SOFTWARE DE APRENDIZAJE DE ALGORITMOS ESTRUCTURADOS

BRANDON ALEXANDER RODRIGUEZ CARO

UNIVERSIDAD PILOTO DE COLOMBIA
FACULTAD DE INGENIERÍA
INGENIERÍA DE SISTEMAS
BOGOTÁ D.C
2020

CONTENIDO

Pág.
1. TITULO 8
2. INTRODUCCIÓN: FORMULACIÓN DEL PROBLEMA 9
3. PREGUNTA PROBLEMA 11
4. JUSTIFICACIÓN 12
5. ALCANCE 15
6. OBJETIVOS 17
6.1 OBJETIVO GENERAL 17
6.2 OBJETIVOS ESPECÍFICOS 17
7. MARCO REFERENCIAL 18
7.1 MARCO CONCEPTUAL 18
7.1.1 Java. 18
7.1.2 Intérprete. 18
7.1.3 Compilador. 19
7.1.4 Analizador Léxico/ Morfológico. 19
7.1.5 Analizador Sintáctico. 20
7.1.6 Tabla de Símbolos. 21
7.1.7 Investigación Cualitativa. 22
7.1.8 Investigación Cuantitativa. 23
7.1.9 Metodología Ágil SCRUM. 23
7.1.10 Pedagogía en cursos Introductorios a la programación. 24
7.1.11 Indentación. 24
7.1.12 Heurística. 25
7.2 MARCO TEÓRICO 25
7.2.1 Procesos de aprendizaje en la introducción a la programación. 26
7.2.2 Dificultades en el aprendizaje del paradigma estructurado. 29
7.2.3 Causas y variables involucradas. 31
7.2.4 El software educativo en la introducción a la programación. 32
7.2.5 Conclusión. 35
7.3 ESTADO ACTUAL 35

8. DISEÑO METODOLÓGICO PRELIMINAR 39
8.1JUSTIFICACIÓN DE METODOLOGÍA – INVESTIGACIÓN CUALITATIVA..... 39
8.2JUSTIFICACIÓN DE METODOLOGÍA – DESARROLLO ÁGIL SCRUM 40
9. RECURSOS DISPONIBLES..... 42
9.1 RECURSOS MATERIALES..... 42
9.2 RECURSOS INSTITUCIONALES 42
9.3 RECURSOS FINANCIEROS 42
10. CRONOGRAMA 43
11. ESTADO DEL ARTE..... 46
11.1 EN EL MUNDO 46
11.2 EN LATINOAMÉRICA..... 47
11.3 EN COLOMBIA 48
12. LEVANTAMIENTO DE REQUERIMIENTOS 50
12.1 DIFICULTADES EN UN LENGUAJE FORMAL 50
12.2 REQUERIMIENTOS FUNCIONALES..... 51
12.2.1 Integración y remoción de estructuras básicas de programación por bloques. 52
12.2.2 Ingresar expresiones aritméticas y lógicas en los campos requeridos. 52
12.2.3 Ejecutar o simular la ejecución del pseudocódigo diseñado. 52
12.2.4 Generar una prueba de escritorio automática. 53
12.2.5 Ingresar una prueba de escritorio manual. 53
12.2.6 Lectura de datos para la asignación de variables en ejecución. 53
12.2.7 Impresión de datos en ejecución. 53
12.2.8 Notificar errores léxicos, sintácticos y semánticos en las expresiones..... 54
12.2.9 Guardar pseudocódigo en un archivo para su posterior uso. 54
12.2.10 Abrir pseudocódigo desde un archivo para continuar su desarrollo. 54
12.3 REQUERIMIENTOS NO FUNCIONALES 54
12.3.1 Lenguaje de programación Java..... 55
12.3.2 Aplicación Standalone. 55
12.3.3 Gestor de archivos. 56
13. DESARROLLO 57
13.1 SCRUM..... 57
13.1.1 Product owner. 57

13.1.2 Development Team y SCRUM Master. 57
13.1.3 Sprints. 58
13.1.4 Historias de Usuario. 58
13.1.5 Product backlog. 59
13.1.6 Sprint planning. 59
13.1.7 Daily scrum. 60
13.1.8 Sprint retrospective. 60
13.1.9 Sprint review. 60
13.2 MODELOS Y VISTAS 60
13.2.1 Vista funcional. 60
13.2.2 Vista de contexto. 62
13.2.3 Vista de despliegue. 63
13.2.4 Diagrama de clases. 64
13.3 SISTEMA 65
13.3.1 Entorno gráfico. 65
13.3.2 Sistema de generación de bloques. 66
13.3.3 Analizador léxico. 67
13.3.4 Analizador sintáctico. 68
13.3.5 Analizador semántico. 70
13.3.5.1 Notacion polaca inversa. 71
13.3.5.2 Evaluación de notación polaca inversa. 72
13.3.6 Tabla de símbolos. 73
13.3.7 Secuenciador. 74
13.3.7.1 Nodo secuenciador. 76
13.3.7.2 Nodo de cierre. 76
13.3.8 Ejecutor. 76
13.3.8.1 Declaración. 77
13.3.8.2 Asignación. 77
13.3.8.3 Condicional e Iteración condicional. 77
11.3.8.4 Iteración definida. 77
13.3.8.5 Lectura. 78
13.3.8.6 Escritura. 78

14. VALIDACIÓN 79
14.1 PRUEBAS FUNCIONALES 79
14.2 GRUPO FOCAL 83
15. CONCLUSIONES 90
16. RECOMENDACIONES 94
17. BIBLIOGRAFÍA 95
18. ANEXOS 107
18.1 MARCO HISTORICO 107
18.1.1 Java. 107
18.1.2 Intérprete. 107
18.1.3 Compilador. 107
18.1.3 Analizador Léxico-Morfológico. 108
18.1.4 Analizador Sintáctico. 108
18.1.5 Tabla de Símbolos. 109
18.1.6 Pedagogía en cursos introductorios a la programación. 110
18.1.7 Indentación. 110
18.1.8 Heurística de George Polya. 111
18.2 DIAGRAMA DE CLASES 112

LISTA DE TABLAS

Tabla 1. Estado Actual, comparación entre aplicativos 36
Tabla 2. Relación objetivo - metodología..... 39
Tabla 3. Comparación de metodologías de desarrollo de software..... 40
Tabla 4. Cronograma del proyecto 43
Tabla 5. Orden de historias de usuario - *Sprint Goals*. 58
Tabla 6. Ejemplos de expresiones en notación polaca inversa. 71
Tabla 7. Jerarquía de los operadores. 72
Tabla 8. Set de pruebas de lexicas 79
Tabla 9. Set de pruebas de expresiones aritméticas. 80
Tabla 10. Set de pruebas de expresiones lógicas. 82

LISTA DE FIGURAS

Figura 1. Estructura de un intérprete 19
Figura 2. Autómata finito para comentarios tipo C..... 20
Figura 3. Arbol sintactico para una asignación 20
Figura 4. Uso de tabla de símbolos en conjunto con tabla de referencias 22
Figura 5. Proceso cualitativo..... 22
Figura 6. Esquema de la Metodológia Ágil SCRUM 23
Figura 7. Vista funcional del sistema interno de TestDesk. 61
Figura 8. Vista de contexto de TestDesk. 62
Figura 9. Vista de despliegue de TestDesk. 63
Figura 10. Resumen del diagrama de clases de TestDesk. 64
Figura 11. Componentes gráficos de editores de código estilizados (RichText a la izquierda, StyledTextArea a la derecha). 65
Figura 12. Proceso de inserción de estructuras (campo habilitado a edicion libre en amarillo) 66
Figura 13. Notificación de error léxico. 68
Figura 14. Ejemplo de autómata determinista finito para la declaración de una o varias variables. 68
Figura 15. Notificación de error sintáctico. 69
Figura 16. Ejemplo de árbol de análisis del programa con anotaciones. 70
Figura 17. Notificación de error semántico. 73
Figura 18. Ejemplo de tabla de símbolos usando una *HashTable*. 74
Figura 19. Ejemplo de pseudocódigo y de lista de instrucciones del mismo. 75
Figura 20. Instantánea de ejecución. 78
Figura 21. Resultado general respecto a la pregunta de usabilidad en la encuesta. 85
Figura 22. Resultado general respecto a la pregunta de afabilidad en la resolución de problemas en la encuesta. 86
Figura 23. Resultado general respecto a las preguntas de detección de errores y ejecución paso a paso en la encuesta. 87
Figura 24. Resultado general respecto a las preguntas calificativas del prototipo en la encuesta. 88
Figura 25. Uso de Tabla de Simbolos para cada ámbito (izquierda) y uso de Tabla de Simbolos para todos los ámbitos (derecha). 109

1. TITULO

TESTDESK: SOFTWARE DE APRENDIZAJE DE ALGORITMOS ESTRUCTURADOS

2. INTRODUCCIÓN: FORMULACIÓN DEL PROBLEMA

Actualmente como lo menciona Blake[1], la programación se ha convertido en una actividad esencial en el desarrollo de diferentes ciencias como las ingenierías. No solo ellas sino que también, como lo menciona Shackelford y LeBlanc[2], otro tipo de disciplinas la han adoptado como un proceso esencial en los programas académicos. Más allá de esto, la principal razón se debe a la transformación digital que ha conllevado al surgimiento de nueva era para el mercado laboral. Dicho mercado desde hace unos años ha entrado en un periodo de escasez en talento TI, donde estas compañías requieren de perfiles que puedan satisfacer sus necesidades de cara a la tecnología y a la demanda de esta. Esta situación se ha vuelto cada vez más evidente con el pasar de los años, en una entrevista con Mark Zuckerberg[3] en el 2013 el menciona "El completo limite en el sistema es que no hay suficientes personas que estén entrenadas y tengan estas habilidades hoy en día". Por otro lado, en un foro del 2019 de la Universidad de los Andes[4] se expresa por parte de diferentes empresarios la dificultad de encontrar perfiles con talento TI que puedan valorizar su empresa en el equipo de trabajo. Por último, Cisco[5] determinó en un estudio que para el 2019 existe una escasez de 500.000 personas en Networking (ciencia relacionada con TI).

Todo esto manifiesta un grado de preocupación por parte de comunidades educativas ya que al menos en la Universidad de los Andes[6] se mencionó en un foro del 2019 que se están graduando tan solo 60 estudiantes por año. Debido a

---

[1] BLAKE, Jonathan D. The Journal of Computing Sciences in Colleges [en línea]. En: Papers of the Sixteenth Annual CCSCNortheastern Conference. (15-16, noviembre. Massachusetts, Estados Unidos). Language considerations in the first year CS curriculum. Springfield: Western New England College. 2011. p. 124-129. [Consultado: 25 de septiembre de 2020]. Disponible en: https://www.researchgate.net/publication/234781683

[2] SHACKELFORD, Russell y LEBLANC, Richard. Teaching and Learning in an Era of Change [en línea]. En: Proceedings Frontiers in Education 1997 27th Annual Conference. (5-8, noviembre. Pennsylvania, Estados Unidos). Introducing Computer Science Fundamentals Before Programming. Illinois: Stipes Publishing. 1997. p. 285-289. [Consultado: 29 de septiembre de 2020]. Disponible en: https://ieeexplore.ieee.org/document/644858

[3] Code.org. Lo que la mayoría de las escuelas no enseñan [video]. Estados Unidos: YouTube, Code.org. (23 de febrero de 2013). 5:43 minutos. [Consultado: 19 de abril de 2021]. Disponible en: https://www.youtube.com/watch?v=nKIu9yen5nc

[4] Universidad de los Andes. REVIVA EL TERCER FORO: DÉFICIT EN TALENTO TI [video]. Bogotá: YouTube, Universidad de los Andes. (2 de septiembre de 2019). 189:08 minutos. [Consultado: 2 de octubre de 2020]. Disponible en: https://uniandes.edu.co/es/noticias/ingenieria/tercer-foro-deficit-en-talento-ti

[5] PINEDA, Evelyn y GONZALEZ, Carlos. Networking Skills in Latin America. En: IDC Analyze the Future [en línea]. mayo de 2016 pp. 1-32. [Consultado: 29 de septiembre de 2020] . Disponible en: https://www.cisco.com/c/dam/assets/csr/pdf/IDC_Skills_Gap_-_LatAm.pdf

[6] UNIVERSIDAD DE LOS ANDES, Op. cit., [video]

ello, se han llevado a cabo iniciativas gubernamentales que puedan suplir esta demanda en la industria TIC como lo es Jóvenes 4.0 del Ministerio TIC[7] la cual propone formar a 100.000 programadores.

Más allá de esto, la situación académica en las ciencias de la computación no es muy óptima ya que según un estudio realizado por Bennedsen y Caspersen[8] con datos de la UNESCO, sólo el 26.8% de los estudiantes inscritos en esta carrera son graduados. Cabe aclarar que el enfoque de esta propuesta no va únicamente dirigido a estudiantes pertenecientes a ciencias de la computación ni carreras a fines sino a estudiantes cuyo programa académico incluya introducción a la programación. Desde este punto de vista más preciso, Bennedsen y Caspersen[9] en ese mismo estudio estadístico llevado a cabo en instituciones educativas de Norte América, Europa y parte de Oceanía determinaron que cerca del 28% de estudiantes fracasan (pierden, omiten o cancelan la materia) en cursos introductorios a la programación. Por tanto la pregunta problema para la siguiente investigación es la que se evidencia a continuación.

---

[7] MINISTERIO DE TECNOLOGÍAS DE LA INFORMACIÓN Y LAS COMUNICACIONES [sitio web]. Colombia. MINTIC. MinTIC inicia proyecto para formar 100.000 colombianos en lenguaje de programación. [Consultado: 29 de septiembre de 2020]. Disponible en: https://www.mintic.gov.co/portal/inicio/Sala-de-Prensa/Noticias/146273:MinTIC-inicia-proyecto-para-formar-100-000-colombianos-en-lenguaje-de-programacion

[8] BENNEDSEN, Jens y CASPERSEN, Michael E. Failure Rates in Introductory Programming. En: SIGCSE Bull [en línea]. Nueva York: Association for Computing Machinery, junio de 2007. vol.39, nro. 2. p. 32–36. [Consultado: 29 de septiembre de 2020]. Disponible en https://www.researchgate.net/publication/220613179. ISSN: 0097-8418.

[9] Ibíd., p. 102.

3. PREGUNTA PROBLEMA

¿Puede un software educativo apoyar el proceso de enseñanza-aprendizaje en algoritmos estructurados?

4. JUSTIFICACIÓN

Debido a estos altos índices de fracaso en cursos introductorios a la programación, se han realizado investigaciones acerca del tema que puedan identificar las causas de este. En un estudio de Fuentes y Moo[10] presentado en la revista de la Asociación Colombiana de Facultades de Ingeniería, identificó que las causas de la problemática recaen en el desconocimiento de estructuras lógicas, la dificultad de codificación de las estrategias, la memorización de rutinas, la abstracción de problemas, el desconocimiento de la sintaxis, desconocimiento de las herramientas del entorno y comúnmente la desmotivación del estudiante. Comprendiendo este último factor como la perspectiva que tiene el alumno ante el problema propuesto, la cual suele ser inalcanzable su solución; esto mismo resulta relevante no solo en la introducción a la programación sino en un entorno académico general. Por otro lado, en una investigación de Muñoz *et al* [11] expuesta en el Congreso Internacional de Informática Educativa en Chile determinó que cerca del 73% de los estudiantes involucrados en su estudio tienen problemas a la hora de encontrar errores, además de coincidir con variables expuestas en la investigación de Fuentes y Moo.

Así mismo, estos estudios encuentran aspectos que son mejor aceptados por los estudiantes a la hora de aprender a programar. Entre ellos encontramos una aceptación por los medios digitales como también las guías de ejercicio ya que desde la práctica el estudiante puede comprender conceptos dinámicos en entornos dinámicos. Estos estudios también proponen el aprendizaje como una actividad autónoma, la cual requiere de un proceso propio del estudiante para comprender los conceptos y entender sus errores. Durante la investigación de Muñoz[12], se evidenció una relevancia por estos procesos que usualmente son llevados a cabo en papel. Todos estos aspectos fueron tenidos en cuenta para la elaboración de la presente propuesta, los cuales serán explicados en detalle en el Marco Teórico del documento.

Actualmente existen herramientas con el propósito de apoyar el proceso de aprendizaje de los estudiantes universitarios, ejemplos de estas son PseInt

---

[10] FUENTES, Jorge y MOO, Melquizedec. Dificultades de aprender a programar. En: Revista Educación en Ingeniería [en línea]. Colombia: Asociación Colombiana de Facultades de Ingeniería, julio de 2017. vol. 12, nro. 24. p. 76-82. [Consultado: 29 de septiembre de 2020]. Disponible en https://doi.org/10.26507/rei.v12n24.728

[11] MUÑOZ, Roberto et al. TALLER INTERNACIONAL DE SOFTWARE EDUCATIVO [en línea]. En: Memorias del XVII Congreso Internacional de Informática Educativa, Nuevas Ideas en Informátíca Educativa. (Santiago, Chile). Determinando las dificultades en el aprendizaje de la primera asignatura de programación en estudiantes de ingeniería civil informática. Santiago: 2012. p. 120-126. [Consultado: 29 de septiembre de 2020]. Disponible en: http://www.tise.cl/volumen8/TISE2012/17.pdf

[12] Ibíd., p. 122.

(Carrizo[13]), RAPTOR, Visual Logic (Cooper[14]), SICAS (Gomes[15] y Marcelino[16]), FreeDfd (Del Prado[17]). Las cuales proponen de una manera didáctica la introducción de la programación por medio de pseudocódigo y (principalmente) diagramas de flujo, que apoyen el diseño de estrategias para solución de problemas. Por otro lado se encuentra JkarelTheRobot (Buck[18]), ProBot (Moreno[19]) y Laberithm (Satorre[20]) las cuales por medio de un juego inmersivo tratan de apoyar conceptos de las primitivas de programación situando eventos imaginarios para un personaje. Otra

---

[13] CARRIZO, Rosa; CORSO, Lorena y OLMEDO, Adriana. Congreso Argentino de Ingeniería Industrial [en línea]. En : VIIIº Congreso Argentino de Ingeniería Industrial. (12-13, noviembre. Córdoba, Argentina). Aplicación de una herramienta basada en Software Libre para la enseñanza de Algoritmos y Lógica de Programación. Córdoba. 2015. p. 1-9. [Consultado: 1 de noviembre de 2020]. Disponible en: http://www.edutecne.utn.edu.ar/coini_2015/trabajos/F030_COINI2015.pdf

[14] COOPER, Gina. World Congress in Computer Science, Computer Engineering, & Applied Computing [en línea]. En: Proceedings of The 2014 World Congress in Computer Science, Computer Engineering, and Applied Computing. (21-24, julio. San Diego, Estados Unidos). Using Visual Logic with Pseudocode to Teach an Introductory Programming Course. San Diego. 2014. p. 58-63. [Consultado: 1 de noviembre de 2020]. Disponible en: http://worldcomp-proceedings.com/proc/p2014/FEC2198.pdf

[15] GOMES, Anabela y MENDES, Antonio. Conference: Computers and Education [en línea]. En: Towards an Interconnected Society. (21, enero). SICAS: Interactive system for algorithm development and simulation. 2015. p. 1-9. [Consultado: 1 de noviembre de 2020]. Disponible en: https://www.researchgate.net/publication/228328489

[16] MARCELINO, María et al. 4th International Conference on Computer Systems and Technologies [en línea]. En: Educational Aspects of Computer Systems and Technologies (e-Learning). (17-18 junio. Rousse, Bulgaria). Using a computer-based interactive system for the development of basic algorithmic and programming skills. 2004. p. IV.8-(1-6). [Consultado: 1 de noviembre de 2020]. Disponible en: https://www.researchgate.net/publication/228328510_Using_a_computer-based_interactive_system_for_the_development_of_basic_algorithmic_and_programming_skills

[17] DEL PRADO, Ana y LAMAS, Nancy. Alternativas para la enseñanza de pseudocódigo y diagrama de flujo. En: Revista Electrónica Iberoamericana de Educación en Ciencias y Tecnología [en línea]. Catamarca: Universidad Nacional de Catamarca, diciembre de 2014. vol. 5, nro. 3. p. 102-113. [Consultado: 1 de noviembre de 2020]. Disponible en http://www.exactas.unca.edu.ar/riecyt/VOL%205%20NUM%203/F%20%20SI%203%2014%20Trabajo%20Completo%20Fundamentos.pdf, ISSN: 1852-852X

[18] BUCK, Duane y STUCKI, David. JKarelRobot: A Case Study in Supporting Levels of Cognitive Development in the Computer Science Curriculum. En: SIGCSE Bull [en línea]. Nueva York: Association for Computing Machinery, marzo de 2001. vol. 33, nro. 1. p. 16–20. [Consultado: 1 de noviembre de 2020]. Disponible en https://www.researchgate.net/publication/221536705. ISSN: 0097-8418.

