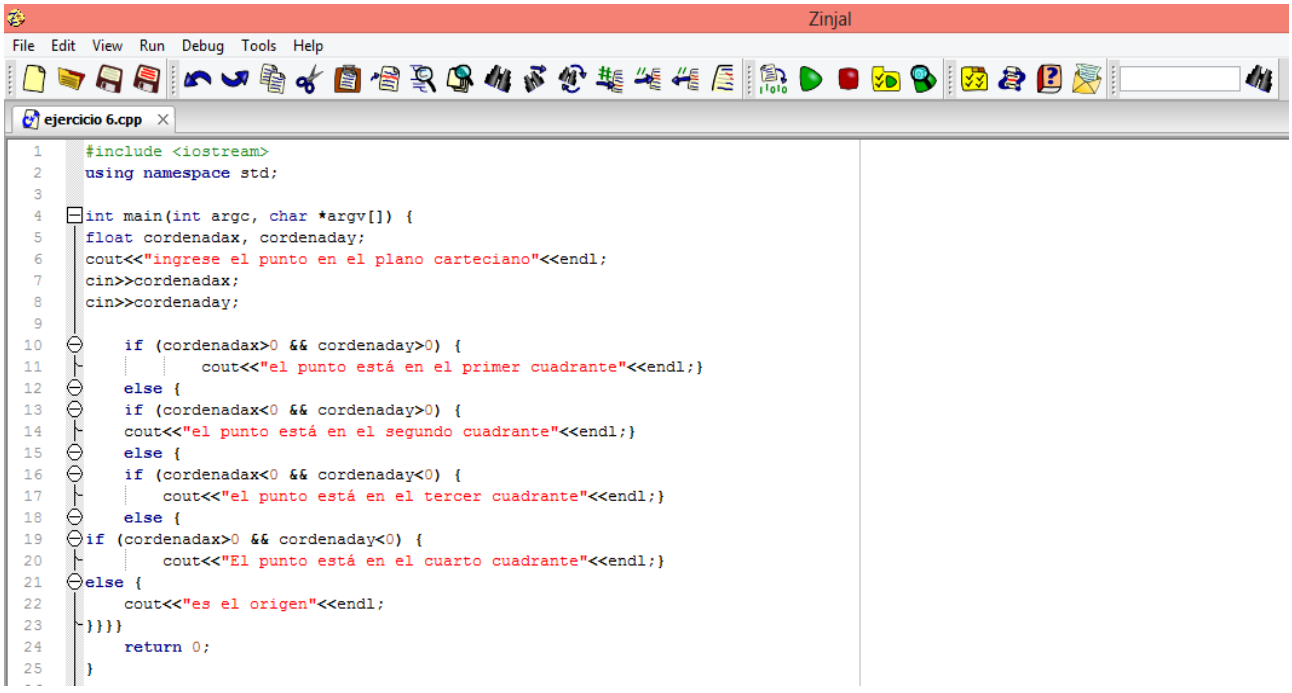

Figura 3. Ejemplo de código original.

