Brunifier: An extendable multi-language beautifier designed for visually impaired people

Emiliano Gioria
Facultad Regional Santa Fe
Universidad Tecnológica Nacional
Santa Fe, Argentina
emigioria@hotmail.com

Diego García Lozano
Facultad Regional Santa Fe
Universidad Tecnológica Nacional
Santa Fe, Argentina
diegogarcialozano95@gmail.com

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

Marta Castellaro
Facultad Regional Santa Fe
Universidad Tecnológica Nacional
Santa Fe, Argentina
mcastell@frsf.utn.edu.ar

***Abstract* — At the beginning of the year 2016, as part of the UTN FRSF academic community, we received an exciting but challenging piece of news: we were going to have a blind student starting that year. Therefore, every single chair needed to adapt its tools and way of teaching accordingly. For example, while he was taking courses of the programming area, we realized that while the most popular IDEs give a lot of useful information using visual resources such as colors, highlighting and font weights, they don't provide any help for the blind user. Moreover, the currently available screen readers are not capable of verbalizing the visually represented metadata provided by the IDEs. That's when, from the chairs of *Algorithms and Data Structures* and *Programming Paradigms*, we decided to tackle that difficulty the student was experiencing. To accomplish that, we developed *Brunifier*, an extendable multi-language code beautifier designed for visually impaired people. The application will process the files of a determined programming language and will add many inline comments to the source code. Those comments provide useful information about the structure of the code and can be read by any screen reader, enhancing the experience of the blind user while coding. What is more, the application architecture is designed in such a way that it allows to easily incorporate new supported programming languages and different comments to the processed files**

Keywords — Blind programmer, software tool, accessibility.

I. INTRODUCCIÓN

En el ámbito universitario, la accesibilidad es hoy en día un tema de atención e investigación en cuanto a la inclusión. La existencia de herramientas de software que contribuyen al aprendizaje de personas no videntes ha crecido, pero su aplicación y grado de apoyo varían según las disciplinas y habilidades requeridas.

En la enseñanza de ciencias de la computación, el desarrollo de habilidades de programación es un tema prioritario. Las *Herramientas de Desarrollo* (HD) 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, ya que cada interesado tiene la posibilidad de elegir el conjunto de HD que mejor se adapte a sus necesidades sin incurrir en un gran esfuerzo y/o costo. Sin embargo, la mayoría de las HD utiliza interfaces de usuario centradas en la visualización (colores, secciones, indentación, resaltado, imágenes, íconos) para enfatizar los aspectos relevantes. Luego, estas herramientas no se encuentran orientadas a personas no videntes. Aunque la mayoría de las herramientas disponibles contribuyen a la labor cotidiana, la falta de estos aspectos dificulta el proceso de aprendizaje.

En este trabajo se presenta una herramienta de soporte a la enseñanza y al aprendizaje de programación en personas ciegas, que busca solucionar los principales inconvenientes detectados en las HD existentes. La motivación en el desarrollo de esta herramienta surgió en el año 2016 con el ingreso de un estudiante no vidente a la carrera Ingeniería en Sistemas de Información de la Facultad Regional Santa Fe - Universidad Tecnológica Nacional (Argentina).

El resto del trabajo se encuentra estructurado de la siguiente manera. La sección II presenta la justificación pedagógica de la solución, poniendo énfasis en los problemas detectados en relación a las herramientas disponibles para el aprendizaje de programación de personas no videntes. La sección III detalla el diseño de la herramienta, sentando las bases requeridas para la implementación de las soluciones a los problemas detectados. La sección IV presenta los resultados obtenidos hasta el momento, presentando un caso de estudio en C++. Finalmente, la sección V se encuentra dedicada a las conclusiones.

II. JUSTIFICACIÓN PEDAGÓGICA

La tecnología cumple un papel muy importante en lo que a accesibilidad respecta ya que es un medio que permite disminuir las barreras existentes ya sean edilicias, comunicacionales, cognitivas o sociales, entre otras. Todos aquellos dispositivos y programas, hardware y software, específicamente diseñados para hacer accesible a los ciegos la tecnología de la información se denominan "tiflotecnología". El término tiflotecnología, del griego tiflo (ciego), se incorpora al Diccionario de la Real Academia de la Lengua Española en 2008, donde se define como el "estudio de la adaptación de

procedimientos y técnicas para su utilización por los ciegos". Más precisamente, se refiere al conjunto de técnicas, conocimientos y recursos encaminados a procurar a las personas ciegas y deficientes visuales los medios oportunos para la correcta utilización de la tecnología, con el fin de favorecer su autonomía personal y plena integración social, laboral y educativa [1]. Es por tanto, una tecnología de apoyo.