[19] MORENO, Julian y MONTAÑO, Edgar. TALLER INTERNACIONAL DE SOFTWARE EDUCATIVO [en línea]. En: XIV TALLER INTERNACIONAL DE SOFTWARE EDUCATIVO, TISE ´09. (1-3, diciembre. Santiago de Chile, Chile). ProBot: Juego para el aprendizaje de lógica de programación. Santiago. 2009. p. 1-7. [Consultado: 1 de octubre de 2020]. Disponible en: http://www.tise.cl/2009/tise_2009/pdf/1.pdf

[20] SATORRE, Rosana; LLORENS, Faraón y PUCHOL, Juan. Enseñar Programación en las Ingenierías Informáticas. En: II Jornadas Nacionales de Innovación en las Enseñanzas de las Ingenierías: resúmenes de comunicaciones [en línea]. Alicante: Universidad de Alicante, 1996. vol. nro. 2. p. 840-847. [Consultado: 1 de noviembre de 2020]. Disponible en http://www.dccia.ua.es/~faraon/docs/programacion.pdf

herramienta es ABEA la cual como lo describe Arellano[21] es un software desarrollado bajo las 4 etapas heurísticas de Polya[22] el cual únicamente incorpora diagramas de flujo como recurso para el diseño de la estrategia. Pero ninguna de estas hace énfasis en la detección de errores lógicos o la incorporación de conceptos como la indentación (a excepción de PSeInt) en beneficio de la comprensión del código.

Esta propuesta surge debido a que en el curso de Introducción a la Programación de la Universidad Minuto de Dios, toda su metodología debe ser llevada a cabo por medios estáticos. Lo cual genera una gran dificultad a la hora de enseñar-aprender por medio del entorno remoto actual (causado por las medidas de bioseguridad ante la pandemia COVID-19), donde estos medios estáticos (y físicos) no resultan prácticos a la hora de desarrollar los conceptos. Es por esto que se plantea esta investigación y propuesta la cual apoyará el proceso de aprendizaje-enseñanza, especialmente en este entorno. Asimismo, va principalmente dirigida al público que no sabe programar, es por esto que sus recursos son básicos y sus conceptos se trabajará únicamente desde las primitivas de la programación.

---

[21] ARELLANO, Jesús et al. Software para la enseñanza-aprendizaje de algoritmos estructurados. En: Revista Iberoamericana de Educación en Tecnología y Tecnología en Educación N°8 [en línea]. La Plata: Universidad Nacional de La Plata, diciembre de 2012. vol. nro. 8. p. 23-33. [Consultado: 29 de septiembre de 2020]. Disponible en http://sedici.unlp.edu.ar/bitstream/handle/10915/25531/Documento_completo.pdf?sequence=1&isAllowed=y. ISSN 1850-9959.

[22] POLYA, George. Cómo plantear y resolver problemas. 2 ed. México: Trillas, 1989. 28 p. [Consultado el 20 de abril de 2021]. Disponible en: https://d1wqtxts1xzle7.cloudfront.net/34996114/Polya_-_Como_plantear_y_resolver_problemas.pdf?1412448720=&response-content-disposition=inline%3B+filename%3DPolya_Como_plantear_y_resolver_problemas.pdf&Expires=1619015790&Signature=cjnudU94TRKoD9YsLcpYA9UZYY~Y61rDOKYoURCP6KBWPtmM1TT2lj5wKP8iqn9E6PneiEUeaEv60hhNzzHmjIcZ0UfTwz33adaV7ZDsk9PQmXg-dqcelX2D4M5hkKknM526nqn9ceFdyrgSOyRkhtatFOg-vcjvuTrWzbTD775zafT-xD14EpVwBQ~nHovVYqyKZJoCYQk~gktitCrrupKij5Ulrt0iB5B3ENTiDiCBqg9a2miA0jee5L5CDW3wcaE0Fag8ubpQP2-aHg4PtB4wpys-AZkwbEV0XZqwupj5K8kRKPyBvWybG5~sFtQx0nRnzawPViUVTI3WM7guCQ__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA. ISBN 968-24-0064-3

5. ALCANCE

El alcance de esta propuesta se limita al uso de primitivas de programación (asignación, lectura, escritura, estructuras condicionales, estructuras cíclicas, anidamiento). Haciendo uso de estos para el desarrollo de algoritmos estructurados pertenecientes al paradigma de la programación orientada a objetos, siendo esta última más compleja en conceptos abstractos (herencia simple o múltiple, polimorfismo, etc). Estos conceptos anteriormente mencionados no se llevarán a cabo. Por lo tanto, la complejidad de los algoritmos por los cuales se desarrolló esta propuesta son de carácter simple o sencillo, sin ser extensos mayormente haciendo uso únicamente de un método principal sin invocar a secundarios. Más allá de esto, sus alcances estarán restringidos bajo las funcionalidades propuestas.

El entregable de esta propuesta se comprende en 4 partes esenciales: área de inserción de pseudocódigo, registro de cambios de variables (automático), registro de cambios de variables (manual) y consola. Estas áreas serán de interés para el desarrollo de los requerimientos funcionales del aplicativo, entre los cuales se perciben:

- Inserción y remoción de pseudocódigo por medio del sistema de generación de bloques según las primitivas de programación incorporando indentación
- Interpretación de pseudocódigo y ejecución del mismo
- Consola de entradas y salidas
- Realización de prueba de escritorio de manera automática
- Área de realización de prueba de escritorio de manera manual
- Comparación entre las pruebas de escritorio e identificación de discrepancias

Finalmente, entre los requerimientos no funcionales está el desarrollo de este software en lenguaje Java, su presentación debe ser standalone y la gestión de la información se llevará a cabo por medio de archivos. Siendo así una delimitación de apoyar el proceso de aprendizaje del estudiante en algoritmos estructurados por medio de la construcción un software educativo incorporando las primitivas ya mencionadas.

Entre otras características a continuación se detalla información que compone el pseudocódigo. El pseudocódigo al ser independiente es posible delimitarlo a 4 tipos de datos (entero, decimal, texto, lógico) con el propósito de acercar al estudiante al contexto de tipación en los lenguajes. Con base a estos tipos de datos se excluyeron vectores o matrices de los mismos. Los operadores básicos involucrados en esta propuesta son los siguientes símbolos ||, &&, ==, !=, >, <, <=, >=, !, +, -, *, /, %. Los

cuales en el mismo orden representan las operaciones lógicas *ó*, *y*, *igual que*, *diferente que*, *mayor que*, *menor que*, *menor o igual que*, *mayor o igual que* y *negación*. Mientras que los operadores aritméticos representan (en el orden restante) las operaciones de *suma*, *resta*, *multiplicación*, *división*, *módulo*.

En cuanto a las estructuras cíclicas únicamente se trabajarán de 2 tipos, la iteración definida por un número entero positivo y con paso 1 (*Repetir un número definido de veces*), y la repetitiva *Mientras que*. Por otro lado, en los bloques condicionales se trabajará la condicional simple *Si* y la condicional compuesta *Si-sino*. Asimismo, este alcance no incluye la escritura ni el llamado de métodos, no incluye la recursión, y tampoco incluye el manejo de librerías externas.

Como se mencionó anteriormente se debe incluir la realización de la prueba de escritorio para la comprensión de errores lógicos y la notificación de errores léxicos, sintácticos y semánticos.

6. OBJETIVOS

6.1 OBJETIVO GENERAL

- Construir un prototipo de software educativo para apoyar el procesos de aprendizaje de algoritmos estructurados en entornos educativos desde las primitivas básicas de programación.

6.2 OBJETIVOS ESPECÍFICOS

- Establecer el levantamiento de los requerimientos funcionales como no funcionales del prototipo que permitan la identificación de las necesidades en el proceso de aprendizaje en los cursos introductorios a la programación del paradigma estructurado.
- Diseñar y desarrollar un prototipo de software stand-alone por medio de la metodología ágil SCRUM en función a los requerimientos previamente establecidos.
- Validar el prototipo por medio de pruebas funcionales y un grupo focal de estudiantes universitarios en ingeniería en sistemas.

7. MARCO REFERENCIAL

7.1 MARCO CONCEPTUAL

7.1.1 Java.

En principio Deitel[23] define java como un poderoso lenguaje de programación apropiado para el desarrollo de aplicativos con tamaño considerable. Para la fecha del libro, java es uno de los lenguajes de programación más utilizados en el mundo y esto se debe al auge que ha tenido el paradigma orientado a objetos el cual permite el desarrollo de aplicativos elaborados. Así mismo, Java es un lenguaje portable lo cual permite la ejecución de programas en múltiples plataformas. Este proyecto ha crecido en múltiples direcciones lo cual permite que su información no sea centralizada. Finalmente, este es un lenguaje altamente tipado y sensible a mayúsculas en sus identificadores.

7.1.2 Intérprete.

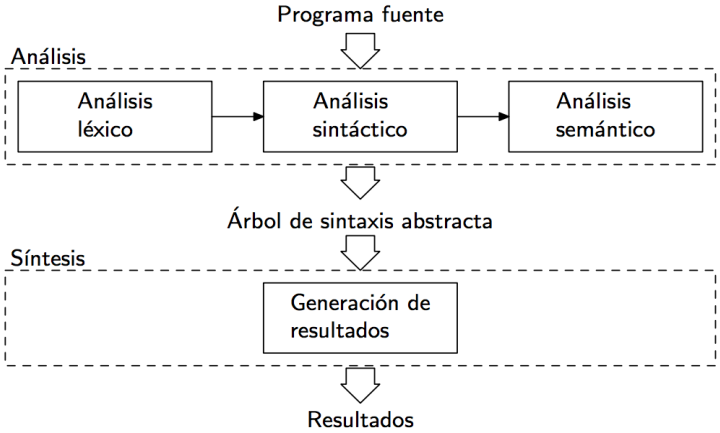
Para el autor Alfonseca[24], un intérprete es un procesador del lenguaje que analiza un programa escrito (en alto nivel) y de ser correcto lo ejecuta directamente en el lenguaje máquina del intérprete. Cada vez que desea ejecutar el programa que desea, debe interpretarlo de nuevo. Estos intérpretes-compiladores se ejecutan en 2 fases diferentes: la compilación o introducción del programa y la interpretación o ejecución del programa. En la primera etapa se compila y se traduce a un formato intermedio el cual suele ser propio del intérprete, este formato es el resultado del análisis morfológico o sintáctico. En la segunda etapa se interpreta y se ejecuta el formato de la fase anterior, esta se suele realizar *n* cantidad de veces. En la *Figura 1* se muestra la estructura de un intérprete.

---

[23] DEITEL, Paul y DEITEL, Harvey. Cómo programar en Java. 7 ed. Naucalpan de Juárez: PEARSON PRENTICE HALL, 2008. XXVIII p. [Consultado el 20 de abril de 2021]. Disponible en: http://www.mfbarcell.es/docencia_uned/fund_inf_ing/libros/-%20Como%20Programar%20en%20Java%20Deitel.pdf. ISBN 978-970-26-1190-5

[24] ALFONSECA MORENO, Manuel et al. Intérpretes. En: Compiladores e intérpretes: teoría y práctica [en línea]. PEARSON PRENTICE HALL, 2006. [Consultado el 20 de abril de 2021]. Capítulo 8. 317 p. Disponible en https://d1wqtxts1xzle7.cloudfront.net/51585857/Compiladores-e-interpretes-teoria-y-practica.pdf?1485977162=&response-content-disposition=inline%3B+filename%3DCompiladores_e_interpretes_teoria_y_prac.pdf&Expires=1619016405&Signature=X0MqgGKuxe65w3xbiczHClbBiH9051cOn9xDh8qjZzzHTU6mfJkcbn4dPo1hrCrbWkgCq~w799geheJmWB3JneM3hUnNwNWI0LrzDZgvgZiX1CkHl-j4Mx2DSybOW~NHwlFGpDIXGAUAVpyQp14zvcrQp12XnXVuVQXqWg9xsnpv8~hWVeEBqW-h5W3TB-1yFFW~4-9MM3TqJuOxctDLDj9BnyjNPJTWVrXVnBuOsdwSbtMKSsOzUNWClvvLsTfk2OvLsdlZTwqFdr2~Oi0~lUBUO04J90dNOfuhJvZ7-QbPHqhjCxcmcwv8xxQwvAZGBwK2MgXxYGnTMv~iMY5a-Q__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA

Figura 1. Estructura de un intérprete



Fuente: Universitat Jaume I. 4º Ingeniería Informática II26 procesadores de lenguaje: Estructura de los compiladores e intérpretes

7.1.3 Compilador.

Para el autor Alfonseca[25], un compilador es un procesador del lenguaje que analiza un programa escrito y de ser correcto lo ejecuta directamente en lenguaje máquina en que está ejecutando el intérprete. La diferencia con el concepto anterior recae en su composición (ya que debe gestionar la memoria) como también en su función debido a que luego de ser transcrito el código fuente en el formato especial (lenguaje objeto), este se ejecuta sin contemplar cambios. Por lo tanto, no se puede gestionar cada instrucción en el momento de su ejecución.

7.1.4 Analizador Léxico/ Morfológico.

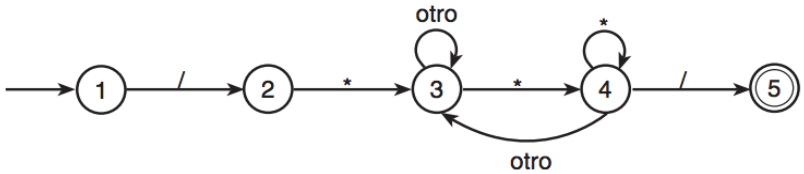
Para Alfonseca[26], el analizador morfológico (también conocido como analizador léxico) es el encargado de dividir el código en unidades sintácticas (tokens) por

[25] Ibíd., p. 321.

[26] ALFONSECA MORENO, Manuel et al. Análisis morfológico. En: Compiladores e intérpretes: teoría y práctica [en línea]. PEARSON PRENTICE HALL, 2006. [Consultado el 20 de abril de 2021]. Capítulo 3. 65 p. Disponible en https://d1wqtxts1xzle7.cloudfront.net/51585857/Compiladores-e-interpretes-teoria-y-practica.pdf?1485977162=&response-content-disposition=inline%3B+filename%3DCompiladores_e_interpretes_teoria_y_prac.pdf&Expires=1619016405&Signature=X0MqgGKuxe65w3xbiczHClbBiH9051cOn9xDh8qjZzzHTU6mfJkcbn4dPo1hrCrbWkgCq~w799geheJmWB3JneM3hUnNwNWI0LrzDZgvgZiX1CkHl-j4Mx2DSybOW~NHwlFGpDIXGAUAVpyQp14zvcrQp12XnXVuVQXqWg9xsnpv8~hWVeEBqW-h5W3TB-1yFFW~4-9MM3TqJuOxctDLDj9BnyjNPJTWVrXVnBuOsdwSbtMKSsOzUNWCIvvLsTfk2OvLsdlZTwqFdr2~Oi

medio de las palabras clave (keywords). La división en unidades sintácticas recae en expresiones regulares las cuales caracterizan al lenguaje, estas son cadenas de caracteres que se ajustan a un patrón, lo cual permite limpiar el contenido denotando únicamente la parte funcional. De aquí se derivan diversas operaciones como concatenación, unión y cierre. Asimismo el analizador está compuesto por autómatas finitos no deterministas y deterministas, donde los no deterministas pueden incurrir en varios estados mientras el determinista no. También existen otros tipos de autómatas que permiten apoyar el desempeño en el análisis. A continuación en la *Figura 2* se muestra un ejemplo de un autómata finito para un comentario tipo C.

Figura 2. Autómata finito para comentarios tipo C.



Fuente: ALFONSECA MORENO, Manuel et al. Compiladores e intérpretes: teoría y práctica

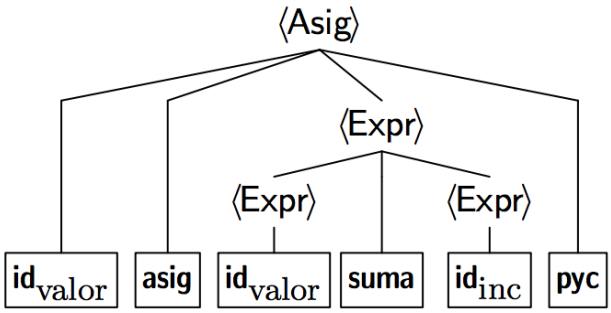
7.1.5 Analizador Sintáctico.

Para un artículo de la Universidad de Jaume I[27] este elemento es el encargado de construir un árbol sintáctico el cual parte de los *tokens* encontrados en la fase anterior (analizador morfológico). Esta construcción parte de las estructuras presentes en el código fuente, las cuales permiten evidenciar la función en conjunto de los *tokens* con algún propósito. En la *Figura 3* se evidencia un ejemplo de árbol sintáctico para una asignación.

Figura 3. Arbol sintactico para una asignación

0~IUBUO04J90dNOfuhJvZ7-QbPHqhjCxcmcwv8xxQwvAZGBwK2MgXxYGnTMv~iMY5a-Q__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA

[27] UNIVERSITAT JAUME I. Procesadores de Lenguaje: Estructura de los compiladores e interpretes. En: Universidad Jaime I de Castellón [en línea]. Castellón de la Plana: Universidad Jaime I, 2010-2011. vol. , nro. p. 1-9. [Consultado: 1 de noviembre de 2020]. Disponible en http://repositori.uji.es/xmlui/bitstream/handle/10234/22656/II26_estructura_compiladores.pdf?sequence=1



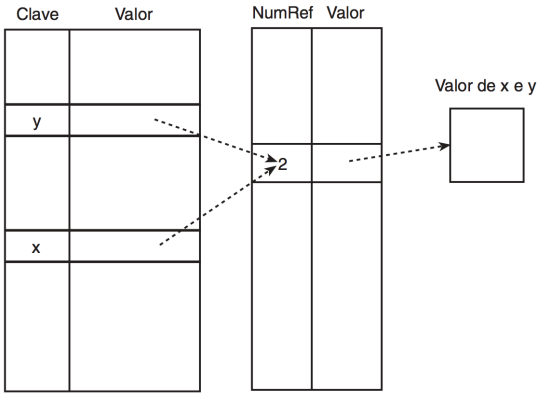
Fuente: Universitat Jaume I. 4º Ingeniería Informática II26 procesadores de lenguaje: Estructura de los compiladores e intérpretes

7.1.6 Tabla de Símbolos.

Para Alfonseca[28] la tabla de símbolos es un medio encargado de todos los aspectos del código fuente designados con un identificador (variables, funciones, constantes, *keywords*). Estos valores deben ser tomados en cuenta durante su compilación para poder denotar sus cambios o definiciones. Un gran factor involucrado en este aspecto es su rendimiento, debido a la búsqueda de estos valores. Es por esto que se sugiere usar las tablas *hash* como medio de almacenamiento para el diccionario de identificadores, estas tablas almacenan la clave y consigo la posición del dato (su método se llama *hash* o dispersión). En la búsqueda del identificador se parte de la clave y dentro de la función *hash* (el desempeño de su búsqueda recae en está) se calcula su posición, permitiendo agilidad a la hora de determinar la información del dato. Esta función *hash* deberá poseer un calculo cuya posición ofrezca una distribución uniforme dentro de la tabla. Debido a esto, es probable que existan colisiones y para ello se plantean diferentes alternativas. La alternativa más útil sería el *hash* con encadenamiento la cual facilita la inserción de un dato dada una colisión. En la *Figura 4* se evidencia una representación visual de una tabla de símbolos.