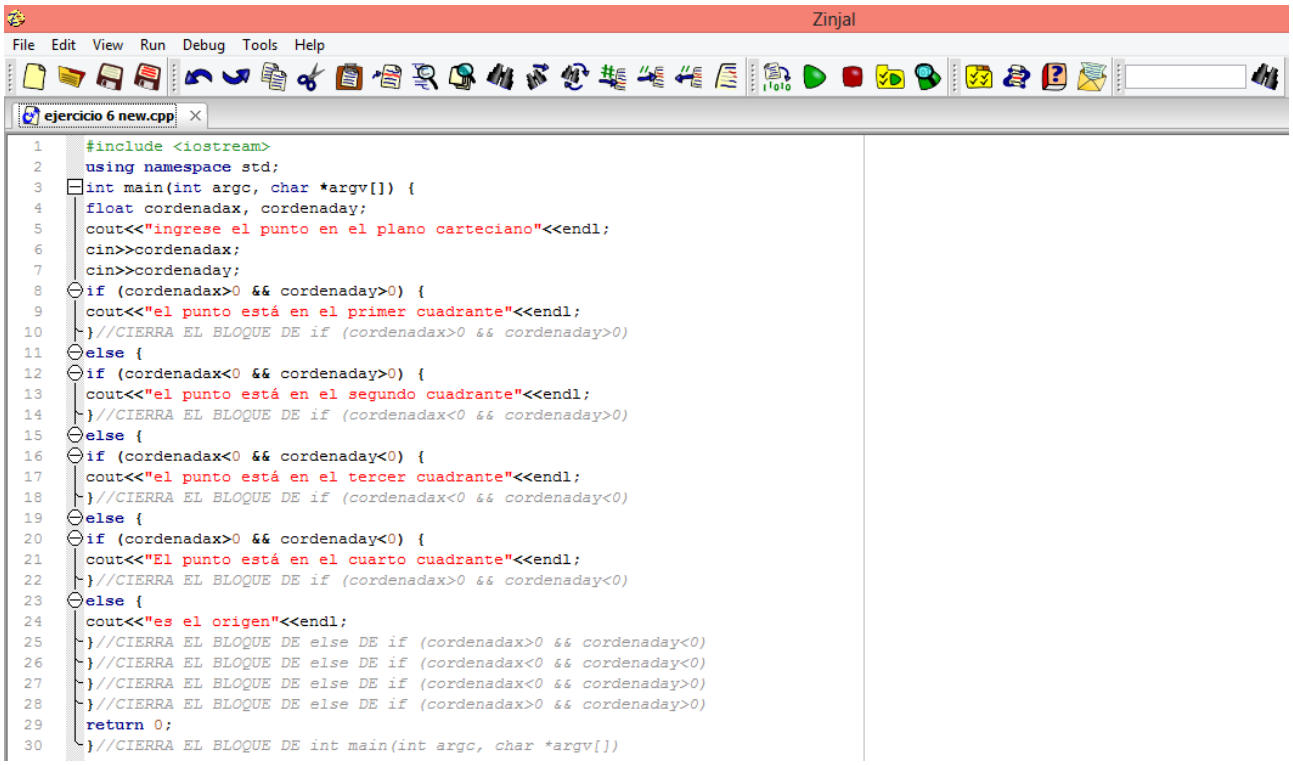

Figura 4. Código resultante luego de ejecutar el CP #2 sobre el código original.

6. Trabajos Futuros

Tomando como base la estrategia propuesta, a futuro se espera completar el diseño y desarrollo de la herramienta incorporando nuevos módulos de acuerdo a las dificultades evidenciadas durante el transcurso del segundo cuatrimestre. Tal incorporación se verá favorecida por la arquitectura de software propuesta, ya que este tipo de diseño arquitectónico facilita la adición de nuevos componentes de procesamiento en interacción con los previamente definidos por medio del intercambio de información.

La construcción por medio de prototipos evolutivos permitirá realizar una prueba en las diferentes etapas de desarrollo, lo que posibilitará mejorar cada uno de los módulos implementados de forma independiente. En consecuencia, al finalizar la construcción de la herramienta el proceso de pruebas estará concluido.

Específicamente dentro de la facultad, la herramienta quedará a disposición del alumno no vidente para que la utilice a lo largo del aprendizaje. Además, podrá ser mejorada por las cátedras que así lo requieran a medida que avance con el cursado de la carrera. Se dejará disponible para su descarga una versión empaquetada que podrá ser utilizada por cualquier usuario que así lo desee.

Como extensión de este trabajo, existe la posibilidad de desarrollar un complemento para el IDE ZinjaI de forma tal que las marcas y sugerencias se incorporen al código fuente a medida que el programador no vidente se encuentre codificando.

7. Conclusiones

En este trabajo se ha presentado una herramienta de soporte a la enseñanza de programación pensada para ser utilizada en el caso de alumnos no videntes. Aunque los principales aspectos de la herramienta se encuentran aún bajo desarrollo, los resultados preliminares han demostrado ser de utilidad para el programador.

Tanto la incorporación de nuevas características como la mejora de las existentes, forman parte del proceso de continuo de adaptación de la herramienta propuesta, no restringiendo su ámbito de aplicación a la cátedra Algoritmos y Estructuras de Datos. Tal como se ha mencionado con anterioridad, la arquitectura definida posibilita la incorporación de nuevas características.

En este contexto, es importante destacar que la accesibilidad es un tema que aún debe ser atendido y explorado a nivel universitario. A diferencia de la educación primaria y secundaria, donde se tienen asistentes especiales y talleres extracurriculares de apoyo, a nivel universitario existe una ausencia de mecanismos de contención que ayuden tanto al docente como al alumno a lograr un verdadero aprendizaje.

8. Referencias

[1] A. Cypher and D. C. Halbert, *Watch what I Do: Programming by Demonstration*. MIT Press, 1993.

[2] R. G. McDaniel and B. A. Myers, "Getting More out of Programming-by-demonstration" in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, New York, NY, USA, 1999, pp. 442–449.

[3] M. B. Rosson and C. D. Seals, "Teachers As Simulation Programmers: Minimalist Learning and Reuse" in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, New York, NY, USA, 2001, pp. 237–244.

[4] H. Lieberman, *Your Wish is My Command: Programming by Example*. Morgan Kaufmann, 2001.