Debido a su discapacidad, las personas ciegas no podrían hacer uso de las nuevas tecnologías sin una adaptación adecuada. Es por esto que la tiflotecnología se ha convertido en una herramienta indispensable para las personas no videntes ya que les permite acceder a las nuevas tecnologías, ya sea mediante equipos específicos o adaptaciones, de acuerdo con las necesidades u objetivos de cada usuario. En [2] los autores establecen una distinción entre tiflotecnología específica (creada para uso exclusivo de personas ciegas) y adaptada (diseñada para que un ciego o deficiente visual pueda utilizar un equipo estándar).

Específicamente para el aprendizaje de programación, los estudiantes no videntes se valen de dos tipos de herramientas, a saber: *lectores de pantalla* y *entornos de desarrollo* (IDE, por sus siglas en inglés). Los lectores de pantalla son categorizados como tiflotecnología adaptada, ya que refieren a aplicaciones que permiten identificar e interpretar aquello que se muestra en pantalla. Esta interpretación se presenta al usuario mediante sintetizadores de texto a voz, iconos sonoros, o una salida braille. De esta manera, este tipo de herramientas provee una interfaz entre el sistema operativo, las aplicaciones y el usuario. Entre los lectores de pantalla más populares se destacan JAWS (Job Access WithSpeech) [3], NVDA (NonVisual Desktop Access) [4], Window-Eyes [5], VoiceOver [6], Orca [7] y ChromeVox [8].

Por su parte, los IDE son herramientas informáticas que proporcionan servicios integrales para el desarrollo de software [9]. Generalmente incluyen un editor de código fuente, herramientas de construcción automática y un depurador. Además, suelen incluir un compilador y/o intérprete.

La mayoría de las características incluidas en los IDE son de utilidad para los desarrolladores. Sin embargo, en [10]-[17] se detallan diferentes estrategias y soluciones aplicables a la problemática que presenta la mayoría de las herramientas de desarrollo con respecto a la utilización de interfaces de usuario centradas en lo visual. Tales problemas se evidencian cuando los usuarios de los IDE son personas ciegas que deben combinar su funcionamiento con el de los lectores de pantalla.

En estos casos, no sólo la interoperabilidad de las aplicaciones es fundamental, sino que también lo es la posibilidad de que todo el contenido representado (en cualquier formato) dentro del IDE pueda ser llevado a comandos de voz haciendo uso del lector de pantalla. Es decir, para todas las representaciones visuales (no textuales) debe existir una traducción a texto que posibilite, al usuario ciego, la comprensión del código de la misma forma que la representación visual ayuda al usuario sin discapacidad visual.

La herramienta propuesta en este trabajo brinda una solución a este tipo de problemas sobre códigos desarrollados en los lenguajes GNU Smalltalk, C++ y Scheme.

III. Diseño de la Herramienta

La herramienta desarrollada se encuentra diseñada como un aplicativo de software que brinda soporte a los programadores ciegos a fin de simplificar las tareas de lectura y análisis de código fuente.

Para esto, la herramienta procesa un conjunto de códigos implementados en algún lenguaje de programación válido, con el objetivo de devolver los mismos códigos pero con agregados textuales (denominados “marcas”) que refieren a las representaciones visuales que normalmente incluyen los IDE. Además, se incorporan sonidos que sirven como guía durante el proceso de transformación. Tanto la carga de los archivos como su procesamiento pueden manipularse desde teclado, lo que facilita su uso por parte de personas ciegas.

A fin de garantizar su funcionamiento en múltiples plataformas, se decidió implementar la aplicación haciendo uso del lenguaje de programación Java. Además, este lenguaje es compatible con la mayoría de los lectores de pantalla por lo que se asegura que los usuarios no videntes puedan manipular correctamente su contenido.