[28] ALFONSECA MORENO, Manuel et al. Tabla de símbolos. En: Compiladores e intérpretes: teoría y práctica [en línea]. PEARSON PRENTICE HALL, 2006. [Consultado el 20 de abril de 2021]. Capítulo 2. 33 p. Disponible en https://d1wqtxts1xzle7.cloudfront.net/51585857/Compiladores-e-interpretes-teoria-y-practica.pdf?1485977162=&response-content-disposition=inline%3B+filename%3DCompiladores_e_interpretes_teoria_y_prac.pdf&Expires=1619016405&Signature=X0MqgGKuxe65w3xbiczHClbBiH9051cOn9xDh8qjZzzHTU6mfJkcbn4dPo1hrCrbWkgCq~w799geheJmWB3JneM3hUnNwNWI0LrzDZgvgZiX1CkHl-j4Mx2DSybOW~NHwlFGpDIXGAUAVpyQp14zvcrQp12XnXVuVQXqWg9xsnpv8~hWVeEBqW-h5W3TB-1yFFW~4-9MM3TqJuOxctDLDj9BnyjNPJTWVrXVnBuOsdwSbtMKSsOzUNWCIvvLsTfk2OvLsdlZTwqFdr2~Oi0~IUBUO04J90dNOfuhJvZ7-QbPHqhjCxcmcwv8xxQwvAZGBwK2MgXxYGnTMv~iMY5a-Q__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA

Figura 4. Uso de tabla de símbolos en conjunto con tabla de referencias



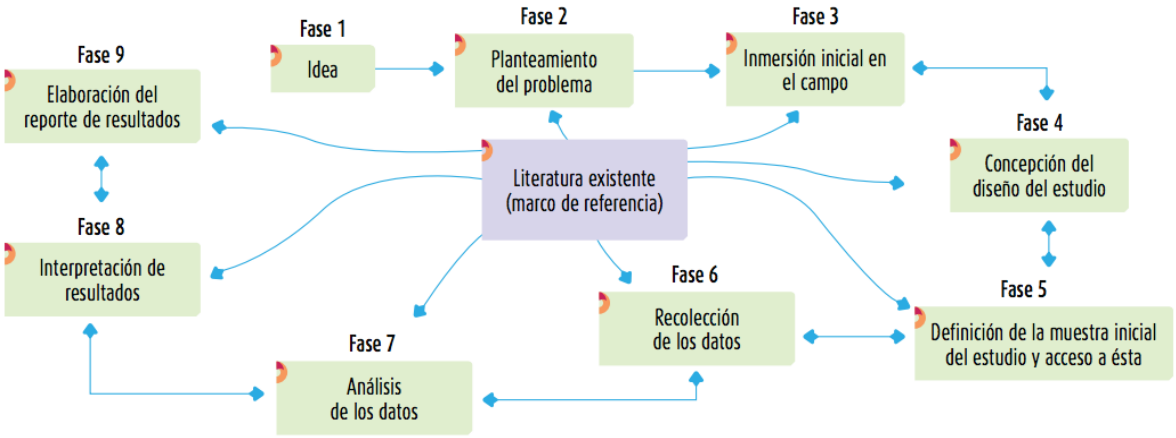
Fuente: ALFONSECA MORENO, Manuel et al. Compiladores e intérpretes: teoría y práctica

7.1.7 Investigación Cualitativa.

Para Sampieri[29] la determinación de hipótesis y preguntas de este tipo de investigación es cambiante en el proceso, con frecuencia las actividades permiten la identificación de estas así como sus respuestas. Por otro lado es inductiva, lo cual pretende ir de lo general a lo particular. El proceso de esta investigación es circular y flexible a cambios. La revisión bibliográfica puede complementar la búsqueda y apoyar el planteamiento del problema, incluso la elaboración de resultados. Se puede saltar entre etapas si es necesario para modificar variables en el estudio. Sus herramientas principales son la observación no estructurada, las entrevistas abiertas, la revisión de documentos, la discusión de grupo, etc. En el diagrama de la *Figura 5* se explica su proceso.

Figura 5. Proceso cualitativo

[29] SAMPIERI, Roberto; FERNÁNDEZ, Carlos y BAPTISTA, María del Pilar. Definiciones de los enfoques cuantitativo y cualitativo, sus similitudes y diferencias. En: Metodología de la Investigación [en línea]. McGraw-Hill Interamericana, 2014. [Consultado el 20 de abril de 2021]. Capítulo 1. 8 p. Disponible en https://www.uca.ac.cr/wp-content/uploads/2017/10/Investigacion.pdf



Fuente: SAMPIERI, Roberto y otros. Metodología de la Investigación

7.1.8 Investigación Cuantitativa.

Para Sampieri[30] esta investigación es secuencial y probatoria, no se pueden saltar pasos. Se deben derivar objetivos y preguntas de investigación constituidos en una revisión literaria estableciendo hipótesis y determinando variables. Se traza un plan para probarlas y se miden en un contexto. Esta investigación es deductiva y buscan establecer un resultado para todos los casos posibles.

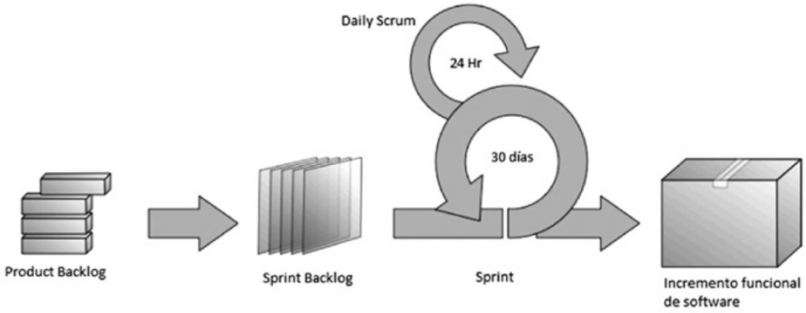
7.1.9 Metodología Ágil SCRUM.

Según Navarro[31] "La metodología Scrum para el desarrollo ágil de software es un marco de trabajo diseñado para lograr la colaboración eficaz de equipos en proyectos, que emplea un conjunto de reglas y artefactos y define roles que generan la estructura necesaria para su correcto funcionamiento". En la *Figura 6* se aprecia un esquema de esta metodología.

Figura 6. Esquema de la Metodológia Ágil SCRUM

[30] Ibíd., p. 18.

[31] NAVARRO, Andrés; FERNÁNDEZ, Juan y MORALES, Jonathan. Revisión de metodologías ágiles para el desarrollo de software. En: PROSPECTIVA [en línea]. Barranquilla: Universidad Autónoma del Caribe, junio-diciembre de 2013. vol. 11, nro. 2. p. 30-39. [Consultado el 1 de noviembre de 2020]. Disponible en https://www.redalyc.org/pdf/4962/496250736004.pdf. ISSN: 1692-8261.



Fuente: NAVARRO, Andrés; FERNÁNDEZ, Juan y MORALES, Jonathan. Revisión de metodologías ágiles para el desarrollo de software.

7.1.10 Pedagogía en cursos Introductorios a la programación.

En los cursos introductorios a la programación es común ver un modelo pedagógico enfocado en el paradigma de la programación estructurada, ejemplos de esto se evidencia en documentos por autores como Redondo[32], Satorre[33] y Cooper[34]. Si bien este paradigma resulta hoy en día implícito en el paradigma Orientado a Objetos, es de gran utilidad enseñar la programación estructurada de manera explícita con conceptos básicos inmersos en la programación como lo son las primitivas básicas. Esto también se debe a que el método de enseñanza es constructivo en donde se suele empezar con el detalle para crear un modelo mental amplio y pueda entender en completitud conceptos más complejos. En el estudio de Kereki[35] se comprueba la comprensión en detalle para el desarrollo de algoritmos desde el paradigma estructurado.

7.1.11 Indentación.

La indentación, también conocida como sangría es un elemento léxico el cual permite en programación entender el encapsulamiento definido en los bloques del código. Este elemento como lo menciona como lo menciona el autor Clifton[36], facilita

---

[32] REDONDO, Miguel; MENDES, Antonio; MARCELINO, María y BRAVO Crescencio. TALLER INTERNACIONAL DE SOFTWARE EDUCATIVO [en línea]. En: 8vo Taller Internacional de Software Educativo. (24-26, noviembre. Santiago, Chile). Planificación colaborativa del diseño para el aprendizaje de la programación. Santiago de Chile. 2003. p. 1-11. [Consultado el 1 de noviembre de 2020]. Disponible en: http://www.tise.cl/2010/archivos/tise2003/papers/planificacion_colaborativa.pdf

[33] SATORRE, Op. cit., p. 843

[34] COOPER, Op. cit., p. 60

[35] KEREKI, Ines y SALVETTO, Fernando. Enseñanza Inicial de la Programación: Comparación entre Programación Estructurada y Programación Orientada a Objetos. En: Universidad ORT Uruguay [en línea]. Montevideo: Universidad ORT Uruguay. vol. , nro. p. 1-8. [Consultado el 1 de noviembre de 2020]. Disponible en https://fi.ort.edu.uy/innovaportal/file/5553/2/comparacionpe_poo.pdf

[36] CLIFTON, Mitchell. A technique for making structured programs more readable. En: SIGPLAN Not [en línea]. Nueva York: Association for Computing Machinery, abril de 1978. vol.13 nro. 4. p. 58–63.

la lectura del código y por tanto su comprensión, su modificación y mantenibilidad. Asimismo, suele considerarse como buena práctica la cual puede contribuir en la detección de errores lógicos.

7.1.12 Heurística.

El autor Santos[37], plantea una heurística como un instrumento metodológico que apoya el desarrollo de conocimiento cuyo principal enfoque puede recaer en la resolución de problemas. Esta resolución puede fomentarse desde perspectiva cualitativa de quien vive la experiencia y encuentra el método para resolver un problema dado. Así mismo, como lo menciona Polya[38] la heurística suele plantearse de manera general, estudiando métodos que puedan resolver problemas de cualquier tipo.

7.2 MARCO TEÓRICO

Como lo define Daros[39], un problema se caracteriza por generar dificultad ya sea de carácter psicológico, lógico o real. Donde el psicológico representa una dificultad desde la perspectiva del sujeto pues este carece de presaberes lo suficientemente necesarios para entender el problema. Siendo este carácter de dificultad el apropiado para la presente investigación ya que según la investigación cualitativa de Sampieri *et al*[40] se debe partir del fenómeno de interés; en este caso 'la dificultad de aprendizaje en cursos introductorios a la programación'. Fenómeno el cual es únicamente perceptible para quien posee dicha dificultad en el contexto ya mencionado. Daros[41] también menciona que debe ser necesario explicitar una hipótesis que ayude a comprender el problema o ayude a la solución del mismo en cierto grado. Para el presente proyecto la hipótesis planteada 'TestDesk apoya el proceso de aprendizaje en algoritmos estructurados desde las estructuras lógicas y primitivas básicas de la programación (siguiendo el alcance definido) siempre y

---

[Consultado el 1 de noviembre de 2020]. Disponible en: https://dl.acm.org/doi/abs/10.1145/953411.953415. ISSN: 0362-1340.

[37] SANTOS Raúl; CHUC Fernando; CADENA Selene y SILVA Hilda. EL MÉTODO HEURÍSTICO DE POLYA EN UN ESCENARIO DE INVESTIGACIÓN: APLICACIÓN EN UN CASO ESPECÍFICO. En: I. C. Investig@ción [en línea]. Campeche: Instituto Campechano, junio-noviembre de 2018. vol. nro. 14. p. 9-21. [Consultado el 1 de noviembre de 2020]. Disponible en http://instcamp.edu.mx/wp-content/uploads/2018/11/Ano2018No14_9_21.pdf

[38] POLYA, Op. cit., p. 22

[39] DAROS, William. ¿Qué es un marco teórico?. En: Enfoques [en línea]. Bolivia: Universidad Adventista del Plata, 2001. vol. 14, nro. 1-2. p. 73-112. [Consultado el 12 de abril de 2021]. Disponible en http://www.publicaciones.uap.edu.ar/index.php/revistaenfoques/article/view/348/338. ISSN: 1669-2721

[40] SAMPIERI, Op. cit., p. 4.

[41] DAROS, Op. cit., p. 78.

cuando existan los pre saberes necesarios en el estudiante' trata de responder a la pregunta problema '¿Puede un software educativo apoyar el proceso de enseñanza-aprendizaje en algoritmos estructurados?'. Dicha hipótesis supone lo que debería suceder en la búsqueda de un grado de solución al problema.

Daros[42] explica que para poder constatar el valor de la comprobación de la hipótesis se debe establecer un diseño metodológico, donde a groso modo se establecen 4 fases en el proceso de investigación (problema, marco teórico, diseño metodológico y realización de lo planificado). En vista de que en capítulos anteriores se ha descrito el problema y el diseño metodológico a trabajar, únicamente en el presente capítulo se describe el marco teórico entorno a la hipótesis planteada.

Para Daros[43], el marco teórico pretende describir los problemas en la investigación en referencia a la teoría o hipótesis planteada, abstraer el problema y comprenderlo en sus partes, dar sentido a los hechos y dar orden a los mismos, y por último ser el eje integrador de la investigación. Con lo cual, los subcapítulos posteriores a esta introducción tienen el propósito de dar función a lo descrito anteriormente.

7.2.1 Procesos de aprendizaje en la introducción a la programación.

En este subcapítulo se pretende detallar, más a fondo, el proceso de aprendizaje en cursos introductorios a la programación. Denotando modelos contemporáneos y tradicionales de cómo se ha venido elaborando este proceso en las instituciones educativas o contextos análogos a este. Con el fin de encuadrar, ubicar y centrar al lector acerca de la manera en cómo este evento se ha venido desarrollando y de esta manera identificar las dificultades y sus posibles variables causantes.

Uno de los cuestionamientos, antes de introducir a los modelos educativos, es la incógnita sobre el temario adecuado para la introducción a la programación, esto a razón de que son diversas las instituciones que introducen este curso y por tanto sus temarios pueden variar en gran y menor medida. García[44] afirma que incluso en el auge de la programación orientada a objetos es común ver en el mundo académico la introducción basada en el paradigma estructurado utilizando Pascal,

---

[42] DAROS, Op. cit., p. 75.
[43] DAROS, Op. cit., p. 81.
[44] GARCÍA, Jesus. Jornadas de Enseñanza universitaria de la Informática [en línea]. En: Actas de las VII Jornadas de Enseñanza universitaria de la Informática. (16-18, julio. Palma de Mallorca, España). ¿Es conveniente la orientación a objetos en un primer curso de programación?. En: Universidad de les Illes Balear. 2001. p. 291-298. [Consultado el 12 de abril de 2021]. Disponible en: http://bioinfo.uib.es/~joemiro/aenui/procJenui/ProcWeb/actas2001/gaesc43.pdf]

Modula o Ada. Aún mejor, en una investigación llevada a cabo por Schulte y Bennedsen[45] al encuestar diferentes instituciones de diferentes países (Estados Unidos, Dinamarca, Alemania y otros) se pudo determinar que ninguna de ellas excluye en sus temarios a la introducción a la programación conceptos trabajados en el paradigma estructurado.

Ahora bien, García[46] también argumenta desde la hipótesis del desconocimiento total para un estudiante novato cualquier concepto relacionado a la programación, por ello mismo es necesario dotar al estudiante con mecanismos inmersos en el diseño de soluciones en un marco imperativo como lo es el paradigma estructurado. Esto con el fin de crear un modelo mental que le permita fundamentar adecuadamente sus ideas a la hora de introducirse en el Paradigma Orientado a Objetos (POO). Así mismo, García[47] contraargumenta con el concepto donde el POO puede considerarse como un paradigma construido sobre la base del paradigma estructurado (PE) debido a que encapsula en su modelo operacional la asignación, el cambio de estado, las estructuras de control secuencial, condicional e iterativas como mecanismos de control en la ejecución. Es por esta razón por la cual múltiples académicos en los cursos introductorios a la programación se enfocan en primera instancia en la enseñanza-aprendizaje de conceptos netamente íntegros del PE. Por último, García[48] también propone dividir el curso en caso de iniciar con el POO para la introducción a la programación. De tal forma en que se da un enfoque del PE haciendo énfasis en sus estructuras básicas y posteriormente en la otra parte del curso explicar conceptos como objetos, herencia, clases, polimorfismo propios del POO.

Dicho esto, podemos encontrar modelos tradicionales tales como el de Astrachan[49] el cual, debido a la década en la que se desarrollaba, sus recursos eran netamente rudimentarios ya que en su mayoría de clases se desarrollaba un modelo presencial

---

[45] SCHULTE, Carsten y BENNEDSEN, Jens. International Computing Education Research Workshop [en línea]. En: ICER '06: Proceedings of the second international workshop on Computing education research. (septiembre. Nueva York, Estados Unidos). What do teachers teach in introductory programming?. Nueva York. 2006. p. 17-28. [Consultado el 12 de abril de 2021]. Disponible en: https://www.researchgate.net/publication/234772749_What_do_teachers_teach_in_introductory_programming

[46] GARCÍA, Op. cit., p. 292.

[47] GARCÍA, Op. cit., p. 293.

[48] GARCÍA, Op. cit., p. 294.

[49] ASTRACHAN, Owen. Concrete Teaching: Hooks and Props as Instructional Technology. En: SIGCSE Bull [en línea]. Nueva York: Association for Computing Machinery, septiembre de 1998. vol. 30 nro. 3. p. 21-24. [Consultado el 1 de noviembre de 2020]. Disponible en: https://users.cs.duke.edu/~ola/papers/propsfinal.pdf. ISSN: 0097-8418.

y utilizaban tableros, hojas y libros como material de apoyo. Así mismo, para poder disipar las dudas de los estudiantes se realizaron actividades lúdicas las cuales requerían la interacción de los estudiantes para comprender el comportamiento de los conceptos en un entorno dinámico, a esto se le atribuyen actividades utilizando frisbees, bloques de construcción didáctica y ganchos de goma (como se detalla en el anexo de Marco Histórico). Otro proceso pedagógico es el de Siegel[50], el cual pretende enseñar por medio de canciones (revisar anexo de Marco Histórico).

Debido a estos modelos tradicionales, a lo largo del tiempo se han venido creando recursos de apoyo que puedan facilitar el proceso de aprendizaje para el estudiante. Recursos tales como los Objetos de Aprendizaje (OA) u Objetos Virtuales de Aprendizajes (OVA), los cuales como lo explica Olmos[51] en el campo de la programación se podrían ver incluidos en la comprensión de estructuras algoritmicas. Para Kovalchick y Dawson[52] estos OAs y OVAs son un medio creado para dar apoyo en la educación el cual puede ser reutilizado múltiples veces. Siendo estos una buena opción a la hora de reinventar los procesos de aprendizaje mediante instrumentos tecnológicos. Así mismo, Olmos[53] *et al* afirman que estos OAs pueden ser enfocados al diseño de soluciones, competencia inmersa en la resolución de problemas, y de esta manera ser un medio de retroalimentación para el estudiante.

Entre los modelos medianamente contemporaneos, Xie, Li y Geng[54] denotan un proceso pedagógico bautizado como 'Blended Learning' (aprendizaje combinado)

---

[50] SIEGEL, Eric. Innovation and Technology in Computer Science Education [en línea]. En: ITiCSE '99: Proceedings of the 4th annual SIGCSE/SIGCUE ITiCSE conference on Innovation and technology in computer science education. (junio. Cracow, Polonia). Why Do Fools Fall Into Infinite Loops: Singing To Your Computer Science Class. Cracow. 1999. p. 167–170. [Consultado el 1 de noviembre de 2020]. Disponible en: http://www.predictionimpact.com/Siegel-educational-computer-songs.pdf

[51] OLMOS, Karla; MORALES, Cristal; ROJAS, Teresa y FERNÁNDEZ, Luis. Objetos de Aprendizaje Enfocados a la Resolución de Problemas para Facilitar la Enseñanza de la Programación. En: Universidad Autónoma de Ciudad Juárez, Instituto de Ingeniería y Tecnología [en línea]. Ciudad Juárez: Universidad Autónoma de Ciudad Juárez, 2010. vol. nro. p. 1-6. [Consultado el 12 de abril de 2021]. Disponible en https://www.academia.edu/1303813/Objetos_de_Aprendizaje_Enfocados_a_la_Resoluci%C3%B3n_de_Problemas_para_Facilitar_la_Ense%C3%B1anza_de_la_Programaci%C3%B3n

[52] KOVALSCHICK, Ann y DAWSON, Kara. Education and Technology. 1 ed. Santa Barbara, California; Denver, Colorado; Oxford, England: ABC-CLIO, 2003. 544p. ISBN 1-57607-351-3

[53] OLMOS, Op. cit., p. 3.

[54] XIE, Jianjun; LI, Junhong y GENG, Yi. International Conference on Computer Science and Software Engineering [en línea]. En: 2008 International Conference on Computer Science and Software Engineering. (12-14, diciembre. Wuhan, China). Blended Learning in Computer Hardware

el cual pretende mezclar y/o utilizar dos métodos de entrenamiento en paralelo. En dicho modelo se pueden encontrar combinaciones tales como combinado de instrucción en línea con acceso a un tutor, combinado de simulaciones con cursos estructurados, combinado de instrucción con trabajo de sesiones informales. El principal propósito de este modelo es crear un entorno integrado donde el estudiante pueda comprender el contenido, resolver problemas y validar sus ideas.