[5] M. Travers, "Recursive Interfaces for Reactive Objects" in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, New York, NY, USA, 1994, pp. 379–385.

[6] J. Montemayor, "Physical Programming: Software You Can Touch" in *CHI '01 Extended Abstracts on Human Factors in Computing Systems*, New York, NY, USA, 2001, pp. 81–82.

[7] J. Sánchez and F. Aguayo, "APL: Un Lenguaje de Programación basado en Audio para Aprendices Ciegos" *IE Comun. Rev. Iberoam. Informática Educ.*, no. 1, p. 3–, 2005.

[8] I. Kopecek and A. Jergova, "Programming and visually impaired people" presented at the IFIP world computer congress, 1998, pp. 365–372.

[9] C. Frauenberger and M. Noistering, "3D audio interfaces for the blind" in *Proceedings of the 2003 International Conference on Auditory Display*, Boston, MA, USA, July 6-9, 2003.

[10] A. C. Smith, J. M. Francioni, and S. D. Matzek, "A Java Programming Tool for Students with Visual Disabilities" in *Proceedings of the Fourth International ACM Conference on Assistive Technologies*, New York, NY, USA, 2000, pp. 142–148.

[11] R. M. Siegfried, "Visual Programming and the Blind: The Challenge and the Opportunity" in *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education*, New York, NY, USA, 2006, pp. 275–278.

[12] Kirchner, C., & Schmeidler, E. (2001). Adding Audio Description: Does it make a difference?. *Journal of Visual Impairment & Blindness (JVIB)*, 95.

[13] A. D. N. Edwards, "Soundtrack: An Auditory Interface for Blind Users" *Hum-Comput Interact*, vol. 4, no. 1, pp. 45–66, Mar. 1989.

[14] J. Sánchez and F. Aguayo, "Blind Learners Programming Through Audio" in *CHI '05 Extended Abstracts on Human Factors in Computing Systems*, New York, NY, USA, 2005, pp. 1769–1772.

[15] P. Doustdar, "The Tools of a Blind Programmer" *Parham Doustdar's Blog*. [Online]. Available: https://www.parhamdoustdar.com/2016/04/03/tools-of-blind-programmer/. [Accessed: 21-Jul-2016].

[16] S. W. Mereu and R. Kazman, "Audio Enhanced 3D Interfaces for Visually Impaired Users" in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, New York, NY, USA, 1996, pp. 72–78.

[17] J. Sánchez, M. Lumbreras, and L. Cernuzzi, "Interactive Virtual Acoustic Environments for Blind Children: Computing, Usability, and Cognition" in *CHI '01 Extended Abstracts on Human Factors in Computing Systems*, New York, NY, USA, 2001, pp. 65–66.

[18] J. Sánchez, N. Baloian, T. Hassler, and U. Hoppe, "AudioBattleship: Blind Learners Collaboration Through Sound" in *CHI '03 Extended Abstracts on Human Factors in Computing Systems*, New York, NY, USA, 2003, pp. 798–799.

[19] American Foundation for the Blind. [Online]. Available: http://www.afb.org/default.aspx. [Accessed: 21-Jul-2016].

[20] JAWS Screen Reader - Best in Class. [Online]. Available: http://www.freedomscientific.com/Products/Blindness/JAWS. [Accessed: 21-Jul-2016].

[21] NV Access. [Online]. Available: http://www.nvaccess.org/. [Accessed: 21-Jul-2016].

[22] GW Micro - Window-Eyes. [Online]. Available: http://www.gwmicro.com/window-eyes/. [Accessed: 21-Jul-2016].

[23] Accessibility - OS X - VoiceOver - Apple. [Online]. Available: http://www.apple.com/accessibility/osx/voiceover/. [Accessed: 21-Jul-2016].

[24] Lector de pantalla Orca. [Online]. Available: https://help.gnome.org/users/orca/stable/. [Accessed: 21-Jul-2016].

[25] The Speakup Project. [Online]. Available: http://www.linux-speakup.org/. [Accessed: 21-Jul-2016].

[26] "ChromeVox." [Online]. Available: http://www.chromevox.com/. [Accessed: 21-Jul-2016].

[27] I. R. Salavert and M. D. L. Pérez, *Ingeniería del software y bases de datos: tendencias actuales*. Univ de Castilla La Mancha, 2000.

[28] PSeInt. [Online]. Available: http://pseint.sourceforge.net/. [Accessed: 25-Jul-2016].

[29] R. Pressman, *Software Engineering: A Practitioner´s Approach*, 7th ed. McGraw-Hill, 2010.

[30] S. Albin, *The art of software architecture: design methods and techniques.* John Wiley & Sons, 2003.