Es importante destacar que el aplicativo se encuentra desarrollado en español.

1) Metodología de desarrollo

El desarrollo basado en prototipos posibilita la construcción de modelos de aplicaciones de software sobre los cuales es posible analizar las funcionalidades básicas requeridas sin necesidad de incluir toda la lógica y/o características del modelo final. Así, 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 [18].

Esta metodología de desarrollo en combinación con metodologías ágiles tiene múltiples beneficios ya que permite evaluar los prototipos a fin de lograr una retroalimentación de utilidad para refinar los requerimientos y, en consecuencia, ajustar el prototipo del software actual. Al mismo tiempo, posibilita que el desarrollador entienda la naturaleza del producto de software bajo desarrollo, generando resultados a corto plazo. Si estos resultados se manejan de forma evolutiva e incremental, se logra controlar la complejidad y los riesgos, desarrollando una parte del producto de software en una etapa y reservando el resto de los aspectos requeridos para el desarrollo futuro.

Al momento de desarrollar la herramienta, las ventajas de utilizar estas metodologías fueron evidentes. No solo permitieron evaluar la adecuación de las soluciones propuestas para cada uno de los problemas identificados en relación a los entornos de desarrollo, sino que además posibilitaron la incorporación de nuevas funcionalidades a medida que avanzó en el desarrollo.

2) Estilo arquitectónico: Pipeline

Los sistemas de flujo de datos se caracterizan por la forma en la cual se mueve la información a través de sus componentes. 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 estos datos no queda restringida por el estilo arquitectónico [19]. Si los vínculos entre componentes se dan de forma secuencial, la arquitectura corresponde a un modelo pipeline. En este modelo, el flujo de datos de salida de un componente se transforma en el flujo de datos de entrada del siguiente.

El diseño de la herramienta desarrollada se planteó en base a una arquitectura pipeline (Fig. 1). En este sentido, se planificó un Componente de Procesamiento (CP) por cada dificultad identificada y, en cada uno de los componentes de procesamiento indicados, se implementó la automatización de la solución propuesta para la dificultad asociada. De acuerdo con este esquema, cada componente de procesamiento actúa sobre una versión del código fuente a fin de modificarla y retransmitirla (como flujo de salida) hacia otro componente. El flujo de datos inicial es el código original. El flujo de datos final es el código con la representación textual (marcas) de todos los elementos visuales.

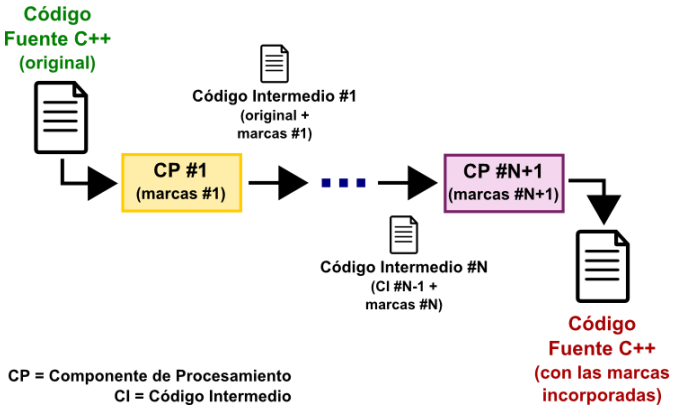


Fig. 1. Diseño de la arquitectura de los componentes de la herramienta.

Este esquema arquitectónico permite, además de una correcta modularización de las soluciones para cada problema identificado, la posibilidad de incorporar incrementalmente nuevo contenido sin requerir modificaciones en el desarrollo previo. Incluso pueden agregarse nuevos lenguajes de programación a fin de incorporar marcas específicas de su forma de codificación como parte del proceso de marcado. Esto implica que la arquitectura elegida facilita la adición de nuevas marcas por medio de la incorporación de nuevos componentes de procesamiento en interacción con los ya implementados (en base al intercambio de información del flujo de salida-entrada).

IV. Resultados

Considerando que en la actualidad el aplicativo trabaja sobre los lenguajes de programación GNU Smalltalk, C++ y Scheme y, teniendo en cuenta que las marcas incorporadas para cada lenguaje son propias de su estilo de codificación, en esta sección se presenta, a modo de ejemplo, la transformación de un programa codificado en C++.