Entre las entrevistas llevadas a cabo con (Giovanni Fajardo Utria) profesor de la Universidad Minuto de Dios, se declararon modelos pedagógicos basados en mecanismos tradicionales. Esto debido a que la institución disponía para el curso introductorio a la programación únicamente recursos físicos como tableros, marcadores, hojas y libros. De este modo, las clases llevadas a cabo en esta institución recurrieron al desarrollo de algoritmos en papel, la ejecución de manera manual y la exposición de los conceptos de manera abstracta. Proceso el cual dificulta la compresión en completitud de los temas expuestos. No solo eso, sino que el impacto que tuvo fue determinante al tratar de incorporar la educación remota impuesta por la pandemia COVID-19 ya que sus recursos eran netamente físicos y no digitales.

7.2.2 Dificultades en el aprendizaje del paradigma estructurado.

Diferentes estudios académicos han tratado de denotar las dificultades relacionadas con la tasa de fracaso en el paradigma estructurado. Entendiendo el fracaso como la inhabilidad de comprender los conceptos expuestos en el curso y por tanto su abandono o pérdida del curso.

Por un lado para Connolly[55], una dificultad entre los estudiantes está el hecho de tener que trasladar los ejemplos llevados a cabo en el curso a escenarios análogos que posiblemente son latentes a un problema real. Es decir, una inhabilidad analitica a la hora de resolver problemas. De lo cual puede recaer en la confusión, falta de confianza y falta de motivación. Variables las cuales se detallarán en el subcapítulo posterior.

Course Education. 2008. p. 676-680. [Consultado el 12 de abril de 2021]. Disponible en: https://ieeexplore.ieee.org/document/4723051

[55] CONNOLLY, Thomas. Using Games-Based eLearning Technologies in Overcoming Difficulties in Teaching Information Systems. En: Journal of Information Technology Education [en línea]. Estados Unidos: Informing Science Institute, 2006. vol. 5, nro. 1. p. 459-476. [Consultado el 12 de abril de 2021]. Disponible en https://www.researchgate.net/publication/220590643_Using_Games-Based_eLearning_Technologiesin_Overcoming_Difficulties_in_Teaching_Information_Systems

Para Xie, Li y Geng[56] una de las dificultades más presentes a la hora de educar es hacer que sus estudiantes tomen el contenido del curso y lo involucren en su vida cotidiana. Por otro lado, Mathrani, Christian y Ponder-Sutton[57] denotaron dificultades a la hora de transferir conocimientos teóricos adquiridos al ejercicio práctico. Así mismo, para relacionar los contenidos del curso con el uso de la industria. También, la falta de interés en los contenidos del curso y dificultad para comprender temas conceptuales debido a la falta de habilidades analíticas y lógicas fue un elemento presente en su estudio.

Además, diferentes autores (de diferentes investigaciones) como por ejemplo Bennedsen y Schulte[58], Derus, Mohamad Ali[59], y Milne y Rowe[60] han recalcado la dificultad que existe en el estudiante para comprender qué es lo que ocurre con su programa en memoria. La creación de este modelo mental en el estudiante le permitirá sentar los fundamentos para las tareas más esenciales en la programación como lo es la detección y corrección de errores lógicos el diseño de algoritmos y la manipulación de datos tipados y no tipados. Siendo esta última una de las dificultades que han venido teniendo auge en lenguajes de programación débilmente tipados como Javascript y Python.

Edwards[61] afirma también en un estudio sobre las dificultades a la hora de enseñar la programación está la perspectiva equivocada de lo que representa la actividad de

---

[56] XIE, LI, GENG, Op. cit., p. 676.

[57] MATHRANI, Anuradha; CHRISTIAN, Shelly y PONDER-SUTTON, Agate. PlayIT:Game Based Learning Approach for Teaching Programming Concepts. En: Journal of Educational Technology & Society [en línea]. Taiwan: International Forum of Educational Technology & Society, julio de 2016. vol. 19, nro. 2. p. 5-17. [Consultado el 8 de marzo de 2021]. Disponible en: https://pdfs.semanticscholar.org/98ce/1c523d7617e2b64af12d87f3f1bb69e4f0ca.pdf

[58] SCHULTE, Op. cit., p. 17.

[59] DERUS, Siti y ALI, Mohamad. International Conference on Current Issues in Education [en línea]. En: 1st International Conference oo Current Issues in Education. (19-21, noviembre. Madrid, España). Difficulties in learning programming: Views of students. Madrid. 2012. p. 1-6. [Consultado el 12 de abril de 2021]. Disponible en: https://www.researchgate.net/publication/267338258_Difficulties_in_learning_programming_Views_of_students

[60] MILNE, Iain y ROWE, Glenn. Difficulties in Learning and Teaching Programming—Views of Students and Tutors. En: Education and Information Technologies. Estados Unidos: Education and Information Technologies, marzo 2002. vol. 7, nro. p. 55-66. [Consultado: 21 de abril de 2021]. Disponible en: https://doi.org/10.1023/A:1015362608943

[61] EDWARDS, Stephen. Special Interest Group on Computer Science Education [en línea]. En: Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education. (3-7, marzo. Virginia, Estados Unidos). Using software testing to move students from trial-and-error to reflection-in-action. Virginia. 2004. p. 1-5. [Consultado el 12 de abril de 2021]. Disponible en:

programar, lo cual puede verse como un impedimento a la hora de cambiar dicha perspectiva al explicar el curso. También afirma que otra apreciación errónea es la corrección de programas a base de intentos prueba-error, con lo cual puede llegar a la inhabilidad de comprender y analizar adecuadamente las soluciones.

En conclusión, todas estas dificultades representan un compendio de aspectos evidenciados en estudiantes involucrados en la incomprensión del curso. Dichas dificultades pueden ser causadas o compuestas por una o varias variables las cuales se declararán el siguiente subcapítulo. Cabe aclarar, que se tuvieron en cuenta las diferentes variables involucradas para el diseño y modelo de la propuesta. Esto con el fin de poder apoyar el proceso de aprendizaje en la mayor medida posible referente a las necesidades encontradas en los diferentes estudios expuestos.

7.2.3 Causas y variables involucradas.

Como se mencionó anteriormente por Xie, Li y Geng[62], el reto a la hora de llevar el contenido del curso a la vida real puede radicar en la dificultad del mismo y lo abstracto que pueden ser sus temas. Entendiendo abstracción como una operación o modelo mental fundamentado en un concepto no tangible que a su vez puede ser dinámico en diferentes situaciones.

Entre las variables involucradas según Muñoz[63] en un estudio llevado a cabo en diferente estudiantes en cursos introductorios a la programación está la dificultad al manejar un entorno de desarrollo, la sintaxis en el pseudocódigo, el diseño y codificación de un pseudocódigo, por último la detección de errores en el programa. Para lo cual entre las dificultades con mayor peso se encuentra la detección de errores, la codificación y el diseño de la estrategia, posteriormente está el uso del entorno de desarrollo. Así mismo, en el estudio de Muñoz[64] se pretende identificar los conceptos que generan mayor dificultad en los estudiantes al aprender un paradigma estructurado siendo estos la entrada y salida de datos, posteriormente las estructuras de repetición y finalmente las estructuras de selección.

---

https://www.researchgate.net/publication/221537536_Using_software_testing_to_move_students_from_trial-and-error_to_reflection-in-action

[62] XIE, LI, GENG, Op. cit., p. 676.

[63] MUÑOZ, Op. cit., p. 123.

[64] MUÑOZ, Op. cit., p. 124.

Mientras tanto en la investigación de Fuentes[65], determinar variables como la fobia a los problemas complejos (falta de confianza), lógica incompleta (abstracción), desconocimiento del lenguaje (sintaxis), incomprensión del entorno de desarrollo, falta de motivación y en algunos casos gestión del tiempo. Estas variables pudieron ser identificadas al realizar un estudio en 45 estudiantes los cuales debían resolver diferentes problemas, una vez identificadas las dificultades se puntualizaron las variables anteriormente mencionadas.

Más allá de eso, la aceptación de los estudiantes por la tecnología se ha convertido un factor crítico en el entendimiento. Esto con mayor razón a la hora introducir conceptos que adquieren un comportamiento dinámico en su ejecución. Muñoz[66] también afirma esta posición el sentido de que la aceptación por los medios digitales se ha vuelto crucial a la hora de enseñar el PE ya que en su estudio la gran mayoría de estudiantes consideraron los ejercicios, los programas de ejemplo y el internet como recursos útiles a la hora de aprender a programar. Muñoz[67] también declara en su estudio que los estudiantes aprenden mucho más en sus clases prácticas, prácticas autónomas o grupales y especialmente prácticas en el computador.

Connolly[68], autor anteriormente mencionado, había declarado variables como la confianza, la motivación y la confusión. Estas variables suelen ser detonantes en la inhabilidad analitica para la resolución de problemas. Esto a razón de que el estudiante debe estar dispuesto a intentar modelar soluciones, probarlas, y corregirlas. Si este proceso no es llevado a cabo, entonces es muy probable que el estudiante no pueda superar estos factores que lo limitan y que a su vez no lo motivan. Siendo la motivación intrínseca un factor de interés que puede ser determinante en el proceso educativo, ya que llevará al estudiante a enfrentar conceptos más complejos y desafíos más retadores.

7.2.4 El software educativo en la introducción a la programación.

Diferentes instituciones, investigadores y docentes han incorporado el software educativo en sus procesos pedagógicos introductorios a la programación. Esto a razón de las dificultades anteriormente expuestas. El propósito de estos actores a la hora de introducir estas herramientas es el de apoyar el proceso de aprendizaje.

---

[65] FUENTES, Op. cit., p. 79.
[66] MUÑOZ, Op. cit., p. 121.
[67] MUÑOZ, Op. cit., p. 124.
[68] CONNOLLY, Op. cit., p. 460.

Anderson[69] plantea la situación académica por la cual transitaba el Instituto Tecnológico de Durango en donde se expone una tasa de fracaso en el curso fundamentos de programación del 44%. Debido a este alto índice en los estudiantes Anderson[70] documentó el proceso de selección de un software educativo que pudiera acoplarse con las necesidades de los estudiantes que se encontraban en la tasa de fracaso. Es por ello que se implementó PSeInt como una herramienta de apoyo para el curso introductorio a la programación. Obteniendo así como resultado una disminución del 44% al 9% de reprobación del curso, siendo exitosa la implementación de la herramienta para mitigar el impacto negativo que venía teniendo con la metodología tradicional de enseñanza. Otros ejemplos de la implementación de esta herramienta con una efectiva viabilidad en sus resultados es la investigación de Carrizo[71], la cual aborda de manera detallada los recursos de la herramienta y explica cómo se puede ajustar a las necesidades del curso.

Así mismo, Redondo[72] expone un trabajo cuyo propósito gira entorno al proceso de aprendizaje como modelo colaborativo de instrucción. Este trabajo implementa diferentes herramientas que puedan atender a las necesidades del curso. En dicho documento se expresa la viabilidad de la aplicación de estas herramientas como recursos de apoyo los cuales le brindaran una perspectiva más concreta al estudiante lo que está trabajando. Entre las herramientas a trabajar, para la enseñanza del paradigma estructurado, se resalta SICAS como candidata a trabajar en el diseño establecido para el proceso pedagógico planteado.

Gomes[73] en su investigación expone la herramienta SICAS, herramienta la cual se compone desde la metodología heurística de George Polya[74] para la resolución de problemas. La cual pretende dividir el proceso en las 4 etapas (comprensión del problema, diseño de solución, ejecución de la solución, verificación de la solución). Por un lado, SICAS utiliza los diagramas de flujo como recurso para el diseño de soluciones y es posible su ejecución dentro de la herramienta. Así mismo, destaca

---

[69] ANDERSON, Juan y GONZALES-BAÑALES, Dora. PSeInt como herramienta para mejorar el proceso de enseñanza aprendizaje de algoritmos, pseudocódigo y diagramas de flujo. En: Tecnologías de la Información en Educación: Sistematización de experiencias docentes [en línea]. 1 ed. Durango: Red Durango de Investigadores Educativos, 2020. [Consultado el 1 de noviembre de 2020]. Capítulo 8. p. 91-102. Disponible en https://www.researchgate.net/profile/Dora_Gonzalez-Banales/publication/342243248_Tecnologias_de_informacion_en_educacion_Sistematizacion_de_experiencias_docentes/links/5eea3e96a6fdcc73be84b5aa/Tecnologias-de-informacion-en-educacion-Sistematizacion-de-experiencias-docentes.pdf#page=99

[70] Ibíd., p. 98.

[71] CARRIZO, Op. cit., p. 7.

[72] REDONDO, Op. cit., p. 2.

[73] GOMES, Op. cit., p. 3.

[74] POLYA, Op. cit., p. 22.

la posibilidad de incorporarse en el desarrollo de las clases para mejorar los conocimientos dirigidos a la programación así como su experiencia. Marcelino[75] sucede en esta investigación al incorporar esta herramienta en los cursos introductorios a la programación en la universidad de Coimbra (Portugal) y abandonando el método tradicional a base de papel y lápiz. Además, con dicha herramienta se abre la posibilidad de animar al estudiante a resolver los problemas propuestos por su propia iniciativa recibiendo una pronta retroalimentación y adquiriendo experiencia.

Por otro lado, Del Prado[76] en su investigación evalúa por medio de diferentes características el desarrollo y comportamiento de dos herramientas en el contexto académico como herramientas de apoyo a la introducción a la programación. Las herramientas son PSeInt y FreeDFD, y sus características evaluadas son su finalidad, representación en diagramas de flujo, usabilidad, amigable con el usuario, comprobación de errores, ejecución paso a paso, ejecución multiplataforma y documentación. Del Prado[77] destaca el valor que poseen estas herramientas como medio de apoyo y por tanto el impacto que tiene al educar los fundamentos en programación. Así mismo, concluye que la integración del software educativo en conjunto al acompañamiento pedagógico ha permitido identificar y orientar el papel del docente respecto al proceso educativo, y mejor aun permitiéndole al estudiante construir conocimiento de manera cooperativa y autónoma.

Xie, Li y Geng[78] en su artículo denotan el uso de laboratorios virtuales en el modelo educativo 'Blending Learning', esto con el fin validar de supuestos presentados por los estudiantes así como otorgar la libertad del diseño de soluciones y recibir retroalimentación de sus prácticas. Este tipo de satisfacción en los usuarios es clave en la implementación de programas E-Learning debido a la repercusión que tiene en la motivación del estudiante y su recepción. Como tal su modelo se encuentra estructurado con un enfoque en hardware más sin embargo también apropian temas referentes al paradigma estructurado para la puesta en práctica de sus ejercicios.

En conclusión, como se ha visto en las diferentes investigaciones expuestas se puede denotar una viabilidad en la introducción de software educativo como recurso de apoyo en los cursos introductorios a la programación. Así mismo, apoya la idea de renovar el proceso de enseñanza con el cual se ha trabajado tradicionalmente,

---

[75] MARCELINO, Op. cit., p. IV.8-4.
[76] DEL PRADO, Op. cit., p. 102.
[77] DEL PRADO, Op. cit., p. 112.
[78] XIE, LI, GENG, Op. cit., p. 678.

adaptándose así a nuevas generaciones más cercanas a la tecnología y los medios digitales.

7.2.5 Conclusión.

Con este capítulo se pudo concluir diferentes aspectos los cuales reflejan una inmersión en el campo para la comprensión del problema así como una concepción del diseño del estudio mediante una metodología cualitativa. Posterior a esta etapa se procederá a definir una estrategia la cual sirva como guía en el desarrollo de la solución propuesta y se ejecutará la misma. Finalmente, la propuesta será puesta en práctica en el respectivo grupo focal (muestra) para la recolección de datos, análisis de los mismos e interpretación de los resultados.

Por último, en este capítulo se pudo comprender el temario que incorpora el modelo pedagógico en la introducción a la programación así como la importancia del paradigma estructurado en la formación del estudiante siendo este paradigma un fundamento esencial. También se pudo evidenciar los procesos pedagógicos más comunes y su evolución respecto al tiempo, evolución la cual ha reinventado estos procesos con una tendencia hacia la tecnología de manera cada vez más propensa. Posteriormente, basados en los procesos pedagógicos expuestos, se pudo detallar en las dificultades evidenciadas por los diferentes investigadores. Dichas dificultades expresan, en muchas ocasiones, la consecuencias directas de variables involucradas en el proceso de aprendizaje. Siendo estas variables relativas en cada uno de los estudiantes o grupo de estudiantes, pero que a su vez son comunes en los individuos pertenecientes a la tasa de fracaso en cursos introductorios a la programación. Finalmente, a través de las diferentes investigaciones llevadas a cabo que incorporan el software educativo en los procesos de aprendizaje, se pudo detonar sus resultados y el impacto que tuvo en los grupos estudiados.

7.3 ESTADO ACTUAL

En la *Tabla 1* a continuación se listan las herramientas cuyo propósito recae en el apoyo a la introducción a la programación, destacando sus aspectos/características más relevantes y el modo en como se trabaja su solución.

Nombre	Descripción	Tipo	Puntaje	Link	S.O	Precio	Permisos	Lanzamiento	Actualización	Versión
PseInt	PSeInt es una herramienta para asistir a un estudiante en sus primeros pasos en programación. Mediante diferentes pseudolenguajes en español (complementado con un editor de diagramas de flujo).	Entorno de desarrollo basado en pseudolenguaje y diagramas de flujo. Uso múltiple de identificadores y sintaxis.	-	http://pseint.sourceforge.net/	Linux, Windows, MacOs	Gratuito	Almacenamiento	29/12/03	1/05/20	20200501
RAPTOR	El software permite al estudiante escribir y ejecutar programas que utilizan diagramas de flujo. Típicamente es utilizado en las academias para enseñar introducción a los conceptos de programación	Entorno de desarrollo basado en diagramas de flujo, sintaxis requerida al mínimo.	2,6	https://raptor.martincarlisle.com/	Windows	Gratuito	Almacenamiento	-	22/04/15	4.0.6
Visual Logic	Visual Logic proporciona una introducción de sintaxis mínima a los conceptos de programación esenciales que incluyen variables, entrada, asignación, salida, condiciones, bucles, procedimientos, matrices y archivos.	Entorno de desarrollo basado en diagramas de flujo, sintaxis requerida al mínimo.	-	https://www.visuallogic.org/	Windows	Gratuito	Almacenamiento	-	-	2.2.10
SICAS	SICAS es una herramienta que permite diseñar algoritmos (por medio de la construcción de algoritmos con diagramas de flujo) a problemas propuestos y ejecución de los mismos.	Entorno de desarrollo basado en diagramas de flujo.	-	https://www.researchgate.net/publication/228328489_SICAS_Interactive_system_for_algorithm_development_and_simulation	Windows	Uso privado, Universidad de Coimbra	Almacenamiento	-	-	-
FreeDfd	FreeDFD es un editor e intérprete de diagramas de flujo para los sistemas operativos Windows y Linux (con Wine) que permite trabajar tanto con algoritmos sencillos como complejos.	Entorno de desarrollo basado en diagramas de flujo.	3	https://freedfd.uptodown.com/windows	Windows, Linux (requiere complementos)	Gratuito	Almacenamiento	-	31/08/07	1.1
JkarelTheRobot	Karel es una herramienta de enseñanza que presenta los conceptos de forma visual, lo cual es menos abstracto que la programación en Java. Enfocada en la solución de problemas en un juego inmersivo.	Juego inmersivo exploratorio por medio de niveles.	4.1	https://jkarelrobot.software.informer.com/Descargar-gratis/	Linux, Windows, MacOs	Gratuito	Almacenamiento	12/02/09	29/02/12	1.3

Tabla 1. Estado Actual, comparación entre aplicativos

Nombre	Descripción	Tipo	Puntaje	Link	S.O	Precio	Permisos	Lanzamiento	Actualización	Versión
ProBot	ProBot es un juego educativo en el campo de la lógica de programación de computadores en el que los estudiantes pueden afianzar y mejorar sus habilidades respecto a los conceptos de secuenciación, iteración definida y anidación.	Juego inmersivo de combate por medio de niveles en presentación web.	-	https://www.researchgate.net/publication/228841113	Linux, Windows, MacOs (navegador)	Gratuito	Almacenamiento	00/01/2012	-	-
Laberithm	Laberithm es un juego inmersivo en donde en un laberinto se hace uso de las primitivas de programación.	Juego inmersivo exploratorio por medio de niveles.	-	http://www.dccia.ua.es/~faraon/docs/programacion.pdf	Windows	Uso privado, Universidad de Alicante	Almacenamiento	-	-	-
ABEA	Recurso didáctico en la enseñanza-aprendizaje de algoritmos estructurados a nivel superior con enfoque en el análisis y planteamiento del problema, diseño y traza de la solución.	Entorno de desarrollo basado en diagramas de flujo, requiere de importación de problemas.	-	http://sedici.unlp.edu.ar/bitstream/handle/10915/25531/Documento_completo.pdf?sequence=1&isAllowed=y	Linux, Windows, MacOS	Uso privado, Universidad del Istmo Campus Tehuantepec	Almacenamiento	-	-	-
SCRATCH	Scratch ayuda a los jóvenes a aprender a pensar de forma creativa, a razonar sistemáticamente, y a trabajar de forma colaborativa. Con Scratch puedes programar tus propias historias interactivas, juegos y animaciones y compartirlas.	Entorno de desarrollo (considerado lenguaje) basado en construcción por bloques con enfoque a la producción de videojuegos o contenido visual	Acepta-ción estadís-tica	https://scratch.mit.edu/	Linux, Windows, MacOs, Navegador	Gratuito	Almacenamiento	9/05/12	2020	3.11.1
Zinjal	Zinjal es un IDE (entorno de desarrollo integrado) libre y gratuito para programar en C/C++.	Entorno de desarrollo basado en pseudolenguaje y diagramas de flujo basado en lenguaje C y C++	-	http://zinjai.sourceforge.net/	Linux, Windows, MacOs	Gratuito	Almacenamiento	10/05/08	6/10/19	20190808

Tabla 1. (Continuación) Estado Actual, comparación entre aplicativos

Fuente *Propia*; Researchgate, Zinjai, Scratch, U. del Itsmo, U. de Alicante, JkarelRobot, Freedfd, VisualLogic, Raptor, PSeInt.

