UNIVERSIDAD CATÓLICA BOLIVIANA "SAN PABLO"

UNIDAD ACADÉMICA REGIONAL COCHABAMBA

Departamento de Ciencias Exactas e Ingeniería

Carrera de Ingeniería de Sistemas

Prototipo de herramienta para el apoyo al proceso de enseñanza aprendizaje de programación a personas con discapacidad o perdida de miembros superiores.

Perfil de Proyecto de Grado de Licenciatura en Ingeniería de Sistemas

César Iván Delgado Silva

Cochabamba – Bolivia

Mayo de 2018

ÍNDICE GENERAL

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1. INTRODUCCIÓN

En una época donde el uso de la tecnología y la computadora ha llegado a ser una necesidad; detectar y resolver problemas usando el computador presenta una oportunidad real para mejorar diferentes actividades del diario vivir. Por eso es importante introducir a los jóvenes en el mundo de la programación desde edades tempranas y así poder aumentar sus posibilidades laborales. A pesar de que existe una gran cantidad de puestos laborales en esta área, las personas con discapacidad o pérdida de miembros superiores no tienen un fácil acceso al mundo de la programación, ya sea por las herramientas a emplear o la desmotivación que implica el uso de *hardware* (teclado y mouse) para poder programar.

Por otro lado, las herramientas de programación actuales, para personas con discapacidad en miembros superiores, no brindan una solución viable cuando se trata de aprender a programar mediante editores de código visual (Scratch, Alice, Lego, etc.) ya que forzosamente tienen que tener una interacción con el teclado y el mouse.

En este proyecto de grado se propone el desarrollo de un prototipo de herramienta gobernada por instrucciones de voz permitiendo la creación de código visual. Con esta herramienta el proceso de enseñanza-aprendizaje de la programación a personas con discapacidad en miembros superiores será más sencillo; no se tendrá interacción con *hardware* especial; la herramienta reconocerá comandos de voz en idioma español que utilizará el usuario.

2. ANTECEDENTES

Programación y Educación. La programación es una de las pocas disciplinas enseñadas en las escuelas que prepara a los jóvenes para la revolución tecnológica que se extiende por todas las culturas y derriba fronteras [1,3].

A continuación, se cita un párrafo del manifiesto por la educación en ciencias de la computación en el siglo XXI: *"Creemos que todos los niños deberían tener la*

oportunidad de aprender ciencias de la computación, empezando en la escuela (...) Enseñamos física básica a cada niño, no con el objetivo principal de educar físicos si no porque todos ellos viven en un mundo gobernado por sistemas físicos. De la misma manera, todos los niños deberían aprender un poco de informática desde temprana edad porque van a vivir en un mundo en el que la computación está en todas partes" [4].

Programación y campo laboral. La programación es una gran base para una carrera técnica en tecnologías de información, puesto que ahora todas las empresas tienen un departamento de TIs y utilizan algún tipo de software. Los problemas que enfrenta la tecnología de software actual (manejo de millones de datos, procesamiento de datos, inteligencia artificial, *data mining*, *machine learning*,*business intelligence*, millones de visitas) son desafíos que requieren expertos en software, que se dediquen a las diferentes ramas que este campo laboral tiene. Como consecuencia, las empresas están en busca de cada vez más programadores, desarrolladores y/o informáticos; lo cual lleva a que existan más ofertas de trabajo, mejores salarios, lugares de trabajo atractivos y otros beneficios (*Google*, *YouTube*, *Jalasoft*). Por otro lado, la programación ofrece un campo laboral independiente (FreeLancer) y los conocimientos de la programación son de mucha ayuda para empezar con una empresa nueva en el campo de la tecnología [5,6,7].

Discapacidad y campo laboral. Las personas con discapacidad representan aproximadamente mil millones de personas, un 15% de la población mundial. Alrededor del 80 por ciento están en edad de trabajar. Sin embargo, su derecho a un trabajo decente, es con frecuencia denegado. A pesar de que existen leyes de inclusión laboral, las empresas no optan por contratarlos porque deben invertir más para crear un ambiente de trabajo apropiado para las personas con algún tipo de discapacidad [9].

Las personas con discapacidad se enfrentan a enormes barreras actitudinales, físicas y de información; lo que dificulta el disfrute a la igualdad de oportunidades en el mundo del trabajo. En comparación con las personas sin discapacidad, las personas con discapacidad experimentan mayores tasas de desempleo e inactividad económica y están en mayor

riesgo de una protección social insuficiente la cual es clave para reducir la pobreza extrema [8].

La programación y discapacidad: Programación para todos. Las personas con discapacidades en miembros superiores no sienten la suficiente seguridad para aprender a programar y/o recurrir a planes de enseñanza de la programación, ya que como bien se sabe para poder programar, muy aparte de la lógica se debe realizar interacción física para hacer uso de una computadora, teclado y *mouse*; por ende, pueden llegar a sentir que no tienen un lugar en el campo de la programación y que llegan a ser excluidos debido a su discapacidad. Algunas de estas personas son excluidas a pesar de poseer el potencial de una mente que piense de manera algorítmica y usando la lógica llegando a soluciones mucho mejores que otros compañeros sin alguna discapacidad. Se puede entender que por estas razones la mayoría de las personas con discapacidad en miembros superiores abandonan la oportunidad de aprender programación.