Los componentes de procesamiento incluidos para el tratamiento de este lenguaje son tres, a saber:

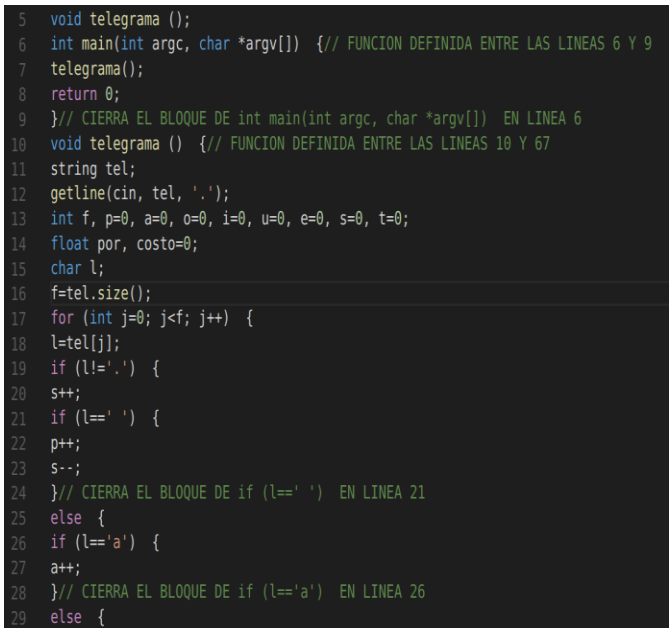
- *Componente de procesamiento #1 – "Formato legible del código fuente"*: Aunque la legibilidad es una propiedad del código fuente a nivel visual, la organización de la información resultante es de mucha ayuda para los programadores ciegos. Por este motivo, el primer componente de procesamiento da un formato comprensible a un código fuente cualquiera a fin de garantizar la correcta legibilidad de su contenido. Es importante destacar que el formateador de código debe ser el primer módulo de la herramienta ya que la información resultante posee un patrón de formato, simplificando el trabajo de procesamiento a realizar en los componentes restantes.
- *Componente de procesamiento #2 – "Comentarios descriptivos en el cierre de bloques"*: A fin de delimitar explícitamente los bloques de código, este componente de procesamiento inserta comentarios descriptivos a continuación de la llave de cierre asociada a un bloque previamente definido. Específicamente, el comentario a insertar indica el contexto en el cual está actuando el contenido del bloque definido a fin de garantizar una correcta interpretación del conjunto de sentencias delimitadas.
- *Componente de procesamiento #3 – "Descriptor de la cantidad de líneas incluidas en una función o procedimiento"*: Con el objetivo de especificar el alcance de las funciones y procedimientos incluidos dentro del programa, se incorpora como parte del encabezado de cada función un comentario descriptivo que indica la cantidad de líneas involucradas junto con los números de línea asociados al inicio y fin de la misma. De esta manera, el usuario puede (sin necesidad de conocer el desarrollo de la función) dimensionar su extensión.

La Fig. 2 presenta un código C++ en una etapa previa al procesamiento en la herramienta, mientras que la Fig. 3 visualiza el código obtenido luego de haberse procesado en el aplicativo. Como puede observarse, el segundo código es más extenso que el primero pero, aun así, para los usuarios ciegos es mucho más simple de interpretar que el código original.

Si el lector desea corroborar esto, solamente imagine que existe un interlocutor al cual se le ha asignado la tarea de que lea en voz alta el contenido de la Fig. 2 (es decir, se transmite a una persona física la responsabilidad del lector de pantalla). Ahora piense que, a partir de esta lectura en voz alta, se debe interpretar la correctitud del código implementado. Sin las marcas incorporadas por la herramienta, es muy difícil dimensionar las sentencias asociadas a cada acción (aun cuando se conoce el código). Sin embargo, si un interlocutor lee textualmente el contenido de la Fig. 3, es mucho más probable que quien lo escuche logre una mejor interpretación del contenido del programa. Esto se debe a que las marcas incorporadas a modo de comentarios ayudan a contextualizar no sólo la lógica del programa sino también los elementos utilizados para describirla. Esto favorece la lectura e interpretación del contenido.

```
void telegrama ();
    int main(int argc, char *argv[]) {
    telegrama();
    return 0;
}
void telegrama () {
    string tel;
    getline(cin, tel, '.');
    int f, p=0, a=0, o=0, i=0, u=0, e=0, s=0, t=0;
    float por, costo=0;
    char l;
    f=tel.size();
    for (int j=0; j<f; j++) {
        l=tel[j];
    if (l!='.') {
        s++;
        if (l==' ') {
            p++;
        s--;
        }
    else {
        if (l=='a') {
            a++;
            }
        else {
```