Como conclusión, cabe destacar que la propuesta de valor de esta proyecto se encuentra presente en diversos aspectos como, el enfoque en la detección de errores lógicos en algoritmos estructurados, uso de un único perfil de pseudocódigo, construcción de pseudocódigo por bloques en un entorno controlado, presentación multiplataforma, funcionalidad de registro de cambios, presentación de software libre, constitución de resolución de problemas con énfasis en las tres últimas etapas de George Polya, vigencia y uso de nuevas tecnologías.

8. DISEÑO METODOLÓGICO PRELIMINAR

A continuación se definirá en una tabla la elección tomada para el desarrollo de cada objetivo. Posteriormente se dará justificación a las metodologías escogidas.

Tabla 2. Relación objetivo - metodología

Objetivo	**Metodología**
Establecer el levantamiento de los requerimientos funcionales como no funcionales del prototipo que permitan la identificación de las necesidades en el proceso de aprendizaje en los cursos introductorios a la programación de paradigma estructurado.	Metodología de desarrollo ágil SCRUM
Diseñar y desarrollar un prototipo de software stand-alone por medio de la metodología ágil SCRUM en función a la problemática expuesta.	Metodología de desarrollo ágil SCRUM
Validar el prototipo por medio de pruebas funcionales y un grupo focal de estudiantes universitarios en ingeniería en sistemas.	Metodología de desarrollo ágil SCRUM

Fuente: *Autor propia*

8.1JUSTIFICACIÓN DE METODOLOGÍA – INVESTIGACIÓN CUALITATIVA

Al ser un proceso que requiere la definición de un marco teórico es importante la investigación como recurso para el desarrollo de este. Donde según Sampieri[79] la investigación es “un conjunto de procesos sistemáticos, críticos y empíricos que se aplican al estudio de un fenómeno o problema”. Actualmente existen dos tendencias más populares, la investigación cualitativa y la investigación cuantitativa. Entre las diferencias principales está el punto de partida donde la cuantitativa conoce una realidad mientras la cualitativa busca descubrirla, en la cualitativa la realidad puede cambiar por las observaciones y recolección de datos. Otra diferencia se debe a que la cuantitativa busca ser objetiva mientras que la cualitativa es subjetiva. Es decir, la cuantitativa busca comprobar y predecir fenómenos mientras que la cualitativa busca comprender fenómenos por medio de percepciones. Por otro lado, la cuantitativa es deductiva mientras que la cualitativa es inductiva. Asimismo, el objetivo de la investigación cuantitativa busca generalizar los resultados mientras que en la cualitativa no.

[79] SAMPIERI, Op. cit., p. 4.

La principal razón por la cual se seleccionó la investigación cualitativa se debe a que el objeto de estudio son las dificultades de los alumnos en cursos introductorios a la programación. Este objeto de estudio no es cuantificable y recae en la experiencia de los individuos.

8.2JUSTIFICACIÓN DE METODOLOGÍA – DESARROLLO ÁGIL SCRUM

Para Tinoco[80] existen 2 tipos de metodologías, las tradicionales (rigurosas en la medición de indicadores y cumplimiento de actividades) y las ágiles (flexibles al cambio enfocadas en el factor humano e integradoras del cliente en su desarrollo). Usualmente ambas metodologías se encuentran definidas por etapas las cuales suelen caracterizan en cuatro tipos como lo menciona Tinoco[81]:

- **Secuencial:** Su desarrollo transcurre de una etapa a otra cuando la anterior finalizó. La metodología más famosa es cascada, analizando requisitos, implementando diseños, desarrollo y pruebas.
- **Incremental:** Su principal objetivo es reducir tiempos al establecer intervalos superpuestos, en este punto los requisitos se dividen en incrementos independientes y funcionales.
- **Iterativo:** Se centra en capturar los requisitos cambiantes y riesgos presentes, cada iteración presenta un entregable.
- **Espiral:** Similar al iterativo, haciendo análisis de alternativas y reducción de riesgos.

En la tabla a continuación, tomada del documento Tinoco[82], se estiman algunos aspectos esenciales de las dos principales categorías de metodologías.

Tabla 3. Comparación de metodologías de desarrollo de software

[80] TINOCO, Oscar; ROSALES, Pedro y SALAS, Julio. Criterios de selección de metodologías de desarrollo de software. En: Industrial Data [en línea]. Lima: Universidad Nacional Mayor de San Marcos, julio de 2010. vol. 13, nro. 2. p. 70-74. [Consultado el 1 de noviembre de 2020]. Disponible en: https://www.redalyc.org/pdf/816/81619984009.pdf. ISSN: 1560-9146
[81] Ibíd., p. 70.
[82] Ibíd., p. 71.

Metodologías ágiles	Metodologías tradicionales
Se basan en heurísticas provenientes de prácticas de producción de código	Se basan en normas provenientes de estándares seguidos por el entorno de desarrollo
Preparados para cambios durante el proyecto	Cierta resistencia a los cambios
Impuestas internamente por el equipo	Impuestas externamente
Proceso menos controlado, con pocos principios	Proceso muy controlado, numerosas normas
Contrato flexible e incluso inexistente	Contrato prefijado
El cliente es parte del desarrollo	Cliente interactúa con el equipo de desarrollo mediante reuniones
Grupos pequeños (<10)	Grupos grandes
Pocos artefactos	Más artefactos
Menor énfasis en la arquitectura del software	La arquitectura del software es esencial

Fuente: TINOCO, ROSALES y SALAS. Criterios de selección de metodologías de desarrollo de software.

Ahora bien, teniendo en cuenta el tiempo de desarrollo (4 meses), la curva de aprendizaje que conlleva el desarrollo (alta probabilidad de cambio) y la cantidad de desarrolladores en el proceso (uno), la gestión del proceso (interna), la existencia de un contrato (ausente) en la propuesta presentada; es necesario evaluar las metodologías ágiles como mejor opción.

Entre la elección de una de las metodologías ágiles existentes es necesario analizar sus propósitos, aspectos principales, documentación y facilidad de implementación.

Según Navarro[83] las más populares son SCRUM, XP y Crystal; más sin embargo se escogió SCRUM debido a las características previamente expuestas, además del conocimiento del equipo de desarrollo sobre esta. Con ello es posible evadir una curva de aprendizaje la cual puede afectar drásticamente los plazos planteados para la entrega del proyecto. Además, para el autor Orjuela[84] en el desarrollo de software educativo definitivamente afirma que debe ser de alguna metodología ágil debido a las características de esta vanguardia.

[83] NAVARRO, Op. cit., p. 33.
[84] ORJUELA, Ailin y ROJAS, Mauricio. Las Metodologías de Desarrollo Ágil como una Oportunidad para la Ingeniería del Software Educativo. En: Revista Avances en Sistemas e Informática [en línea]. Medellín: Universidad Nacional de Colombia, junio de 2018. vol. 5, nro. 2. p. 159-171. [Consultado 1 de noviembre de 2020]. Disponible en:https://www.redalyc.org/pdf/1331/133115027022.pdf

9. RECURSOS DISPONIBLES

9.1 RECURSOS MATERIALES

Librerías:

- JavaFx (tecnologías Oracle para la producción de aplicativos)
    - StyledTextArea
- Librerías básicas incluidas en el JDK

IDE:

- Eclipse

Lenguaje:

- Java

9.2 RECURSOS INSTITUCIONALES

Investigación:

- Profesor de curso 'Introducción a la Programación'
- Estudiantes de Ingeniería en Sistemas de la Universidad Minuto de Dios (Sede Bogotá Calle 80) y estudiantes de Ingeniería en Sistemas de la Universidad Piloto de Colombia.

Documentación:

- Bases de datos disponibles por medio de la Universidad Piloto de Colombia

9.3 RECURSOS FINANCIEROS

- Servicio de Internet
- Servicio de Energía
- Computador Portatil Macbook Pro (Retina, 13-inch, Early 2015)

10. CRONOGRAMA

Objetivo Específico	Software enseñanza-aprendizaje algoritmos estructurados	Sprint	ID	Semana 1	Semana 2	Semana 3	Semana 4	Semana 5	Semana 6	Semana 7	Semana 8	Semana 9	Semana 10	Semana 11	Semana 12	Semana 13	Semana 14
	Correción y revision de Investigación																
	Propuesta																
	Tema provisional																
	Investigación sobre enseñanza cursos introductorios a programación (solución de problemas)																
	Formulación del Problema																
	Investigación sobre deficit de talento IT																
	Investigación sobre la importancia de la programación en otras carreras																
	Investigación sobre la taza de fallas en cursos introductorios a la programación																
	Justificación del Problema																
	Investigación sobre las dificultades de los estudiantes en cursos introductorios a la programación																
	Investigación sobre las herramientas actuales																
	Investigación sobre las estrategias de enseñanza en programación																
	Objetivos																
	Definición de objetivo general																
	Definición de objetivos estratégicos																
	Anteproyecto																
	Formulación del Problema																
	Investigación sobre deficit de talento IT																
	Investigación sobre la importancia de la programación en otras carreras																
	Investigación sobre la taza de fallas en cursos introductorios a la programación																
	Justificación del Problema																
	Investigación sobre las dificultades de los estudiantes en cursos introductorios a la programación																
	Investigación sobre las estrategias de enseñanza en programación																
	Objetivos																
	Definición final de objetivo general																
	Definición final de objetivos estratégicos																
	Marco Referencial																
	Marco Conceptual																
	Investigación y definición de conceptos clave en el proyecto																
	Marco Teorico																
	Ampliación temporal de conceptos clave en el proyecto																
	Estado Actual																
	Investigación sobre las herramientas actuales																
	Diseño metodológico preliminar																
	Recursos																
	Investigación de recursos presentes en el proyecto																
	Definición de recursos presentes en el proyecto																
	Cronograma																
	Estructura de descomposición del trabajo																
	Planeación estimada de actividades																
Establecer el levantamiento de los requerimientos funcionales como no funcionales de la aplicación que permitan la identificación de las necesidades en el proceso de aprendizaje en los cursos introductorios a la programacion de paradigma estructurado.	Levantamiento de requerimientos																
	Analisis de dificultades en el marco teorico																
	Identificacion de dificultades y variables involucradas																
	Relacion de variables involucradas y necesidades																
	Requerimientos funcionales																
	Entrevistas con el product owner (Giovanni Fajardo)																
	Identificacion de requerimientos																
	Analisis y ponderacion de requerimientos																
	Requerimientos no funcionales																
	Entrevistas con el product owner (Giovanni Fajardo)																
	Identificacion de requerimientos																
	Analisis y ponderacion de requerimientos																
Desarrollar un prototipo de software stand-alone por medio de la metodologia ágil SCRUM en funcion a los requerimientos previamente establecidos	Desarrollo																
	Diseños, modelos y vistas																
	Diseñar y modelar vista funcional	0	1														
	Diseñar y modelar vista de contexto																
	Diseñar y modelar vista de despliegue																
	Diseñar y modelar modelo de clases																
	Como estudiante deseo generar y remover estructuras de pseudocodigo para el desarrollo de algoritmos estructurados	1	2														
	Crear frame de la aplicacion																
	Definir areas de trabajo dentro del frame																
	Incluir TabPane del area de inserción de codigo																
	Incluir Tab dentro del TabPane del area de inserción de codigo																
	Importar StyledTextArea y definirla como clase extendida del objeto CodeArea para cada Tab dentro del area de inserción de codigo																
	Crear clase de InsertMenu extendida de ContextMenu																
	Dentro de CodeArea definir motor de estilos basado en keywords ~plabras clave~ de las estructuras de psedocodigo																
	Definir InsertMenu como menu de contexto para CodeArea																
	Dentro de InsertMenu definir motor de insercion para las diferentes estructuras de codigo (indentacion)																
	Definir reglas de inserción dentro de InsertMenu																

Tabla 4. Cronograma del proyecto

Desarrollar un prototipo de software stand-alone por medio de la metodologia ágil SCRUM en funcion a los requerimientos previamente establecidos	Como estudiante deseo ingresar en los campos disponibles las expresiones aritmeticas o logicas para dar sentido al pseudocodigo desarrollado	1	3
	Dentro del motor de estilos de CodeArea definir (según la estructura) los limites de escritura		
	Dentro de CodeArea aplicar un Listener que valide la posicion del signo de intercalación dentro de los limites de escritura		
	Aplicar en motor de CodeArea un corrector de espacios para limpiar indentación		
	Como estudiante deseo que el software me notifique los errores lexicos, sintacticos o sematincos en las expresiones para apoyar el proceso de aprendizaje	2	4
	Crear objeto Tabla de Simbolos *provicional con funciones Hash y objeto Hashtable		
	Definir objeto Variable con tipo, nombre, valor		
	Crear el objeto Analizadores con evaluacion lexica que valide los identificadores inmersos en las expresiones		
	En caso de declaracion, añadir identificador a Tabla de Simbolos		
	Dentro del objeto Analizadores definir dentro del motor un analizador sintáctico que valide la estructura de la expresion		
	Dentro del objeto Analizadores definir dentro del motor un analizador semántico que valide el proposito o sentido de la expresion respecto a la estructura perteneciente desde la notación polaca inversa		
	Crear el metodo evaluativo de expresiones posfijas evaluar que permita operar y conocer el tipo de dato del resultado		
	En caso de incosistencias en los analizadores notificar al motor de estilos la inexactitud o la correcta validacion		
	En CodeArea actualizar estilos segun la respuesta de los analizadores por cada expresion		
	Como estudiante deseo guardar mi pseudocodigo en un archivo para poder tenerlo disponible en cualquier momento	3	9
	Crear boton en BarraSuperior de guardar archivo		
	Crear interfaz GestorDeArchivos		
	Implementar metodo de creacion de archivos basado en el codigo fuente tomado de CodeArea		
	Como estudiante deseo abrir mi pseudocodigo almacenado para poder trabajar interrumpidamente en ellos	3	10
	En BarraSuperior crear boton para abrir archivos, implementando un gestor de lectura de archivos en el sistema		
	En la interfaz GestorDeArchivos implementar metodo de lectura de archivos		
	Crear nueva CodeArea en TabPane y asignar contenido de archivo como codigo fuente		
	Como estudiante deseo ver en consola los errores presentados en mis expresiones para denotar en detalle las posibles causas del mismo	3	11
	Dentro de Frame, denotar area Consolas extendida de TabPane		
	Crear objeto Consola extendido de Tab para consola de Errores, con componente de TextArea no editable ~Considerar cambio a RichText~		
	Implementar metodos de asignacion de texto para componentes de TextArea		
	En Analizadores, identificar posibles errores (comúnes) y clasificarlos, diferenciar expresiones simples o compuestas segun estructura y campo, listarlas		
	Plasmar lista de inconsistencias en consola de errores		
	Como estudiante deseo poder secuenciar mi codigo para luego ver los resultados basados en mi algoritmo	4	5
	Dentro del Frame principal definir area de acciones ~BarraSuperior~ que extiende la clase HBox		
	Crear boton Ejecutar que llama a *ejecutar* de Ejecutor		
	Tomar Tabla de Simbolos *provicional ~Analizadores~ y asignar a Tabla de Simbolos *ejecutar ~Ejecutor~		
	Llamar a *secuenciar* de Secuenciador y enviar lista de instrucciones a *ejecutar* de Ejecutor		
	Crear objeto Secunciador para tomar codigo fuente, secuenciarlo y generar lista de instrucciones		
	Crear objeto NodoSecuenciador con número de instruccion, instruccion y tipo para la lista de instrucciones		
	Crear objeto CierreSecuenciador para denotar saltos en la lista de instrucciones		
	Crear la interfaz Posfijo para convertir las expresiones en notacion polaca inversa		
	Crear la interfaz evaluador para evaluar las expresiones con conicimiento de tipo de dato, nombre y resultado/valor		
	Crear objeto Ejecutor recorrer lista de instrucciones según respuestas de evaluador para cada instrucción, trabajar con Tabla de Simbolos segun las estrucutras (delcaracion/asignacion)		
	Añadir Registro		
	Como estudiante deseo poder ingresar mi prueba de escritorio manual para desarrollar un modelo mental de lo que ocurre con mi programa	4	7
	Crear clase Tabla con Tab de clase extendida para Prueba manual dentro de Tablas		
	Crear botón en BarraSuperior para actualizar titulos en el area manual ~Llamar a columnasNuevas en Tabla~		
	Crear botón en BarraSuperior para agregar filas nuevas en el area manual ~Llamar a filasNuevas en Tabla~ con TextFields para el ingreso manual		

Tabla 4. (Continuación) Cronograma del proyecto

Desarrollar un prototipo de software stand-alone por medio de la metodologia ágil SCRUM en funcion a los requerimientos previamente establecidos	Como estudiante deseo ejecutar mi pseudocodigo paso a paso para poder evidenciar una trazabilidad de la ejecucion del mismo	5-6	8
	En BarraSuperior denotar campo de segundos para recorrer la lista		
	En BarraSuperior implementar boton de secuencia pausada para recorrer a siguiente instruccion solo cuando sea oprimido		
	Dentro de Ejecutor incorporar hilo basado en los segundos definidos en el campo de barra superior		
	Al ejecutar, definir la posicion del signo de interacalacion en CodeArea segun la linea que se este evaluando		
	Dentro de Ejecutor, analizar siguiente instruccion si el boton secuencia pausada es oprimido		
	Como estudiante deseo ver una prueba de escritorio automatica para poder evidenciar mis errores logicos en el codigo	6-7	6
	Crear clase Tablas con TabPane como tabla extendidad para el area de Pruebas de Escritorio		
	Crear clase Tabla con Tab de clase extendida para Prueba automatica dentro de Tablas		
	Definir un ComboBox con los indices de pruebas ejecutadas en BarraSuperior		
	Definir metodos de columnasNuevas (declaracion de identificadores como titulo), filasNuevas (añadir filas con campos para denotar el cambio) y filasNuevasYCambio (para agregar filas y en la columna indicada ~variable~ agregar su nuevo valor segun la ejecución)		
	Recibir Registro segun el indice seleccionado en el ComboBox y recorrerlo en funcion de filasNuevasYCambio para visualizar el registro de manera grafica		
	Como estudiante deseo ingresar datos en la consola de entrada para poder ejecutar mi pseudocodigo respecto a entradas dinamicas	8	12
	En ejecucion crear pendiente de lectura (tipo de dato, nombre y linea de lectura)		
	Asignar cursor a campo de lectura		
	Convertir dato y reasignar valor en tabla de simbolos		
	Aplicar cambios y corrreciones		
	Actualizar informacion en documentacion y vistas y modelos		
	vi	9	13
	Evaluar expresion tipo texto para la escritura		
	Escribir la expresion en el campo de lectura y consola de informacion/errores		
	Aplicar cambios y corrreciones		
	Actualizar informacion en documentacion y vistas y modelos		
Validar el prototipo por medio de pruebas funcionales y un grupo focal de estudiantes universitarios en ingenieria en sistemas	Pruebas		
	Set de pruebas funcionales		
	Correcciones y cambios		
	Reunion de grupo focal		
	Analisis y evaluacion de encuestas		
	Informe de resultados en documento de tesis		

Tabla 4. (Continuación) Cronograma del proyecto

Fuente *Autor propia*

11. ESTADO DEL ARTE

El presente capítulo abordará los principales aportes de investigación con similitudes a la presente investigación en desarrollo, en estos se trabaja el uso de software educativo como herramienta de apoyo para las dificultades que se encuentran presentes en el proceso de aprendizaje en materia de programación. Los presentes se encuentran delimitados en los últimos 4 años en diferentes localizaciones como Grecia, Nueva Zelanda, Argentina, Brasil, Ecuador y Colombia. El propósito de este capítulo es dar contexto a la solución propuesta en apartados anteriores al mostrar similitudes con respecto a los aportes actuales que se encuentran respecto a este problema.