Existen casos de personas con discapacidad o pérdida de extremidades superiores que aseguran que a pesar de su discapacidad pueden ser incluidos dentro del área de la programación y desarrollo de software. No tienen la necesidad de tener que ir a una oficina dentro una empresa, no piensan que les es imposible programar ni aprender sobre programación. Tenemos el caso más popular de Max Strzelecki, el desarrollador que programa con los pies, que desarrolló un videojuego desde su hogar, su juego está en varias tiendas online y funciona con varias plataformas, el asegura que desde pequeño le gustaron mucho los videojuegos y siempre soñó con desarrollar uno [10]. Otro caso es el de Adrián Hands un programador que usaba el Morse para escribir código, después de terminar inmovilizado casi por completo por una enfermedad, solo teniendo la posibilidad de usar un dedo para escribir código, ayudó en cientos de proyectos y no dejo de lado su pasión por la programación [11]. Como estos hay más casos de personas que desean aprender a programar a pesar de sus desventajas.

Herramientas usadas en el proceso enseñanza-aprendizaje (PEA) de programación. Actualmente, existen herramientas para el proceso de enseñanza-aprendizaje de la

programación; las mismas están orientadas a jóvenes o personas interesadas en la programación. La mayoría de estas herramientas permite a las personas aprender a programar haciendo uso de un lenguaje de programación visual de bloques interconectados [12,13]. A continuación, se dará un detalle de las herramientas más conocidas:

- **Scratch:** Es una conocida herramienta *drag and drop* (jalar y pegar) que se basa en ir poniendo bloques en un orden lógico para que estos realicen tareas que los propios jóvenes van proponiendo. Su interfaz es atractiva para niños lo cual ayuda en llamar su atención para el aprendizaje [12,13].

- **Alice:** Es un entorno parecido al de Scratch, solo que este funciona en un ámbito 3D, enseña a los jóvenes programación orientada a objetos y programación de eventos. En Alice, los estudiantes arrastran y sueltan cuadros gráficos con el fin de animar un objeto y crear un programa [13].

- **Lego Mindstorms:** Quizás sea una de las herramientas más conocidas de la famosa marca de construcciones Lego. A diferencia de los anteriores ejemplos, esta herramienta esta orientada para acercar a los niños a la programación de robots. Los kits de Lego Mindstorms, que pueden adquirirse en versiones educativas y de consumo, incluyen sensores y motores. Los kits vienen con lenguajes propios de Lego, pero pueden ser modificados para trabajar con lenguajes de terceros [12,13].

Herramientas orientadas para personas con discapacidad. Existen unas cuantas herramientas para los usuarios con discapacidad en miembros superiores.

- **Darci USB**, es un dispositivo que permite utilizar la computadora a base *clicks*, lo conforma un *mousepad*, un *joystick* y un botón para mandar código morse. Dicta texto en morse a la computadora y se va escribiendo en pantalla [11]. Al funcionar con código morse forzá a la persona a aprender morse y hacer clics haciendo que ingresar texto sea moroso.

• **MouseKey,** funciona al igual que Darci USB, pero incorporando un teclado alfanumérico comprimido, por ende, debe haber interacción de parte del usuario con una mano, obligando a la interacción con *hardware* [15].

• **Dragon Dictate,** es un software de pago, una alternativa para usar la computadora con comandos de voz, puede integrarse a algunos programas y el dictado es fácil e intuitivo, se puede personalizar algunos comandos de voz [16]. Es una herramienta muy bien diseñada y obedece casi cualquier instrucción, como ventaja se pueden personalizar los comandos, pero solo funciona con entornos de escritura y edición de texto; no funciona bien con lenguajes de programación visual que es lo que la mayoría de programas para enseñanza-aprendizaje de programación utilizan.

• **VoiceCode,** fue desarrollada por un programador que quedó inmovilizado de los miembros superiores por un largo periodo de tiempo, entonces busco una forma para poder escribir código sin tener que usar sus manos y desarrollo VoiceCode, es un software adaptado a partir de Dragon Dictate, funciona como si se estuviera hablando con un robot al cual se le da instrucciones establecidas y empezara a escribir en pantalla [14]. Esta herramienta necesita la instalación de un puntero para mover el mouse con la cabeza y todo el dictado es de forma plana en inglés, si bien es útil va más orientado a un uso general para escribir texto.

3. PROBLEMA

3.1. Situación Problemática

• Las herramientas actuales para el PEA de programación requieren una interacción, aunque mínima con el teclado y/o el *mouse*. Como consecuencia, las personas con discapacidad en miembros superiores imposibilitadas de utilizar *hardware* no tienen una alternativa viable para aprender a programar sin hacer uso de estas interfaces.

• Las herramientas con reconocimiento de voz no están pensadas para programar (escribir código fuente), interactuar con editores de código y mucho menos para el proceso de

enseñanza-aprendizaje de programación. Como consecuencia, existe una brecha en el proceso de enseñanza-aprendizaje de la programación sin el uso de teclado y/o *mouse.*

- Las herramientas de ayuda en la interacción con el computador para discapacitados descritas en antecedentes necesitan un grado mínimo de interacción con *hardware*. Lo que provoca que los programadores con discapacidad deban adquirir el *hardware* y aprender a utilizarlo e interactuar con la herramienta.