Fig. 2. Código C++ original.



```
void telegrama ();
int main(int argc, char *argv[])  {// FUNCION DEFINIDA ENTRE LAS LINEAS 6 Y 9
telegrama();
return 0;
}// CIERRA EL BLOQUE DE int main(int argc, char *argv[])  EN LINEA 6
void telegrama ()  {// FUNCION DEFINIDA ENTRE LAS LINEAS 10 Y 67
string tel;
getline(cin, tel, '.');
int f, p=0, a=0, o=0, i=0, u=0, e=0, s=0, t=0;
float por, costo=0;
char l;
f=tel.size();
for (int j=0; j<f; j++)  {
l=tel[j];
if (l!='.')  {
s++;
if (l==' ')  {
p++;
s--;
}// CIERRA EL BLOQUE DE if (l==' ')  EN LINEA 21
else  {
if (l=='a')  {
a++;
}// CIERRA EL BLOQUE DE if (l=='a')  EN LINEA 26
else  {
```

Fig. 3. Código C++ con marcas.

V. Conclusiones

Se ha presentado una herramienta de soporte a la enseñanza de programación pensada para ser utilizada por personas ciegas que desean aprender a programar. Esta herramienta se encuentra diseñada como un aplicativo de software complementario para el programador ciego que simplifica las tareas de lectura y análisis de código fuente, las cuales normalmente son representadas por medio de interfaces visuales en los entornos de desarrollo.

El esquema arquitectónico diseñado junto con la metodología de desarrollo elegida contribuyen a la incorporación de nuevos tipos de marcas y lenguajes de programación, lo que abre un abanico de posibilidades de extensión para el aplicativo implementado.

Es importante destacar que el aplicativo es de uso libre y gratuito y que se continúan mejorando los módulos incluidos a fin de perfeccionar su funcionamiento.

La herramienta ha sido utilizada en dos cursos de programación ("Algoritmos y Estructuras de Datos" y "Paradigmas de Programación"), permitiendo al alumno ciego tanto el leer y escribir programas en forma individual, como así también participar de las actividades prácticas de tipo taller en grupos (realizando proyectos de programación junto con alumnos sin dificultades de visión como trabajos integradores).

Referencias

[1] Pegalajar Palomino, M.C. Tiflotecnología e Inclusión Educativa: evaluación de sus posibilidades didácticas para el alumnado con discapacidad visual. Revista Electrónica de Investigación y Docencia (REID), 9, Enero, 2013, 08-22. ISSN: 1989-2446. Disponible en http://www.revistareid.net/revista/n9/REID9art1.pdf

[2] Cabero, J., Córdoba, M. y Fernández, J.M. Las TICs para la igualdad. Nuevas tecnologías y atención a la diversidad. Sevilla: Eduforma.

[3] Jaws Screen Reader - Best in Class. Disponible en http://www.freedomscientific.com/Products/Blindness/JAWS

[4] NV Access. Disponible en https://www.nvaccess.org/

[5] GW Micro - Window-Eyes. Disponible en http://www.gwmicro.com/window-eyes

[6] Accessibility - OS X – Voice Over - Apple. Disponible en http://www.apple.com/accessibility/osx/voiceover/

[7] Orca. Disponible en https://help.gnome.org/users/orca/stable/

[8] Chrome Vox. Disponible en http://www.chromevox.com/

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

[10] 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. 31–38, 2005.

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

[12] 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.

[13] 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.

[14] 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.

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

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

[17] 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.

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

S. T. Albin. The art of software architecture: design methods and techniques. John Wiley & Sons. 2003.