11.1 EN EL MUNDO

Como bien lo mencionan Giannakoulas y Xinogalo[85] del Departamento de informática Aplicada de la Universidad de Macedonia (Grecia), explican que entre las dificultades más presentes a la hora de enseñar los conceptos de programación está la ausencia de interactividad en los métodos tradicionales. Es por ello que se propone un software educativo con forma de juego, con el fin de introducir a nuevos estudiantes en la programación. *RunMarco* es un juego ambientado en una jungla cuyo personaje principal debe seguir un camino, el cual a lo largo de su progreso encuentra retos más complejos. El modo para operar al personaje es por medio de la programación visual de bloques "*Blockly*". Entre las primitivas básicas se encuentra: programación lineal/secuencial, ciclos, condicionales simples, condicionales compuestos y condicionales anidados.

Otra solución software educativo con esta presentación es *LightBot*, una propuesta expuesta por Mathrani, Christian y Ponder-Sutton[86] de la universidad de Massey (Nueva Zelanda) con el nombre de *PlayIT*. En donde exponen una falta de interés por parte de los estudiantes respecto a conceptos relacionados con el desarrollo TI. Así mismo argumentan la necesidad de brindar experiencias prácticas para reforzar y aplicar conceptos teóricos vistos en clase. *PlayIT* está diseñado para cubrir el

---

[85] GIANNAKOULAS, A. y XINOGALOS, Stelios. A pilot study on the effectiveness and acceptance of an educational game for teaching programming concepts to primary school students. En: Education and Information Technologies [en línea]. Macedonia: Universidad de Macedonia, 2018. vol. 23, nro. 5. p. 2029-2052. [Consultado el 8 de marzo de 2021]. Disponible en: https://ruomo.lib.uom.gr/bitstream/7000/342/1/EAIT-RunMarco-postprint.pdf

[86] MATHRANI, CHRISTIAN, y PONDER-SUTTON, Op. cit., p. 10.

temario de tres cursos orientados a la computación, cuyos temarios se resumen en flujo secuencial de ejecución, funciones/procedimientos, recursión/ciclos y condicionales. La temática principal de este es solucionar diferentes retos presentados ante el personaje de un robot, el cual debe cumplirlos para llegar al siguiente nivel.

Tanto *PlayIT* (*LightBot*) como *RunMarco* tuvieron resultados positivos los cuales permitieron el desarrollo de conceptos teóricos en un entorno práctico dentro del aula. Esto permitió crear un modelo mental en el estudiante el cual apoya su proceso de aprendizaje en materia a la solución de problemas por medio de la programación. Por otro lado en latinoamérica se pueden encontrar software educativo con un enfoque dirigido más hacia una herramienta que un juego.

11.2 EN LATINOAMÉRICA

Salcedo, Müller y Otero[87] de la Universidad Abierta Interamericana (Argentina) exponen una herramienta multiplataforma colaborativa orientada a la programación estructurada. UAICase tiene un enfoque colaborativo en el diseño de algoritmos correspondientes al paradigma estructurado, ya que recoge la interacción y aporte de diferentes participantes como medio de retroalimentación. Una característica de esta herramienta es su portabilidad ya que se presenta como una aplicación móvil. Más allá de ello, no se menciona el modo en cómo esta herramienta se desarrolla en función al diseño de algoritmos y ejecución de los mismos, notificación de errores, así como el apoyo para la detección de errores de carácter lógico.

Camana y Slaguero[88] de la Universidad Técnica de Ambato (Ecuador) presentan VisualWebMedia como herramienta para la enseñanza de conceptos básicos de la programación estructurada. VisualWebMedia es un aplicativo web el cual apoya el proceso de enseñanza desde los recursos multimedia tales como textos, gráficos,

---

[87] SALCEDO, Walter; MÜLLER, Christian y OTERO, Manuel. III Jornadas Argentinas sobre Tecnología, Innovación y Creatividad [en línea]. En: III JATIC 2017. (1-3, noviembre. Buenos Aires, Argentina). UAIDesign: Herramienta Colaborativa Multiplataforma para la enseñanza de Diseño de Algoritmos y Programación Estructurada. Buenos Aires. 2017. p. 1-8. [Consultado el 8 de marzo de 2021]. Disponible en: http://bon-cloud.net/info/pubs/07-jatic-2017.pdf

[88] CAMANA, Roberto y SALGUERO, Anita. Congreso Internacional “Ciencia, Sociedad e Investigación Universitaria” [en línea]. En: II Congreso Internacional “Ciencia, Sociedad e Investigación Universitaria”. (18-20, octubre. Ecuador). Validación y evaluación de VisualWebMedia para la enseñanza de conceptos básicos deprogramación estructurada. Ecuador. 2017. p. 1-16. [Consultado el 8 de marzo de 2021]. Disponible en: https://repositorio.pucesa.edu.ec/handle/123456789/2321

imágenes, animaciones y videos. Este aplicativo por medio de diferentes módulos explica de manera teórica los diferentes conceptos involucrados en la programación estructurada siendo un apoyo audiovisual para el entorno de aprendizaje donde se desarrolla. Su función se limita a la exposición del concepto y no realiza una interacción usuario-aplicativo, la cual no permite comprender el comportamiento del concepto en ejecución de alguna solución diseñada por el usuario.

Por otro lado, en la Universidad de São Paulo (Brasil) Souza, Isotani y Barbosa[89] proponen una solución como medio de apoyo a la introducción de fundamentos en computación, los cuales demuestran un grado de dificultad al ser abstractos y dinámicos. Su propuesta va dirigida a una herramienta evaluativa la cual genera una retroalimentación inmediata respecto a la calidad de los algoritmos generados y los casos de prueba ejecutados, ayudando a los estudiantes a mejorar en la materia respecto a la programación y testeo. El software propuesto es *ProgTest* el cual es un entorno web para la aplicación de pruebas. Este software únicamente se encarga en la evaluación de la tarea propuesta y declara los errores que han tenido los estudiantes al aplicar determinados casos de prueba propuestos por el docente los cuales pueden tener un peso diferente en su calificación.

11.3 EN COLOMBIA

Otra propuesta orientada a la educación en la introducción a la programación es la de Porto *et al*[90] de la Corporación Universitaria Americana (Colombia), la cual se define como Objetos Virtuales de Aprendizaje (OVAs). Los OAs son materiales virtuales educativos cuyo propósito es aumentar la ocasión de aprendizaje según su uso. Es por ello que se expone una plataforma la cual cubre un marco de enseñanza apoyada en OVAs. En donde se exponen asignaturas basadas en guías, lecciones, cuestionarios, foros, multimedia, webgrafía y almacenaje colaborativo. La

[89] OUZA, Draylson Micael; ISOTANI, Seiji y BARBOSA, Ellen Francine. Teaching novice programmers using ProgTest. En: International Journal of Knowledge and Learning [en línea]. Switzerland: Inderscience Enterprises Limited, agosto de 2015. vol. 10, nro. 1. p. 1-18. [Consultado el 8 de marzo de 2021]. Disponible en https://www.researchgate.net/profile/Seiji-Isotani/publication/281578819

[90] PORTO, Roberto et al. MARCO DE ENSEÑANZA BASADO EN OVAs PARA EL ADIESTRAMIENTO EN LA INTRODUCCIÓN DE LOS ALGORITMOS (OLIA). En: Revistas de la Universidad de Granada etic@net [en línea]. Granada: Universidad de Granada, 2017. vol. 17, nro. 1. p. 60-75. [Consultado el 8 de marzo de 2021]. Disponible en http://eticanet.org/revista/index.php/eticanet/article/view/126

principal característica de interés es un compilador javascript, pero en la propuesta no se detalla su uso o el modelo como se aplica.

Por último está el trabajo que se expone en el presente documento, TestDesk es una propuesta diseñada con un modelo flexible de trabajo, con el propósito de moldearse al proceso pedagógico que cubra el docente respecto a su curso. Esto debido a que otras soluciones proponen un modelo categorizado a la hora de presentar los conceptos desde las primitivas básicas de programación, cuyo fin puede repercutir como un limitante a la hora de adaptarse con el modelo pedagógico antepuesto. Así mismo, su flexibilidad mitiga la memorización de rutinas (variable causante del problema). Por otro lado, permite una retroalimentación inmediata con el estudiante a la hora de insertar expresiones y ejecutar diferentes casos de prueba mostrando lo que ocurre en memoria y notificando sus errores en las expresiones. Con lo cual no solo es adaptable en el curso sino que también figura como un apoyo en el trabajo autónomo, esto debido a la capacidad que posee a la hora de ejecutar algoritmos estructurados de manera local (sin la necesidad de una plataforma web). Otro valor agregado es la capacidad de mostrarle al estudiante lo que ocurre con su programa en memoria, lo cual soluciona un problema común en estos eventos causantes en la dificultad a la hora de aprender a programar.

12. LEVANTAMIENTO DE REQUERIMIENTOS

El presente capítulo tiene como propósito analizar las dificultades presentes en los cursos introductorios a la programación con el fin de realizar el levantamiento de requerimientos funcionales y no funcionales, los cuales estarán inmersos en el prototipo. Esto partiendo de las dificultades expuestas en el capítulo de Marco Teórico y de esta manera evadir los errores léxicos, gramaticales y sintácticos al desarrollar un entorno asistido para concentrar al estudiante en los errores lógicos. Por tanto, el presente capítulo se desarrolla desde el análisis de las dificultades, posteriormente en el levantamiento de requerimientos funcionales y finalmente en los requerimientos no funcionales.

12.1 DIFICULTADES EN UN LENGUAJE FORMAL

Como se puede apreciar en el capítulo 7.2 (Marco Teórico), se pueden resumir las dificultades en la introducción a la programación por medio de variables tales como el desconocimiento de estructuras basicas de programacion, dificultad en la codificación de la estrategia, memorización de rutinas, abstracción, desconocimiento de errores sintácticos y lógicos, dificultad en la detección de errores, inhabilidad de comprender lo que ocurre en memoria, incomprensión de conceptos teóricos en entornos dinámicos. Así mismo, la investigación de Muñoz[91] determinó variables favorables a la hora de apoyar el proceso de aprendizaje como la aceptación por medios digitales, la práctica autónoma de los conocimientos y la experimentación como medio de retroalimentación, los medios dinámicos, metodologías heurísticas para la resolución de problemas y el software educativo.

A razón de ello se procede a relacionar dichas variables con recursos pedagógicos que apoyen el proceso de aprendizaje en dichos cursos. En el marco teórico de esta propuesta, la programación se reconoce como una tarea que se basa en la resolución de problemas. Como tal no existe un método definitivo para la resolución de problemas hablando de manera general. Aún así, existen metodologías heurísticas aceptadas por el entorno académico las cuales pueden apoyar este proceso. Una de ellas es la metodología heurística de George Polya[92] la cual comprende la resolución de un problema en 4 etapas: la comprensión del problema,

[91] MUÑOZ, Op. cit., p. 121.
[92] POLYA, Op. cit., p. 28.

el diseño de una estrategia, la ejecución de la estrategia y la comprobación de la estrategia. Siendo esta última importante para el análisis del algoritmo y la detección de errores.

Además de este enfoque es necesario analizar otras partes que componen a la solución, y podría referirse a la presentación de esta. En la presente propuesta se tiene planeado la integración del pseudocódigo como recurso propio de codificación, esto debido a que una de las dificultades de los estudiantes es codificar la estrategia y haciendo uso de este recurso se puede dar transitoriedad a un lenguaje de programación de alto nivel. Además como lo menciona Shneiderman[93] en su investigación, los diagramas de flujo (recurso alternativo) no son significativos en el aspecto general de comprensión, modificación o composición en la educación de la programación; de hecho es posible que omitan información valiosa, declaraciones o formatos de entrada y salida. Dejando en claro que su uso no es esencial ya que genera el mismo impacto que el pseudocódigo. Dentro de su codificación se encuentran aspectos que facilitarán la comprensión del pseudocódigo para el estudiante, uno de ellos es la indentación. En un estudio llevado a cabo por Miara *et al*[94] de la Universidad de Maryland se concluye que la indentación del código facilita la comprensión del mismo siempre y cuando sea llevado a cabo de manera moderada. Según Miara y otros[95], este aspecto (muchas veces considerado como buena práctica) facilita la lectura del código y consigo la comprensión, la modificación, el mantenimiento y las pruebas del código.

12.2 REQUERIMIENTOS FUNCIONALES

Debido a que la implementación de esta propuesta se llevará a cabo con estudiantes de la Universidad Minuto de Dios y posiblemente la Universidad Piloto de Colombia, se llevaron a cabo diferentes entrevistas con un docente encargado de dictar la introducción a la programación en estos entornos (Giovanni Fajardo Utria). Únicamente se tomó la valoración de un docente en función de acople con la

[93] SHNEIDERMAN, Ben et al. Experimental Investigations of the Utility of Detailed Flowcharts in Programming. En: Communications of the ACM [en línea]. Estados Unidos: Universidad de Indiana, junio de 1977. vol. 20, nro. 6. p. 373-381. [Consultado el 2 de Octubre de 2020]. Disponible en: https://www.cs.umd.edu/~ben/papers/Shneiderman1977Experimental.pdf

[94] MIARA, Richard et al. PROGRAM INDENTATION AND COMPREHENSIBILITY. En: Communications of the ACM [en línea]. Estados Unidos: Universidad de Maryland, noviembre de 1983. vol. 26, nro. 11. p. 861-867. [Consultado el 1 de noviembre de 2020]. Disponible en: https://www.cs.umd.edu/~ben/papers/Miara1983Program.pdf

[95] Ibíd., p. 867.

metodología de desarrollo SCRUM, para que este pudiera ejercer el rol de *product owner*. En dichas entrevistas se pudieron denotar las mismas variables referentes a los estudios anteriormente citados, agregando necesidades como la de un prototipo de software educativo asistido de desarrollo para poder ejercer educación remota en vista del cambio que tuvo el modelo pedagógico a razón de la pandemia COVID-19.

En las entrevistas llevadas a cabo entre Agosto - Noviembre del 2020 y Febrero - Marzo del 2021 se pudieron establecer requerimientos como:

12.2.1 Integración y remoción de estructuras básicas de programación por bloques.

La generación de estructuras básicas por medio de bloques es necesaria para evitar los errores sintácticos, léxicos y gramaticales que componen las dificultades para los principiantes en la programación. Se puede destacar esto debido a la cantidad de errores que suelen abrumar a los principiantes en un entorno de desarrollo integrado formal. Del mismo modo, apoya al estudiante al comprender el algoritmo como un conjunto de partes que conforma una solución, en otras palabras, en la abstracción. Entre las estructuras básicas de programación únicamente se denotará la asignación; la declaración; la escritura; la lectura; condicional simple; condicional compuesto; iteración delimitada por un numero positivo de veces con paso 1 y lógica. Los tipos de datos involucrados en el prototipo serán texto, lógico, entero y real; sin usar incorporar vectores o matrices. Además de ello, se debe incorporar la indentación para facilitar la lectura del algoritmo. Por último, a la hora de eliminar el bloque se deben remover las estructuras inmersas en él con el fin de evitar confusión.

12.2.2 Ingresar expresiones aritméticas y lógicas en los campos requeridos.

Es necesario establecer un filtro el cual únicamente le permita al estudiante ingresar las expresiones aritméticas y lógicas de manera libre. Expresiones las cuales son necesarias para que el algoritmo pueda cobrar sentido en la ejecución. Estas expresiones unicamente pueden operar bajo los operadores declarados en el alcance.

12.2.3 Ejecutar o simular la ejecución del pseudocódigo diseñado.

Según los diferentes estudios mencionados y las entrevistas llevadas a cabo con el docente, es necesario mostrarle al estudiante el comportamiento de las estructuras

en composición del algoritmo en un entorno dinámico y de cómo estas interactúan con los valores antepuestos. Así mismo, es necesario indicar en la ejecución la línea la cual se esté evaluando para indicarle al estudiante el curso que está llevando a cabo la ejecución. Además, debe ser una ejecución pausada tipo *debug* para que el estudiante pueda asimilar los cambios.

12.2.4 Generar una prueba de escritorio automática.

Durante la ejecución es necesaria la creación de una prueba de escritorio para superar la inhabilidad de comprender lo que ocurre con el programa en memoria. Así mismo, es necesario denotar en las columnas de la prueba de escritorio las variables en el orden que fueron declaradas por el estudiante. También, los cambios se deben registrar en la misma línea donde el pseudocódigo los ha evaluado y citado para que el estudiante pueda comprender en qué momento ocurrió el cambio y la razón de este.

12.2.5 Ingresar una prueba de escritorio manual.

Si bien la prueba de escritorio automática resulta un apoyo para el estudiante al comprender lo que ocurre con el programa en memoria, puede resultar un facilitador que acostumbre al estudiante a usar este recurso. Es por esto que es necesario un espacio para realizar una prueba de escritorio de manera manual (que el usuario ingrese los datos) para poder apoyar el trabajo autónomo y consigo la creación de un modelo mental en el estudiante, haciéndolo independiente al programar en entornos sin componentes como este.

12.2.6 Lectura de datos para la asignación de variables en ejecución.

Además de la ejecución de las estructuras básicas de programación, es necesaria la lectura y asignación de datos en ejecución. Esto a razón de la necesidad de ejecutar algoritmos dinámicos para diferentes casos de prueba, ya que muchas veces los datos de salida dependen de datos de entrada y no de valores preestablecidos, como es común en la programación formal.

12.2.7 Impresión de datos en ejecución.

Como se discutió anteriormente, es necesaria la impresión de los resultados o valores de salida después de ejecutar el algoritmo diseñado por el estudiante. De esta manera, en la programación formal, se denotan las salidas de un proceso así

como su culminación. Por otro lado, el prototipo debe ser capaz de expresar dichas salidas en formatos establecidos a solicitud del problema o por el estudiante.

12.2.8 Notificar errores léxicos, sintácticos y semánticos en las expresiones.
Si bien el estudiante debe poseer unos presaberes que correspondan con los fundamentos para la introducción a la programación, como operaciones aritmeticas y logicas, jerarquía de los operadores, tablas de verdad, etc; es necesario indicarle al estudiante si una expresión no es válida en el campo indicado debido a la presencia de errores de algún tipo. Esto con el fin de aproximar al estudiante a un lenguaje formal de programación donde no se admite la compilación de un algoritmo que tenga presencia de desaciertos en las expresiones. Asimismo para indicarle al estudiante que tipo de expresiones son admitidas dependiendo de la estructura, ya que puede proceder en un error de tipo semántico.

12.2.9 Guardar pseudocódigo en un archivo para su posterior uso.
Con esta característica lo que se pretende es facilitar el uso del aplicativo en sesiones interrumpidas, de esta manera a lo largo del curso se pueden ver varias sesiones en la construcción de un mismo algoritmo.

12.2.10 Abrir pseudocódigo desde un archivo para continuar su desarrollo.
A razón del último requerimiento expresado, la manera ideal para la gestión de trabajos o proyectos desarrollados con la herramienta es por medio de archivos. Los archivos facilitan el envío de los algoritmos construidos en la búsqueda de una retroalimentación colaborativa.

Como se puede evidenciar, los requerimientos establecidos son complementarios respecto a cada una de las dificultades y variables causantes del problema. De esta manera, el propósito de cada requerimiento es apoyar al estudiante en estas etapas introductorias para crear modelos mentales que sean fundamentales para asumir nuevos retos y cursos más complejos en entornos más formales.

12.3 REQUERIMIENTOS NO FUNCIONALES

Otra parte esencial del prototipo es el cumplimiento de requerimientos no funcionales, los cuales componen al software de manera implícita y discreta para el

usuario. Sin embargo, corresponden a una parte crucial para la presentación de la herramienta. Es por ello que se plantearon los requerimientos no funcionales a continuación.

Como se mencionó en el alcance de esta propuesta, la herramienta se limita al uso de primitivas de programación (asignación, lectura, escritura, estructuras condicionales, estructuras cíclicas, anidamiento). Haciendo uso de estos para el desarrollo de algoritmos estructurados los cuales estan incluidos en paradigma de la programación orientada a objetos. Por lo tanto, la complejidad de los algoritmos por los cuales se desarrolló esta propuesta son de carácter simple, sin ser extensos mayormente haciendo uso únicamente de un método principal sin invocar a secundarios. El pseudocódigo al ser independiente es posible delimitarlo a 4 tipos de datos (entero, decimal, texto, lógico) con el propósito de acercar al estudiante al contexto de tipación en los lenguajes. Con base a estos tipos de datos se excluyeron vectores o matrices de los mismos. Los operadores básicos involucrados en está propuesta son los siguientes símbolos ||, &&, ==, !=, >, <, <=, >=, !, +, -, *, /, %. Los cuales en el mismo orden representan las operaciones lógicas ó, y, igual que, diferente que, mayor que, menor que, menor o igual que, mayor o igual que y negación. Mientras que los operadores aritméticos representan (en el orden restante) las operaciones de suma, resta, multiplicación, división, módulo.