3.2. Formulación del Problema

Las herramientas actuales orientadas al proceso de enseñanza-aprendizaje (PEA) de programación requieren interacción con interfaces de entrada física dificultando la enseñanza y el aprendizaje a las personas con discapacidad o perdida de miembros superiores.

4. OBJETIVOS

4.1. Objetivo General

Desarrollar un prototipo de herramienta de apoyo al proceso de enseñanza-aprendizaje de programación para personas con discapacidad o perdida de miembros superiores.

4.2. Objetivos Específicos

- Definir una gramática para formalizar las instrucciones de voz.

- Desarrollar compilador que transforme las instrucciones de voz a un lenguaje visual de bloques interconectados sintácticamente correctos.

- Desarrollar editor de código visual que permita capturar las instrucciones de voz y generar código haciendo uso del compilador desarrollado.

5. ALCANCES

- Se utilizará un editor de bloques interconectados.

- Los bloques podrán ser acomodados utilizando comandos de voz.
- El editor reconocerá instrucciones de voz definidas en la gramática.
- Se podrá crear variables.
- Se realizarán operaciones lógicas.
- Se realizarán operaciones matemáticas.
- Se podrán implementar ciclos.
- Navegación incluida utilizando comandos de voz.
- Se podrá exportar el código creado por el usuario a *javascript*, *php* y *python*.

6. LÍMITES

- El *software* realizado será un modelo del comportamiento del sistema (prototipo).
- La herramienta será desarrollada para un solo editor de código visual de bloques.
- Las funcionalidades que manejará la herramienta serán básicas: dictar instrucciones, abrir proyecto, nuevo proyecto, borrar, navegación básica de bloques (arriba, abajo, derecha, izquierda, dentro), entre otros.
- Se dará soporte a instrucciones de programación básicas. No así a la completitud de todas las instrucciones, solo las necesarias para un aprendizaje básico en programación, como sentencias: *if, else, while, for, crear nuevas variables y operaciones matemáticas*.
- No se exportará a lenguajes que no sean los contemplados en alcances como *java* o *C++*.
- La herramienta necesita conexión a internet para su correcto funcionamiento.
- Funcionará solo en Google Chrome.

7. JUSTIFICACIÓN

7.1. Justificación técnica

La utilización de esta herramienta esta orientada para mejorar y ampliar el proceso de enseñanza-aprendizaje de la programación en las herramientas de código visual, seguir las instrucciones será mucho más sencillo, con solo dictar la lógica que debe seguir el programa el editor empezará a poner el código de bloques interconectados en pantalla. Se podrá resolver problemas y/o desarrollar *software* al solo utilizar comandos de voz, aumentando la facilidad de uso y permitiendo programar solo dictando instrucciones en un orden lógico al editor de código visual.

7.2. Justificación social

La herramienta brinda una opción más sencilla para aprender a programar, sin tener que lidiar con las complicaciones que trae aprender inglés (que es el lenguaje en el que todos los códigos de programación están diseñados), aprender nombres de instrucciones del lenguaje de programación y brindará una solución viable al uso de herramientas de enseñanza de programación en código visual de bloques al alcance de personas con discapacidad o pérdida de miembros superiores que podrán empezar a aprender a programar de una manera más sencilla y viable sin tener que realizar ninguna interacción con *mouse* o teclado u otro tipo de *hardware*. Como consecuencia, el campo laboral para las personas con discapacidad o perdida de miembros superiores se ampliará dentro de la rama de la tecnología.

8. FUNDAMENTO TEÓRICO

8.1. Proceso de enseñanza-aprendizaje

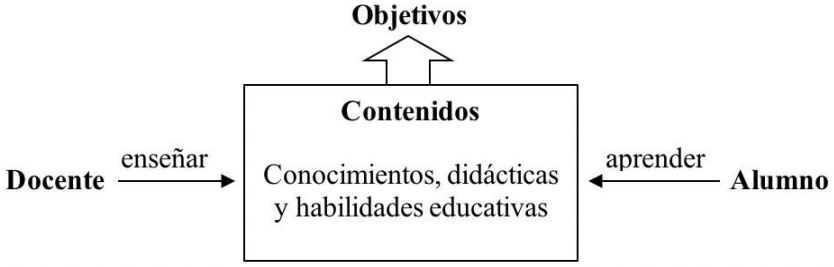
Enseñanza y aprendizaje forman parte de un único proceso que tiene como fin la formación del estudiante. La referencia etimológica del termino enseñar puede servir de apoyo inicial: enseñar es señalar algo a alguien; es mostrar lo que se desconoce.

Esto implica que hay un sujeto que conoce y que puede enseñar, y otro que desconoce y que puede aprender. El que puede enseñar, quiere enseñar y sabe enseñar (el docente); El que puede aprender quiere y sabe aprender (el alumno).

Aparte de estos agentes, están los contenidos, esto es, lo que se quiere enseñar o aprender y los procedimientos o instrumentos para enseñarlos o aprenderlos. Cuando se enseña algo es para conseguir alguna meta (objetivos).

La siguiente figura esquematiza el proceso enseñanza-aprendizaje detallando el papel de los elementos básicos.

Figura 1
Elementos del proceso enseñanza-aprendizaje



Fuente: Elaboración propia 2018

El proceso de aprender es el proceso complementario de enseñar. Aprender es el acto por el cual un alumno intenta captar y elaborar los contenidos expuestos por el docente, o por cualquier otra fuente de información. Se alcanza a través de técnicas de estudio o de trabajo intelectual. Este proceso de aprendizaje es realizado en función de unos objetivos, que pueden o no identificarse con los del profesor y se lleva a cabo dentro de un determinado contexto [27][28].

8.2. Herramientas de apoyo al proceso de enseñanza-aprendizaje

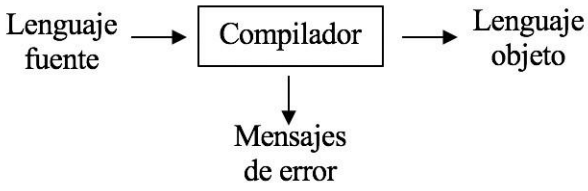
Son instrumentos que permiten orientar los nuevos procesos de enseñanza-aprendizaje hacia los diferentes entornos que tienden a promover la construcción de espacios de aprendizaje más dinámicos e interactivos [17]. En el ámbito de la programación las herramientas de apoyo al PEA según investigaciones han dado muy buen resultado al incluirlas en colegios en los cuales se ve la materia de programación, lo que incentiva a

que se siga desarrollando mas *software* dirigido al apoyo del proceso de enseñanza y aprendizaje de programación [24].

8.3. Compilador

Un compilador es un programa informático que traduce un programa que ha sido escrito en un lenguaje de programación a un lenguaje común. Usualmente el compilador traduce a lenguaje de maquina, aunque también puede ser traducido a un código intermedio (*bytecode*) o a texto. Este proceso de traducción se conoce como compilación, compilar consiste en traducir un programa escrito en un cierto lenguaje a otro [19]. A grandes rasgos, un compilador es un programa que lee un programa escrito en un lenguaje, el lenguaje fuente y lo traduce a un programa equivalente en otro lenguaje, el lenguaje objeto [29].

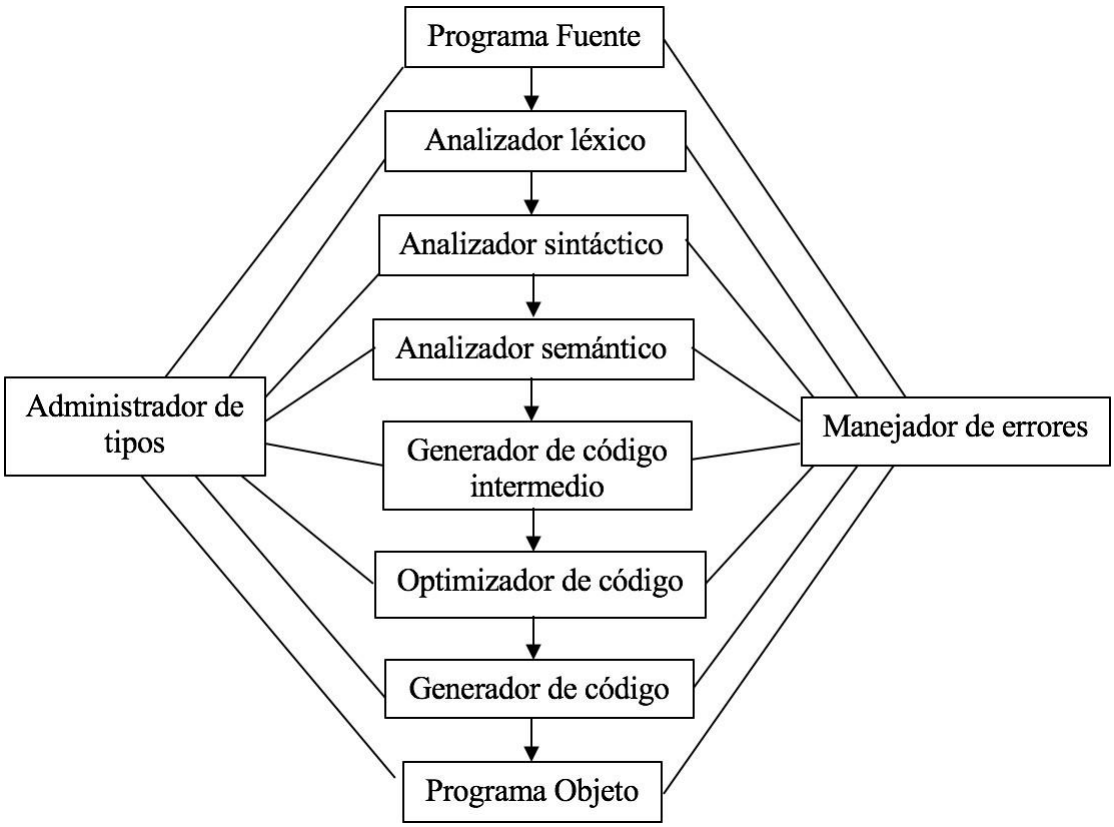
Figura 2
Esquema de un compilador



Fuente: Elaboración propia 2018

• **Estructura de un compilador.** El compilador consta de una serie de pasos, generalmente entrelazados y como resultado convierte un lenguaje fuente en un lenguaje objeto. Los pasos o fases de la compilación se muestran a continuación [29,30,31].