12.3.1 Lenguaje de programación Java.

Si bien el lenguaje puede tener un impacto poco perceptible para el usuario, el impacto si es notable en sus etapas de desarrollo. En vista de que se contaba con un periodo de desarrollo no mayor a las 14 semanas se decidió trabajar con el lenguaje java el cual cuenta con una amplia documentación en línea y el trabajo de este puede facilitarse debido a la experiencia del desarrollador. Así mismo, Java incorpora una familia de productos/tecnologías de Oracle para la creación de interfaces flexibles, estos componentes son pertenecientes a JavaFX. A comparación con lenguajes como Python o C++, Java brinda la posibilidad de trabajar con componentes gráficos desde las librerías inmersas en el JDK sin utilizar agentes externos. Este nivel de acople fue tenido en cuenta para el desarrollo visual de la herramienta y por tanto para la elaboración de la misma.

12.3.2 Aplicación Standalone.

A pesar de que hoy en día son cada vez más populares las aplicaciones web, la disponibilidad de las mismas restringe al usuario con base a los recursos que se

encuentran presentes en ellas. Recursos tales como banda ancha, capacidad de cómputo en la nube, almacenamiento en la nube, disponibilidad, etc. Debido a que el prototipo subyace del software educativo, no existe la necesidad de actualización, registro ni consulta de información en tiempo real en la misma; pues la herramienta en sí puede describirse como un laboratorio virtual cuyo único propósito es retroalimentar al estudiante con base a los elementos ya abordados en el modelo pedagógico incorporado. Es por esto que la presentación de la solución se adecua de manera pertinente a un modelo stand alone, sin la necesidad de relacionar acciones o procesos entre dos o más máquinas o un sistema centralizado.

12.3.3 Gestor de archivos.

Como se mencionó anteriormente en el levantamiento de requerimientos funcionales, el sistema de almacenamiento el cual opera en la herramienta será por medio de archivos. Si bien una alternativa pudo ser en una base de datos, al no encontrarse el prototipo en un sistema centralizado no tendría sentido la incorporación de la misma; ya que el propósito de estas es unificar y centralizar la información. Es por ello que únicamente se determinó abordar los archivos como medio de almacenamiento, además de ser flexible a la hora de compartir los documentos.

13. DESARROLLO

El presente capítulo tiene como propósito definir, describir y documentar el proceso que se llevó a cabo para el desarrollo del prototipo. Dónde transcurre por dos etapas principales, resumidas en diseño/planificación y construcción. Empezando por la organización y desglose de actividades por medio de la metodología SCRUM, la documentación de modelos y vistas explica el proceso de diseño. Posteriormente se comentarán las tecnologías utilizadas así como los conceptos técnicos y teóricos los cuales componen el prototipo.

13.1 SCRUM

En primera instancia dentro de la metodología SCRUM se definen ciertos roles los cuales caracterizan una parte importante para el desarrollo. A continuación se listaran los roles asignados para el desarrollo de la herramienta basados en los conceptos definidos por Schwaber y Sutherland[96].

13.1.1 Product owner.

En el ejercicio del desarrollo de la herramienta, anteriormente se llevaron a cabo una serie de entrevistas con Giovanni Fajardo Utria (docente a cargo de cursos introductorios a la programación en las instituciones ya declaradas). Entrevistas que permitieron el levantamiento de requerimientos así como un acompañamiento a la hora de desarrollar el producto frente al cumplimiento de los mismos para el desarrollo de la solución. Es por ello que también ejerce el rol de *product owner*.

13.1.2 Development Team y SCRUM Master.

Debido a que la presente investigación es llevada a cabo por un investigador principal, yo (Brandon Alexander Rodriguez Caro) ejerceré los roles de development team y SCRUM master con el fin de gestionar y construir cada artefacto entregable.

Dentro de la metodología SCRUM, como lo definen Schwaber y Sutherland[97] se encuentra una serie de eventos inmersos en los Sprint (periodo de desarrollo para

[96] SCHWABER, Ken y SUTHERLAND, Jeff. La Guía Definitiva de Scrum: Las Reglas del Juego. En: SCRUM GUIDES [sitio web]. Estados Unidos: ScrumGuides. Serie "SCRUM Guide". [Consultado: 24 de abril de 2021]. Archivo pdf. Disponible en: https://scrumguides.org/docs/scrumguide/v2016/2016-Scrum-Guide-Spanish.pdf#zoom=100

[97] Ibíd., p. 8.

la entrega del próximo artefacto) tales como el *Sprint Planning*, *Daily Scrums*, el trabajo de desarrollo, *Sprint Review*, y el *Sprint Retrospective*. Es por ello que a continuación se comentará el proceso que fue llevado a cabo respecto a estos eventos teniendo en cuenta la definición que Schwaber y Sutherland[98] proponen.

13.1.3 Sprints.

En primera instancia se definieron sprints de una semana coincidente con el inicio y fin de cada martes desde el mes de febrero hasta el fin de mes de mayo. Lo cual en total pudo componer 9 sprints. Usualmente estos se encuentran divididos respecto a los requerimientos planteados inicialmente o los que vayan surgiendo en el proceso. Es por ello que se pretende ajustar de manera distributiva cada uno de los requerimientos definidos inicialmente.

13.1.4 Historias de Usuario.

En la metodología SCRUM es común transcribir los requerimientos en forma de *Historias de Usuario* de tal manera en que sean comprensibles para cada uno de los involucrados en el proyecto. Es por ello que se definieron en principio 12 historias de usuario las cuales se ponderaron y organizaron en función de dar valor al cliente/usuario. Este orden fue utilizado para la definición de los *Sprint goals* (objetivos de cada sprint) en principio, en la *Tabla 5* se puede apreciar la disposición de estas historias de usuario.

Tabla 5. Orden de historias de usuario - *Sprint Goals*.

Sprint	**Peso**	**ID**	**Historia de Usuario**
1	3	1	Como estudiante deseo generar y remover estructuras de pseudocódigo para el desarrollo de algoritmos estructurados
1	3	2	Como estudiante deseo ingresar en los campos disponibles las expresiones aritméticas o lógicas para dar sentido al pseudocódigo desarrollado
2	5	3	Como estudiante deseo que el software me notifique los errores léxicos, sintácticos o semánticos en las expresiones para apoyar el proceso de aprendizaje
4	12	4	Como estudiante deseo poder secuenciar mi codigo para luego ver los resultados basados en mi algoritmo

[98] Ibíd., pp. 8-13.

6-7	8	5	Como estudiante deseo ver una prueba de escritorio automática para poder evidenciar mis errores lógicos en el pseudocódigo
4	3	6	Como estudiante deseo poder ingresar mi prueba de escritorio manual para desarrollar un modelo mental de lo que ocurre con mi programa
5-6	12	7	Como estudiante deseo ejecutar mi pseudocódigo paso a paso para poder evidenciar una trazabilidad de la ejecución del mismo
3	1	8	Como estudiante deseo guardar mi pseudocódigo en un archivo para poder tenerlo disponible en cualquier momento
3	1	9	Como estudiante deseo abrir mis pseudocodigos almacenados para poder trabajar ininterrumpidamente en ellos
3	8	10	Como estudiante deseo ver en consola los errores presentados en mis expresiones para denotar en detalle las posibles causas del mismo
8	8	11	Como estudiante deseo ingresar datos en la consola de entrada para poder ejecutar mi pseudocódigo respecto a entradas dinámicas
9	5	12	Como estudiante deseo imprimir los resultados en consola para identificar las respuestas concretas

Fuente: *Autor propia*.

13.1.5 Product backlog.

Una vez definidas las historias de usuario y su priorización, se procedió a definir este como la lista del producto inicial para su desarrollo. Esta lista se compone de 13 historias de usuario (incluyendo los diseños) las cuales subdividen los 9 sprints reflejados en el periodo ya mencionado. Este product backlog se encuentra documentado en GitHub (*https://github.com/users/BrandonRodriguezC/projects/3*).

13.1.6 Sprint planning.

En él se define la planeación concreta para cada Sprint, con lo cual se podrá listar una serie de actividades (*Sprint backlog*) las cuales pretenden desglosar los *sprint goals* así como la entrega del artefacto compuesto por la lista de producto pactada en la reunión anterior. Los *sprints planning* se llevaron a cabo todos los martes a las 7am en el periodo ya mencionado y los sprint backlog se encuentran documentados

en los tableros expuestos en *GitHub (https://github.com/users/BrandonRodriguezC/projects/3)*.

13.1.7 Daily scrum.

Dentro de cada sprint se realizó de manera interpersonal un análisis del desempeño que estaba llevando a cabo el desarrollo del proyecto y se ajustaron tiempos y actividades para cumplir con las metas pactadas.

13.1.8 Sprint retrospective.

Después de cada reunión de *sprint planning* los martes, de manera interpersonal se analizó el desempeño y se gestionó la manera ideal para cumplir con las actividades pactadas.

13.1.9 Sprint review.

Cada viernes durante el periodo estipulado se hizo una reunión de 10 minutos o menos, con lo cual se hizo una revisión de los adelantos y pendientes del sprint. Referente a esto se listaron posibles elementos que serían candidatos al siguiente sprint.

13.2 MODELOS Y VISTAS

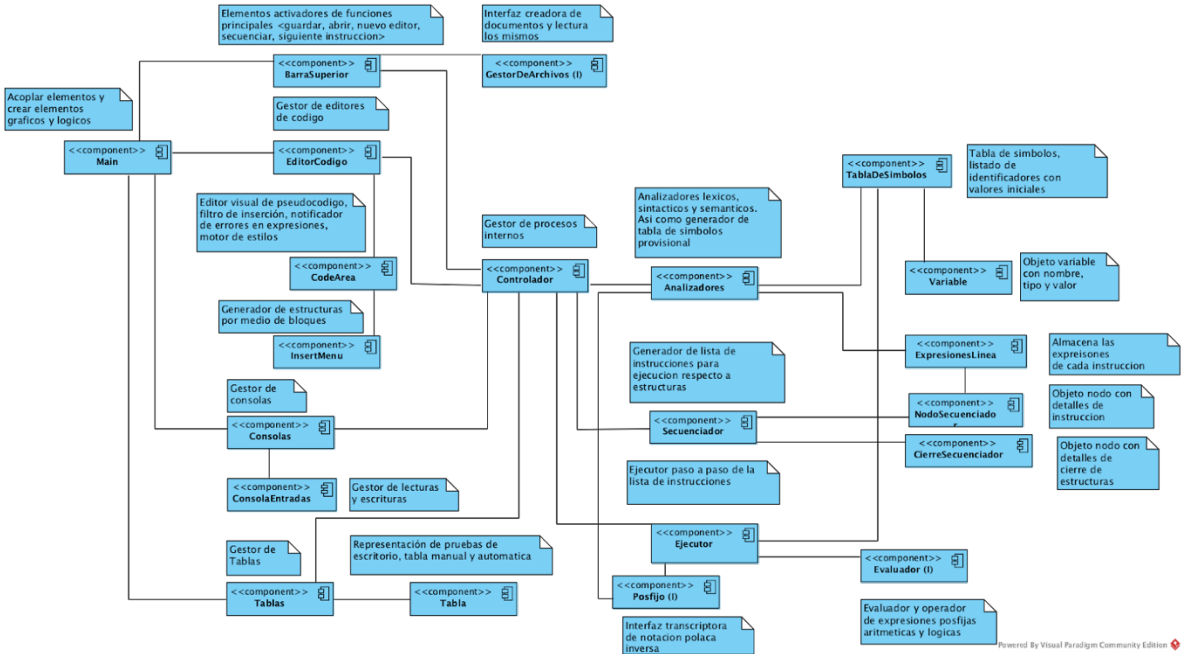
13.2.1 Vista funcional.

Como se menciona en una presentación académica de la Universidad de los Andes[99], el principal propósito de esta vista gira entorno a la descripción de elementos funcionales, incluyendo responsabilidades, interacciones e interfaces. Es por ello que se describe este modelo con el fin de comunicar la capacidad funcional del sistema, definir sus componentes más importantes, definir las estructura interna del sistema y hacer más explícito su diseño. Entre sus atributos encontramos la flexibilidad, la cohesión, el volumen de interacción, el acoplamiento de los componentes y la división de responsabilidades. A continuación, en la *Figura 7* se evidencia dicho modelo y posteriormente su descripción.

---

[99] UNIVERSIDAD DE LOS ANDES: Departamento de Sistemas. Arquitectura de Software: Punto de Vista Funcional [diapositivas]. Repositorio uniandes. Bogotá, 79 diapositivas. [Consultado 24 de abril de 2021]. Disponible en: https://profesores.virtual.uniandes.edu.co/~arti4001/dokuwiki/lib/exe/fetch.php?media=principal:calendario:modulo5-vistafuncional.pdf

Figura 7. Vista funcional del sistema interno de TestDesk.



Fuente: *Autor propia*

El sistema parte de una clase principal la cual es la responsable de inicializar todos los componentes que la integran (tanto visuales como lógicos), de allí parte la división del sistema en 4 áreas de trabajo. La primera es la barra de herramientas la cual se declara en la clase *BarraSuperior*, esta es la encargada de llamar funciones lógicas como el almacenamiento y carga de archivos (función que extiende de la interfaz *GestorDeArchivos*), la creación de áreas de codificación, la limpieza y comparación de pruebas de escritorio, por último la ejecución pausada del pseudocódigo. Otra área integrada es el campo de edición de pseudocódigo que parte de la clase *EditorCodigo* el cual integra las múltiples áreas de pseudocódigo (*CodeArea*), cada área comprende de un motor lógico el cual le permite estilizar el contenido, insertar estructura, filtrar la inserción por teclado, y tomar las expresiones para su analisis lexico, sintactico y semantico. Estas áreas de pseudocódigo incorporan un menú de inserción (*InsertMenu*) el cual evalúa la posición de inserción y construyen el bloque con la indentación requerida para el área de pseudocódigo. La tercer área declarada es la de pruebas de escritorio (*Tablas*), área la cual se compone de una tabla (*Tabla*) generada con las filas y columnas de manera

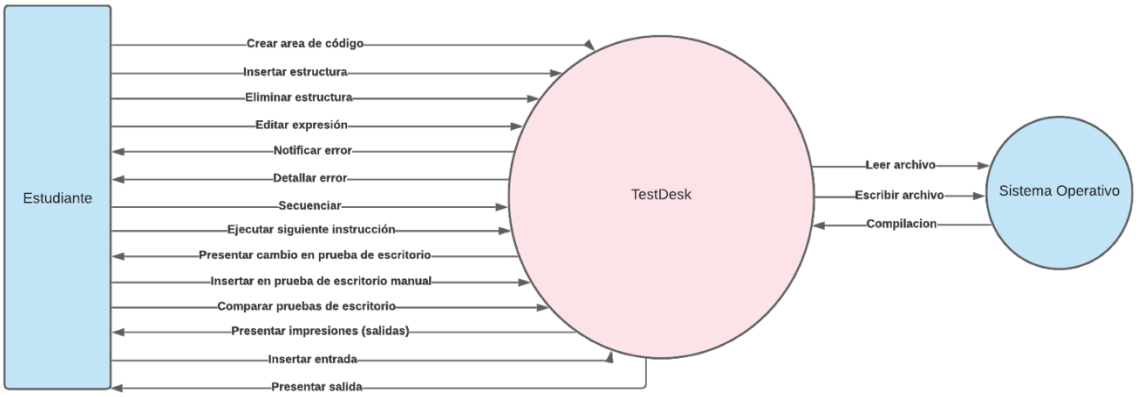
automática para la visualización de cambios en memoria cuando se ejecuta el programa. Por último está el área de *Consolas*, la cual se responsabiliza de denotar información valiosa para el estudiante como errores, registro, además de integrar un área de entradas y salidas (*ConsolaEntradas*) la cual tiene como propósito definir y capturar la información que interactúa con el pseudocódigo. Todos estos componentes se enlazan con la parte lógica por medio de la clase *Controlador*.

Por otro lado, la parte lógica se compone de 3 áreas. La primera (*Analizadores*) permite evaluar las expresiones de manera léxica y sintáctica denotando los símbolos involucrados en el pseudocódigo en *TablaDeSimbolos* por medio de objetos de tipo *Variable*, para el análisis semántico se hace uso de la interfaz *Posfijo* la cual denota las expresiones notación polaca inversa (concepto explicado con mayor detalle en el siguiente subcapítulo) evaluadas por la clase *Evaluador*. Después de su evaluación se almacena la expresión en *ExpresionesLinea* para su posterior uso en ejecución. La siguiente área es *Secuenciador*, la cual se encarga de generar una lista de instrucciones tal que la ejecución pueda tomar sentido bajo el comportamiento de las estructuras, esta lista de instrucciones se genera haciendo uso de objetos tipo *NodoSecuenciador* y *NodoCierre*. Finalmente se encuentra *Ejecutor* el cual se encarga de ejecutar la lista de instrucciones haciendo uso de las interfaces *Posfijo* y *Evaluador* para la evaluación de expresiones necesarias haciendo uso de la tabla de símbolos. Cada una de los cambios realizados en ejecución son denotados en las tablas ya mencionadas en la parte visual, así como la información derivada y errores encontrados mostrados en el área de consolas.

13.2.2 Vista de contexto.

La presente vista pretende representar la interacción entre las entidades y el sistema o los sistemas. Esto con el fin de denotar los límites del mismo, expresando las entradas y salidas desde y hacia los componentes externos. Asimismo, este diagrama permite definir el alcance de manera general para el sistema. En la *Figura 8* se puede apreciar la vista de contexto de la herramienta.

Figura 8. Vista de contexto de TestDesk.

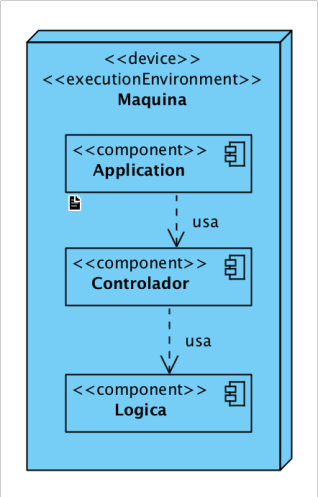


Fuente: *Autor propia*

13.2.3 Vista de despliegue.

Esta herramienta al ser un software de tipo standalone, no requiere un sistema complejo de despliegue. Donde solo se necesita un dispositivo con un ambiente de ejecución, en este caso el ambiente es Java, el cual permite la ejecución de los 3 entornos principales que refieren a los paquetes ya descritos en el diagrama de clases. En la *Figura 9* se puede apreciar la vista de despliegue de este.

Figura 9. Vista de despliegue de TestDesk.



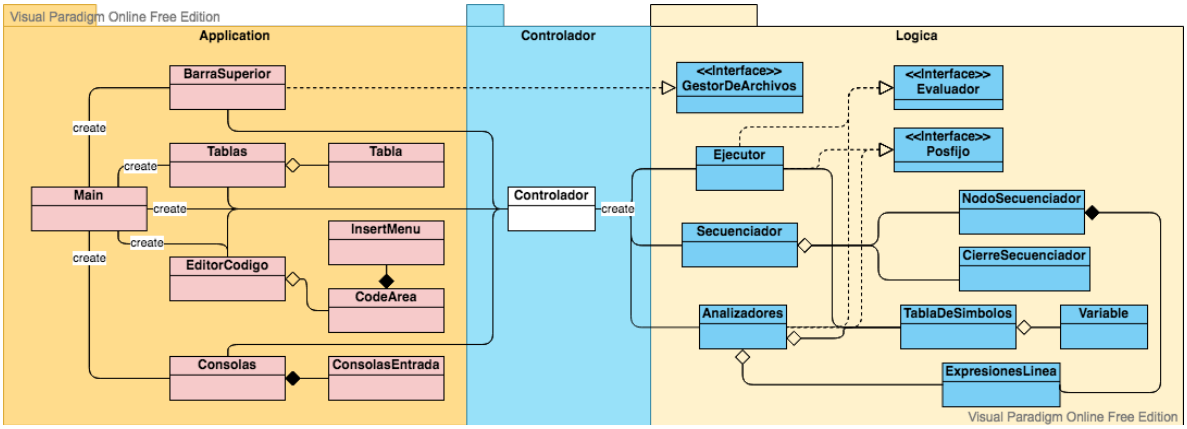
Fuente: *Autor propia*

13.2.4 Diagrama de clases.

Para Gutierrez[100] de la Universidad de los Andes (Venezuela), un diagrama de clases representa la estructura estática del sistema modelado, donde se contemplan las relaciones existentes entre los diferentes componentes y se concentran elementos del sistema de forma independiente del tiempo. A continuación se aprecia una previsualización del modelo detallado de clases, si desea contemplarlo con mayor detalle diríjase a los anexos o visite el enlace (*https://github.com/BrandonRodriguezC/testdesk-documentacion/blob/main/docs/vistas/Diagrama_Clases.png*).