Figura 3
Fases de un compilador



Fuente: Elaboración propia 2018

• **Análisis léxico.** En esta etapa los caracteres introducidos en el programa fuente se leen de izquierda a derecha de arriba hacia abajo y se agrupan en *tokens* (caracteres relacionados entre si), que luego serán suministrados al analizador sintáctico [30,31].

• **Análisis sintáctico.** La tarea del analizador sintáctico es procesar los lexemas que suministra el analizador léxico y comprobar que están bien ordenados, y si no lo están generar informes de error correspondientes. Si el orden es correcto se generará un árbol sintáctico teórico[30,31].

• **Análisis semántico.** La fase de análisis semántico revisa el programa fuente para tratar de encontrar errores semánticos y reúne la información sobre los tipos para la fase posterior de generación de código. En ella se utiliza la estructura jerárquica determinada

por la clase de análisis sintáctico para identificar los operadores y operandos de expresiones y proposiciones[30,31].

- **Generación de código intermedio.** Se puede considerar esta representación intermedia como un programa para una máquina abstracta. Esta representación intermedia debe tener dos propiedades importantes; debe ser fácil de producir y fácil de traducir al programa objeto[30,31].

- **Optimizador de código.** Revisa el código generado a varios niveles de abstracción y realiza las optimizaciones aplicables al nivel de abstracción [30].

- **Generador de código final.** La fase final de un compilador es la generación de código objeto. Es el proceso mas mecánico, ya que consiste en ir pasando las distintas instrucciones del código intermedio al lenguaje objeto [30,31].

- **Manejador de errores.** Cada fase puede encontrar errores. Sin embargo, después de detectar un error, cada fase debe tratar de alguna forma ese error, para poder continuar la compilación, permitiendo la detección de más errores en el programa Fuente Un compilador que se detiene cuando encuentra el primer error, no resulta tan útil como debiera [30,31].

8.4. Analizador lexicográfico

Un analizador léxico o analizador lexicográfico (en inglés scanner) es la primera fase de un compilador consistente en un programa que recibe como entrada el código fuente de otro programa (secuencia de caracteres) y produce una salida compuesta de *tokens* (componentes léxicos) o símbolos. Estos *tokens* sirven para una posterior etapa del proceso de traducción, siendo la entrada para el analizador sintáctico (en inglés *parser*) [23].

Como el analizador léxico es la parte del compilador que lee el texto fuente. También puede realizar ciertas funciones secundarias en la interfaz del usuario, como eliminar del programa fuente comentarios y espacios en blanco en forma de caracteres de espacio en

blanco, caracteres TAB y de línea nueva. Otra función es relacionar los mensajes de error del compilador con el programa fuente. Por ejemplo, el analizador léxico puede tener localizado el número de caracteres de nueva línea detectados, de modo que se pueda asociar un número de línea con un mensaje de error.

En algunos compiladores, el analizador léxico se encarga de hacer una copia del programa fuente en el que están marcados los mensajes de error. Si el lenguaje fuente es la base de algunas funciones de pre procesamiento de macros, entonces esas funciones del preprocesador también se pueden aplicar al hacer el análisis léxico [25].

8.5. Gramática

La gramática es un ente formal para especificar, de una manera finita, el conjunto de cadenas de símbolos que constituyen un lenguaje [22]. En programación las gramáticas son capaces de describir la mayoría, pero no todas, de las sintaxis de los lenguajes de programación. Un analizador léxico efectúa una cantidad limitada de análisis sintáctico conforme produce la secuencia de componentes léxicos a partir de los caracteres de entrada. Ciertas limitaciones de la entrada, como el requisito de que los identificadores se declaren antes de ser utilizados, no pueden describirse mediante una gramática independiente del contexto. Por tanto, las secuencias de componentes léxicos aceptadas por un analizador sintáctico forman un súper-conjunto de un lenguaje de programación; las fases posteriores deben analizar la salida del analizador sintáctico para garantizar la obediencia a reglas que el analizador sintáctico no comprueba [26].

8.6. Expresiones Regulares

Las expresiones regulares son una serie de caracteres que forman un patrón, normalmente representativo de otro grupo de carácteres mayor, de tal forma que podemos comparar el patrón con otro conjunto de carácteres para ver las coincidencias.

Las expresiones regulares están disponibles en casi cualquier lenguaje de programación, pero, aunque su sintaxis es relativamente uniforme, cada lenguaje usa su propio dialecto.

Una Expresión Regular nos sirve para definir lenguajes, imponiendo restricciones sobre las secuencias de caracteres que se permiten en el lenguaje que estamos definiendo. Por tanto, una Expresión Regular estará formada por el conjunto de caracteres del alfabeto original, más un pequeño conjunto de caracteres extra (meta-caracteres), que nos permitirán definir estas restricciones[33].

8.7. Comandos de voz

Los comandos de voz sirven para controlar las aplicaciones o sistemas sin necesidad de utilizar el teclado o la pantalla. A menudo, estos sistemas de comandos de voz se llevan a cabo por medio del teléfono y permiten al usuario comunicar información o comandos directamente a través de un menú de navegación controlado por voz y sin el uso de otros recursos. Estos sistemas también se denominan sistemas IVR (sistemas de respuesta de voz interactiva). El control de los comandos de voz puede estar automatizado total o parcialmente [20].

8.8. Código visual de bloques interconectados

El concepto de programación visual es un poco confuso ya que actualmente se le considera programación visual a los lenguajes de programación textual que tienen una interfaz gráfica para poder visualizar lo que uno esta desarrollando. Este concepto en programación visual es erróneo ya que este es aquel que por medio de iconos se puede ir creando programas sin tener un lenguaje textual atrás de el [18].

La programación visual (*visual programming*) se refiere al desarrollo de software donde las notaciones gráficas y los componentes de software manipulables interactivamente son usados principalmente para definir y componer programas [18].

La programación visual se define comúnmente como el uso de expresiones visuales (tales como gráficos, animación o iconos) en el proceso de la programación, pueden ser utilizadas para formar la sintaxis de los nuevos lenguajes de programación visuales que conducen a los nuevos paradigmas tales como programación por la demostración; o

pueden ser utilizadas en las presentaciones gráficas del comportamiento o de la estructura de un programa [18].

El objetivo de la programación visual es mejorar la comprensión de los programas y simplificar la programación en sí. Más allá, la programación visual deberá fomentar a los usuarios finales a construir sus propios programas, que de otra forma deben ser escritos por programadores profesionales [18].

8.9. Prototipo

En el ámbito de la informática, se conoce como prototipo al modelo que se desarrolla de un software para reflejar cómo se comporta un sistema. Estos prototipos se utilizan para comprender cómo funciona el sistema en cuestión [21].

9. PROPUESTA METODOLÓGICA

En este proyecto se aplicará una metodología de desarrollo iterativo incremental haciendo uso de las siguientes prácticas agiles:

- Reuniones periódicas con los sujetos de estudio para ver los avances y obtener retroalimentación.

- Demo cada reunión, una pequeña demo no más de 5 minutos para entender lo avanzado del proyecto.

- Seguimiento de la iteración con tableros de historias de usuario.

- Refactorización continua.

Se utilizará esta metodología por los periodos de entrega de avances que existe y la parte ágil para la flexibilidad que tendrá el desarrollo de la herramienta de acuerdo a la retroalimentación que se ira recibiendo de los sujetos de estudio.

10. CRONOGRAMA

Tabla 1
Objetivo especifico y sus respectivas acciones

Objetivos específicos	Acciones
Definir una gramática para formalizar las instrucciones de voz.	• Definir alfabeto. • Definir gramática libre de contexto haciendo uso de la notación Backus-Naur. • Diseñar un modelo objeto que permita modelar la sintaxis a través de un árbol de sintaxis abstracta.
Desarrollar un compilador que transforme las instrucciones de voz a un lenguaje visual de bloques interconectados sintácticamente correctos.	• Seleccionar herramienta de reconocimiento de voz que mejor se adapte a nuestras necesidades. • Crear un analizador lexicográfico. • Crear un Analizador sintáctico/semántico (*Parser*) que tome las instrucciones de voz como entrada y genere un árbol de sintaxis abstracta en base a la gramática definida. • Crear un Generador de Código que permita transformar el árbol de sintaxis abstracta en código de lenguaje visual (código intermedio).
Desarrollar un editor de código visual que permita capturar las instrucciones de voz y generar código haciendo uso de nuestro compilador.	• Mediante la herramienta *Blockly* desarrollar el editor de código visual. • Integrar el módulo al editor desarrollado. • Hacer pruebas del prototipo para recabar información de su funcionamiento. • Con los resultados obtenidos realizar mejoras en el editor y el compilador. • Realizar pruebas con la herramienta ya terminada

Fuente: Elaboración propia 2018

Fuente: Elaboración propia

Objetivos específicos	Acciones
Definir una gramática para formalizar las instrucciones de voz.	• Definir alfabeto. • Definir gramática libre de contexto haciendo uso de la notación Backus-Naur. • Diseñar un modelo objeto que permita modelar la sintaxis a través de un árbol de sintaxis abstracta.
Desarrollar un compilador que transforme las instrucciones de voz a un lenguaje visual de bloques interconectados sintácticamente correctos.	• Seleccionar herramienta de reconocimiento de voz que mejor se adapte a nuestras necesidades. • Crear un analizador lexicográfico. • Crear un Analizador sintáctico/semántico (*Parser*) que tome las instrucciones de voz como entrada y genere un árbol de sintaxis abstracta en base a la gramática definida. • Crear un Generador de Código que permita transformar el árbol de sintaxis abstracta en código de lenguaje visual (código intermedio).
Desarrollar un editor de código visual que permita capturar las instrucciones de voz y generar código haciendo uso de nuestro compilador.	• Mediante la herramienta *Blockly* desarrollar el editor de código visual. • Integrar el módulo al editor desarrollado. • Hacer pruebas del prototipo para recabar información de su funcionamiento. • Con los resultados obtenidos realizar mejoras en el editor y el compilador. • Realizar pruebas con la herramienta ya terminada

Tabla

Objetivo especifico y sus respectivas acciones

Tabla 3
Prueba dos y sus respectivas acciones
Fuente: URQUIOLA 95

Prueba dos	Acciones
Definir una gramática para formalizar las instrucciones de voz.	• Definir alfabeto. • Definir gramática libre de contexto haciendo uso de la notación Backus-Naur. • Diseñar un modelo objeto que permita modelar la sintaxis a través de un árbol de sintaxis abstracta.

Tabla 5

Prueba tres	Acciones
Definir una gramática para formalizar las instrucciones de voz.	• Definir alfabeto. • Definir gramática libre de contexto haciendo uso de la notación Backus-Naur. • Diseñar un modelo objeto que permita modelar la sintaxis a través de un árbol de sintaxis abstracta.

Prueba tres y sus respectivas acciones
Fuente: Elaboración propia en base a PÉREZ

Tabla 1
Prueba cuatro y sus respectivas acciones
Fuente: Elaboración propia 2018

Prueba cuatro	Acciones
Definir una gramática para formalizar las instrucciones de voz.	• Definir alfabeto. • Definir gramática libre de contexto haciendo uso de la notación Backus-Naur. • Diseñar un modelo objeto que permita modelar la sintaxis a través de un árbol de sintaxis abstracta.

Figura 4
Cronograma de actividades

OBJETIVOS ESPECIFICOS	ACTIVIDADES	ITERACIÓN									
		1	2	3	4	5	6	7	8	9	10
Definir una gramática para formalizar las instrucciones de voz.	Definir alfabeto.										
	Definir una gramática libre de contexto haciendo uso de la notación Backus-Naur.										
	Diseñar un modelo objeto que permita modelar la sintaxis a través de un árbol de sintaxis abstracta.										
	Investigar herramientas de reconocimiento de voz y seleccionar la que mejor se adapte a nuestras necesidades.										
Desarrollar un compilador que transforme las instrucciones de voz a un lenguaje visual de bloques interconectados sintácticamente correctos.	Crear un analizador lexicográfico.										
	Crear un Analizador sintactico/semantico (Parser) que tome las instrucciones de voz como entrada y genere un árbol de sintaxis abstracta en base a la gramática definida.										
	Crear un Generador de Código que permita transformar el árbol de sintaxis abstracta en código de lenguaje visual (código intermedio).										
	Mediante la herramienta Blockly desarrollar el editor de código visual.										
Desarrollar un editor de código visual que permita capturar las instrucciones de voz y generar código haciendo uso de nuestro compilador.	Integrar el módulo al editor desarrollado.										
	Hacer pruebas del prototipo para recabar información de su funcionamiento.					Entrega 1er Prototipo					
	Con los resultados obtenidos realizar mejoras en el editor y el compilador.										
	Realizar pruebas con la herramienta ya terminada										

Fuente: Elaboración propia 2018

11. BIBLIOGRAFÍA

[1] CINCODIAS (2016), "¿Por qué es importante aprender a programar?". En: <https://cincodias.elpais.com/cincodias/2016/02/11/tecnologia/1455182591_761653.html>, (fecha de consulta 26/02/2018).

[2] FUTURIZABLE (2016), "Por qué debemos enseñar a los niños a programar". En: <https://futurizable.com/educacion>, (fecha de consulta 02/03/2018).

[3] VELASCO, Juan Jesús (2014), "Niños programadores: para qué sirve la enseñanza de programación en las escuelas". En: <http://www.eldiario.es/turing/Ninos-programadores-ensenanza-programacion-escuelas_0_293970921.html>, (fecha de consulta 28/02/2018).

[4] NAUGHTON, John (2012). "A manifesto for teaching computer science in the 21st century". En: The Guardian, BST, p.3 (fecha 31/03/2012).

[5] DUARTE, Eugenio (2012). "¿Por Qué Aprender A Programar?". En: <http://blog.capacityacademy.com/2012/05/25/por-que-aprender-a-programar/>, (fecha de consulta 28/02/2018).

[6] HERNÁNDEZ, Uriel (s.f.). "Programar es importante. ¿Por qué?". En: <https://codigofacilito.com/articulos/programar-es-importante-por-que>, (fecha de consulta 01/03/2018).

[7] KIRKPATRICK, David (2011). "Now Every Company Is A Software Company". En: https://www.forbes.com/sites/techonomy/2011/11/30/now-every-company-is-a-software-company/#696aa5a6f3b1>, (fecha de consulta 02/03/2018).

[8] ORGANIZACIÓN INTERNACIONAL DEL TRABAJO (s.f.). Discapacidad y trabajo. <http://www.ilo.org/global/topics/disability-and-work/WCMS_475652/lang--es/index.htm>, (fecha de consulta 05/03/2018).

[9] SEBASTIÁN HERRANZ, Margarita y REYES NOYA, Arnaiz (2009). Adaptación de puestos de trabajo, España: CEAPAT

[10] PASCUAL, Juan Antonio (2014). "Max Strzelecki, el desarrollador que programa con los pies". En: <https://computerhoy.com/noticias/software/max-strzelecki-desarrollador-que-programa-pies-19425>, (fecha de consulta 18/02/2018).

[11] JOHNBO (2012). "Adrian Hands, el impresionante ejemplo de un programador que usaba el Morse para escribir código". En: <https://www.genbetadev.com/desarrolladores/adrian-hands-el-impresionante-ejemplo-de-un-programador-que-usaba-el-morse-para-escribir-codigo>, (fecha de consulta 24/02/2018).

[12] JAIMOVICH, Desirée (2017). "Herramientas gratuitas y online para aprender a programar". En: <https://www.infobae.com/tecno/2017/10/16/herramientas-gratuitas-y-online-para-aprender-a-programar/> , (fecha de consulta 03/03/2018).

[13] EL RINCÓN DE JMACOE (2018). "4 Herramientas para enseñar a los niños a programar". En: <http://blog.jmacoe.com/aplicaciones/4-herramientas-ensenar-ninos-programar/>, (fecha de consulta 02/03/2018).

[14] VOICECODE (s.f.). "VOICE CODE". En: <https://voicecode.io>, fecha de consulta 20/02/2018).

[15] López-Escribano, C. y Sánchez-Montoya, R. (2012). Scratch y necesidades educativas especiales: Programación para todos. *RED, Revista de Educación a Distancia*. Número34.

[16] NUANCE (2018). "Dragon Dictate". En: <https://www.nuance.com/es-es/dragon.html>, (fecha de consulta 05/03/2018).

[17] RODRÍGUEZ SALAS, Karla y BARBOZA JIMÉNEZ, Lucrecia (s.f.). Las TIC como apoyo al proceso de enseñanza-aprendizaje en Bibliotecología. Universidad Nacional de Costa Rica

[18] HERNÁ NDEZ VALDELAMAR, Eugenio Jacobo y URIBE LEÓ N, Humberto Manuel (s.f.). El paradigma de la programación visual. Fundación Arturo Rosenblueth. Insurgentes Sur 670-3. Colonia del Valle. D.F, México.

[19] WIKIPEDIA (s.f.). "Compilador". En: <https://es.wikipedia.org/wiki/Compilador>, (fecha de consulta 05/03/2018).

[20] NFON (s.f.). "Comandos de voz". En: <https://www.nfon.com/es/acerca-de-nfon/recursos/glosario/comandos-de-voz/>, (fecha de consulta 01/04/2018).

[21] DEFINICION.DE (s.f.). "Definición de prototipo". En: <https://definicion.de/prototipo/>, (fecha de consulta 03/04/2018).

[22] CUEVA LOVELLE, Juan Manuel (2001). *Lenguajes, gramáticas y autómatas.* España: Universidad de Oviedo.

[23] WIKIPEDIA (s.f.). "Analizador léxico". En: <https://es.wikipedia.org/wiki/Analizador_léxico>, (fecha de consulta 11/04/2018).

[24] ZULETA MEDINA, Alejandra y CHACEZ TORRES, Anivar (2011). "Uso de herramientas informáticas como estrategia para la enseñanza de la programación de computadores". En: *Revista Unimar*, Unimar, Año (s.f.), Nº 57, Pasto, Colombia.

[25] UNIVERSIDAD AUTÓNOMA DEL ESTADO DE HIDALGO (s.f.). "Análisis Léxico función del analizador léxico". En: Universidad autónoma del estado de Hidalgo, *Autómatas y compiladores*. (s.l.),(s.e.).

[26] UNIVERSIDAD AUTÓNOMA DEL ESTADO DE HIDALGO (s.f.). "Análisis sintáctico Escritura de una Gramática". En: Universidad autónoma del estado de Hidalgo, *Autómatas y compiladores*. (s.l.),(s.e.).

[27] ANÓNIMO (s.f.). "El proceso de enseñanza-aprendizaje". En: <https://www.infor.uva.es/~descuder/docencia/pd/node24.html>, (fecha de consulta 01/05/2018).

[28] AULANEO (s.f.). “Proceso Enseñanza/Aprendizaje”. En: <https://aulaneo.wordpress.com/didactica/el-proceso-ensenanzaaprendizaje/>, (fecha de consulta 02/05/2018).

[29] UNIVERSIDAD AUTÓNOMA DEL ESTADO DE HIDALGO (s.f.). “Introducción, Fases de un compilador y sus fundamentos teóricos”. En: Universidad autónoma del estado de Hidalgo, *Autómatas y compiladores*. (s.l.),(s.e.).

[30] UNICEN (s.f.). “Diseño de compiladores I”. En: <http://www.exa.unicen.edu.ar/catedras/compila1/index_archivos/Introduccion.pdf>, (fecha de consulta 03/05/2018).

[31] RUIZ CATALÁN, Jacinto (2010), *Compiladores Teoría e implementación*. Madrid-España: RC libros

[32] ANÓNIMO (s.f.). “Procesamiento de lenguajes (PL)”. En: <https://www.dlsi.ua.es/asignaturas/pl/downloads/1415/gramaticas.pdf>, (fecha de consulta 03/05/2018).

[33] VILORIA LANERO, Alejandro (s.f.). “Introducción a las expresiones regulares”. En: Universidad Valladolid, *Teoría de autómatas y lenguajes formales*. (s.l.), Universidad Valladolid.