Por otro lado, otra manera de expresar el modelo detallado de clases en brevedad es como se muestra en la *Figura 10*. En él se aprecian 3 paquetes que respetan el esquema modelo-vista-controlador, los cuales se componen de clases que se asocian entre sí en función de interactuar para satisfacer los requerimientos definidos. Estas asociaciones responden a 4 tipos, asociacion (línea sin puntas), herencia de métodos de interfaz (línea punteada con triángulo al final), agregación de una parte a un todo (rombo vacío) y composición (rombo lleno) que no puede ser poseída de manera simultánea. Partiendo de la ISO 15288[101] se puede definir estos entornos como sistemas individuales, donde el sistema de interés final es una composición de sistemas que interactúan entre sí.

Figura 10. Resumen del diagrama de clases de TestDesk.



[100] GUTIERREZ, Demián. 01 UML Diagramas de Clases (UML ilustrado)[diapositivas]. codecompiling. Venezuela. marzo de 2011, 64 diapositivas. [Consultado 24 de abril de 2021]. Disponible en: https://www.codecompiling.net/files/slides/UML_clase_04_UML_clases.pdf

[101] ISO-IEC-IEEE. Systems and software engineering - System life cycle processes, ISO/IEC/IEEE 15288 [en línea]. Estados Unidos: ISO/IEC JTC 1/SC 7Software and systems engineering. 2015. 108 p. [Consultado 24 de abril de 2021]. Disponible en: https://www.iso.org/standard/63711.html

Fuente: *Autor propia*

13.3 SISTEMA

13.3.1 Entorno gráfico.

Para el desarrollo de la herramienta se evaluaron las librerías de componentes gráficos en Java, actualmente no existen librerías tan ricas en dichos componentes por lo cual solo se evaluaron dos familias de elementos gráficos. Una de ellas fue **java.swing** mientras que la otra fue **JavaFX**, para el presente proyecto se decidió trabajar con esta última a razón de su vigencia y alta variedad de componentes gráficos adaptables para los sistemas operativos actuales. Mientras que por otro lado, *java.swing* se encuentra depreciada y poco abordada en aplicaciones actuales. Además de ello, *JavaFX* se encuentra denominada como un entorno gráfico robusto a la hora de incorporarse con componentes externos. Este entorno gráfico se encuentra incorporado en el *java developer kit 1.8* para Java Standard Edition.

Una vez definido el entorno gráfico a utilizar surgió la necesidad de hacer una búsqueda de componentes gráficos que pudieran representar un editor de código estilizado. Para ello fue necesario clasificar los componentes existentes y filtrar únicamente los que funcionan con JavaFx. De esta búsqueda solo surgieron dos candidatos los cuales se pueden apreciar en la *Figura 11*, *RichText* y *StyledTextArea*. Si bien *RichText* es un componente vigente y propiamente documentado, y *StyledTextArea* es un componente poco documentado y depreciado; se decidió utilizar este último. El factor que permitió la decisión fue la visibilidad del cursor cuando la edición del componente se encontraba inhabilitada. Esto a razón de que la herramienta tendría un sistema de generación de bloques asistido, con lo cual únicamente es posible insertar los bloques desde un menú que se despliega al dar click derecho y se agrega en la posición del cursor, así como ingresar libremente donde sea requerida una expresión (tema que se abordará más adelante).

Figura 11. Componentes gráficos de editores de código estilizados (RichText a la izquierda, StyledTextArea a la derecha).

```
1 println("Hello");
2 int x = 42;
3 int y = x + 8 * 2 / (3 - 1);
4
5
```

```
1 public class Sample {
2
3 }
```

Fuente: *Autor propia*.

13.3.2 Sistema de generación de bloques.

En cumplimiento al requerimiento pre-establecido, la herramienta sería un entorno de desarrollo de algoritmos limitado y asistido, por lo tanto no permitirá la inserción libremente de texto en campos no requeridos. Es por ello que se creó un filtro el cual determinará según la posición del cursor si el componente adquiere la característica editable o no. Dicho filtro funciona al identificar los rangos de los campos de libre edición (campos que se componen de nombres de variables, expresiones aritméticas o lógicas) y verificar si el cursor se encuentra dentro de alguno de esos rangos.

En el proceso de estilizado al insertar una estructura básica de programación, se debe reconocer cada elemento perteneciente al pseudocódigo. Para ello se utilizaron las clases Pattern y Matcher de la librería *java.util.regex*, las cuales facilitaron la búsqueda de elementos en el texto. Es por ello que se definieron expresiones regulares (patrones de escritura) los cuales identificaran las unidades sintácticas y encima lo clasificará según determinado grupo. Entre los grupos se definieron palabras clave, tipos de datos, patrones de paréntesis, operador de asignación, signo de cierre, apertura y cierre de bloques, estructura de comentario, y por último método de lectura y escritura. Dados estos grupos, a la hora de identificar el elemento en el texto y posteriormente clasificarlo, se asigna un estilo y se identifica su función dentro del pseudocódigo. De esta manera se podían identificar los campos de libre escritura y los rangos de estos. El proceso de inserción se puede evidenciar en la *Figura 12*.

Figura 12. Proceso de inserción de estructuras (campo habilitado a edicion libre en amarillo)

Fuente: *Autor propia*

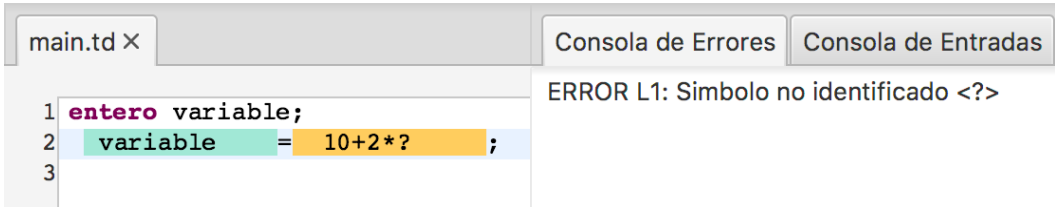
13.3.3 Analizador léxico.

En los campos de libre escritura para dichos identificadores (nombres de variables) o expresiones aritméticas o lógicas, nace la necesidad de la creación de un evaluador léxico, sintáctico y semántico. Dependiendo del tipo de campo libre se deben asignar determinadas propiedades al evaluador el cual dará respuesta basado en dichos argumentos. Evaluador el cual determina si dicha expresión era válida para el campo al cual pertenece y si su composición es correcta. Como lo explica Alfonseca[102], para lograr un análisis léxico de la expresión es necesario un autómata finito determinista (concepto mencionado en el capítulo 7.1.4) este autómata recorre la expresión y se encarga de dividirla en unidades sintácticas. Desde un análisis léxico en caso de no reconocer el elemento dentro del diccionario preestablecido se denota un error en la expresión. Nuevamente se utilizaron las clases Matcher y Pattern las cuales permiten un apoyo a la hora de identificar las unidades sintácticas. En este caso las expresiones regulares establecidas como unidades sintácticas en el diccionario fueron datos de tipo entero, real, lógico, texto, paréntesis, operadores, espacios, comas, identificadores y palabras restringidas. De esta manera al evaluar la expresión en caso de encontrar diferencias del último índice del último elemento reconocido y el primer índice del nuevo elemento reconocido se procederá a notificar un error de tipo léxico. Del mismo modo se

[102] ALFONSECA, Op. cit., p. 65.

notifica el error en caso de que encuentre un elemento perteneciente a un grupo restringido. Un ejemplo de su notificación se puede ver en la *Figura 13*.

Figura 13. Notificación de error léxico.



Fuente: *Autor propia*

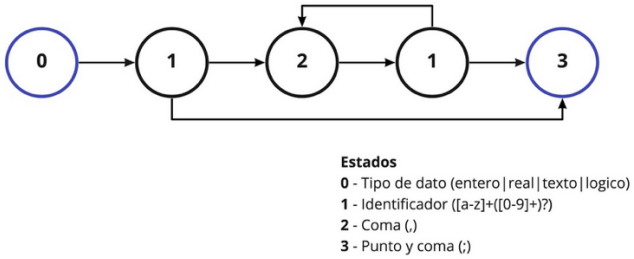
Las propiedades asignadas para el analizador, además de permitir la evaluación de campos libres de diferentes tipos, permite insertar identificadores en la tabla de símbolos (concepto explicado más adelante). La inserción de identificadores en la tabla de símbolos es vital para el análisis de expresiones posteriores, bien sea un análisis de tipo léxico, sintáctico o semántico; es importante tener conocimiento de las variables declaradas para no alertar falsos errores.

13.3.4 Analizador sintáctico.

Nuevamente, Alfonseca[103] explica el uso de autómatas finitos deterministas como medio de análisis sintáctico para un texto, expresión o conjunto gramatical. A diferencia del análisis léxico, el análisis sintáctico pretende evaluar la composición de una expresión la cual posee todas sus unidades sintácticas correctas. Los autómatas finitos deterministas tienen la capacidad de definir estados respecto a las unidades sintácticas. De esta manera es posible determinar si la transición de un estado a otro es correcta. En la *Figura 14* se puede evidenciar un ejemplo de autómata determinista finito para la declaración de variables, expresión la cual corresponde a un campo libre compuesto y de declaración (propiedades vitales para el analizador construido).

Figura 14. Ejemplo de autómata determinista finito para la declaración de una o varias variables.

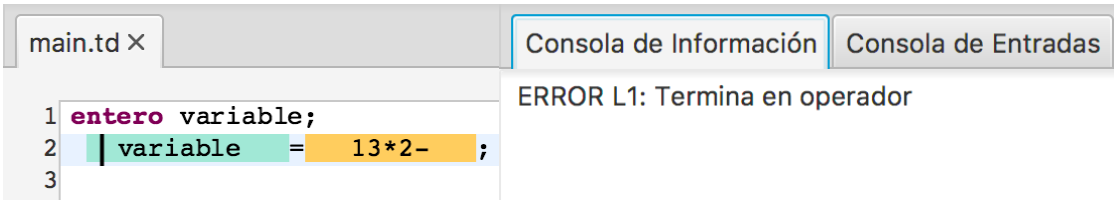
[103] ALFONSECA, Op. cit., p. 90.



Fuente: *Autor propia*

Como se puede ver en la imagen, para que una declaración de variables compuesta pueda ser válida debe transitar por al menos 3 estados. Cada nodo puede representar un carácter, una secuencia de caracteres o una expresión regular, dependiendo del grado de composición que se esté evaluando. En este caso cada nodo de estado 1 representa una expresión regular, que a su vez son autómatas finitos deterministas que se manejaron desde la clase principal *java.regex*. Si bien el autómata estaría en completitud con los 4 estados, únicamente se evaluó y se integraron los estados 1 y 2 (color negro), esto a razón de que el sistema generador de bloques asegura la completitud de unidades sintácticas para los nodos de estado 0 y 3 (color azul). Esto siendo únicamente un ejemplo de las múltiples composiciones presentes en los campos libres disponibles, los cuales pueden resultar en autómatas deterministas complejos o no deterministas complejos; debido a la presencia de expresiones lógicas y aritméticas, siendo estas mismas mucho más complejas si existe la presencia de paréntesis los cuales están compuestos de otras expresiones internamente. En el caso de que exista una transición de estados no válida, sea por ejemplo de 0 a 2, automáticamente se descarta el continuo análisis y se declara el error (ausencia de estado tipo 1 o ausencia de identificador previo a la coma). En la *Figura 15* se puede apreciar un ejemplo de error sintáctico determinado por el autómata de la aplicación.

Figura 15. Notificación de error sintáctico.



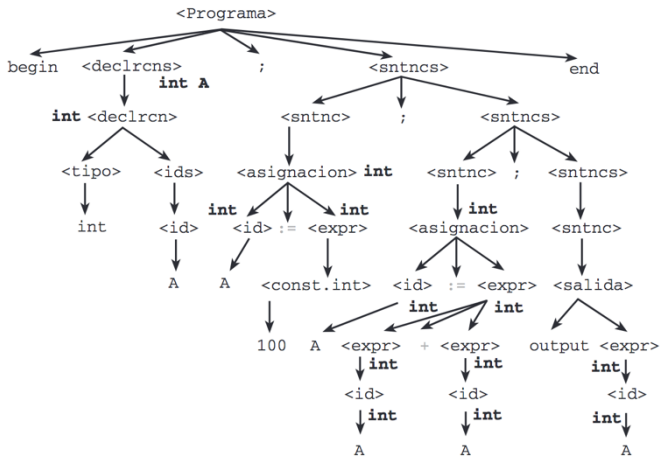
Fuente: *Autor propia*

13.3.5 Analizador semántico.

Al determinar la presencia de una expresión válida en términos léxicos y sintácticos es necesario un análisis semántico, el cual permita verificar la validez semántica. Es decir, que sea válido el sentido de la expresión. Nuevamente se analiza únicamente los campos libres a razón de que el generador de bloques asegura la cohesión completa (sintáctica, léxica y semántica) de la estructura a excepción de la expresión, ya que esta puede variar. Alfonseca[104] denota que la salida de este analizador puede recaer en un árbol del análisis del programa con anotaciones como el de la *Figura 16*. Análisis el cual tiene en cuenta aspectos como:

- El uso de un identificadores declarados previamente.
- En las expresiones, los operandos respetan las reglas sobre los tipos de datos permitidos por los operadores.
- El resultado de una expresión no coincide con el dato asignado.
- Verificación de operadores respecto a las estructuras y sus tipos correspondientes.

Figura 16. Ejemplo de árbol de análisis del programa con anotaciones.



Fuente: Alfonseca (Capítulo 5.1) Compiladores e Intérpretes

[104] ALFONSECA, Op. cit., p. 193.

Asimismo, Alfonseca[105] declara que este resultado o análisis puede variar ya que es bien sabido que diferentes intérpretes y compiladores pueden analizar estos aspectos en etapas anteriores como el analisis lexico y sintactico, lo cual puede resultar en una mezcla de estos. Que en este caso, por ejemplo, se evalúa la declaración previa de un identificador en caso de estar inmerso en una expresión desde el análisis léxico. Para el presente proyecto, en vista de que se están analizando los campos libres (expresiones y declaraciones) de manera individual, se decidió utilizar otro método de análisis el cual determina la validez semántica sin generar un árbol como el mencionado. Esto a razón de que se tuvieron determinados aspectos en cuenta en los análisis léxicos y sintácticos, por lo cual únicamente fue necesario un análisis semántico de segundo paso.

13.3.5.1 Notacion polaca inversa.

Alfonseca[106] lo denota como una de las alternativas de código intermedio o análisis de segundo paso, ya que otra alternativa son las cuádruplas (tema que no se abordará ya que no se incorporó en el proyecto). La notación polaca inversa o notación postfija tiene el propósito de transcribir la expresión en un orden tal que para el sistema le sea más claro operar respetando la jerarquía de los operadores. En la *Tabla 6* se puede evidenciar un ejemplo de transcripción en notación polaca inversa.

Tabla 6. Ejemplos de expresiones en notación polaca inversa.

Expresión	Notación sufija
`a*b`	`ab*`
`a*(b+c/d)`	`abcd/+*`
`a*(b+c*d)`	`ab*cd*+`

Fuente: Alfonseca (Capítulo 6.2.1) Compiladores e Intérpretes

Por medio de un algoritmo de transformación de notación infija a postfija se logró este objetivo haciendo uso de tecnologías como *java.util.Stack*, *java.util.List*

[105] ALFONSECA, Op. cit., p. 194.
[106] ALFONSECA, Op. cit., p. 259.

y *java.util.ArrayList*. Donde se declaró una jerarquía como la que se puede apreciar en la *Tabla 7*.

Tabla 7. Jerarquía de los operadores.

Operador	Prioridad
||	1
&&	2
== != > < <= >=	3
+ -	4
* / %	5
!	6
( )	7

Fuente: *Autor propia*

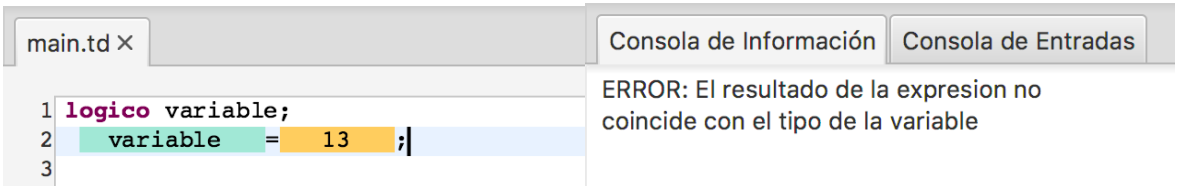
13.3.5.2 Evaluación de notación polaca inversa.

Posteriormente a la transcripción, se evalúa cada expresión únicamente respecto al tipo de dato generado tras cada operación. La evaluación de estas notaciones procede en la lectura de la expresión, almacenando los operandos en una pila (*stack*), y encontrado un operador se recogen los últimos 2 operandos y se computará respecto al operador; una vez computado se almacena el resultado en la pila nuevamente. De esta manera es posible determinar si la expresión es adecuada para el campo donde se indica. Ejemplo de esto es la validación de que el resultado de la operación de una expresión en un campo condicional (sea una estructura *if* o *while*) coincida con el tipo de dato lógico en su resultado. De esta manera se puede corroborar si la expresión es correcta semánticamente.

En principio, los valores al ser evaluados desde la notación postfija poseen un inicial (cero, falso o texto vacío dependiendo del tipo de variable). Pero la evaluación en el momento de la ejecución, las variables si toman un sentido y valor respecto al entorno dinámico donde se desarrollan, más sin embargo la notación polaca inversa

es la misma en el momento del análisis semántico. En la *Figura 17* se puede apreciar un ejemplo de notificación de error semántico en el aplicativo.

Figura 17. Notificación de error semántico.



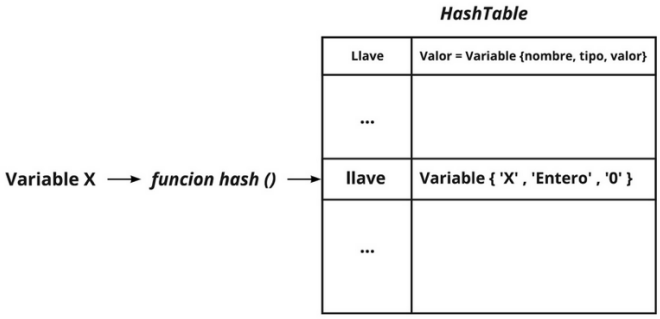
Fuente: *Autor propia*

13.3.6 Tabla de símbolos.

La tabla de símbolos, es una tabla encargada de almacenar cada uno de los identificadores que se encuentran en el pseudocódigo. Los símbolos para esta ocasión representan las variables o identificadores en el algoritmo. Es importante tener registro de estas para reconocer la variable, su tipo y valor en el momento de la evaluación de las expresiones. Para Alfonseca[107], existen diferentes maneras de albergar este registro bien sea desde una lista doblemente anidada hasta un árbol binario, más sin embargo su estructura refiere a una parte importante del intérprete debido al método de búsqueda que incorpora para el uso de identificadores registrados. Es por ello que el método más eficiente y veloz para la búsqueda de estos es haciendo uso de una *HashTable*, la cual es una estructura en forma de tabla cuyo índice de almacenamiento recae en una llave para acceder su valor. Esta llave proviene de una particularidad del dato que se esté llamando la cual debe ser única para los elemento almacenados, en este caso puede ser el nombre ya que en los lenguajes formales de programación no es posible declarar la misma variable en el mismo entorno 2 veces. Esta particularidad no siempre es identificable por lo cual debe tratar el dato por medio de una función o método *hash*, el cual permite generar la llave correspondiente. Para el presente desarrollo se incorporó una tabla símbolos la cual en la *Figura 18* se puede apreciar un ejemplo visual. Esta tabla cumple una función primordial en el análisis léxico, en el cálculo de expresiones en notación polaca inversa y en el registro de la prueba de escritorio, ya que con ella se accede al valor de dicha variable y le hace posible asignar nuevos valores.

[107] ALFONSECA, Op. cit., p. 34.

Figura 18. Ejemplo de tabla de símbolos usando una *HashTable*.



Fuente: *Autor propia*

13.3.7 Secuenciador.

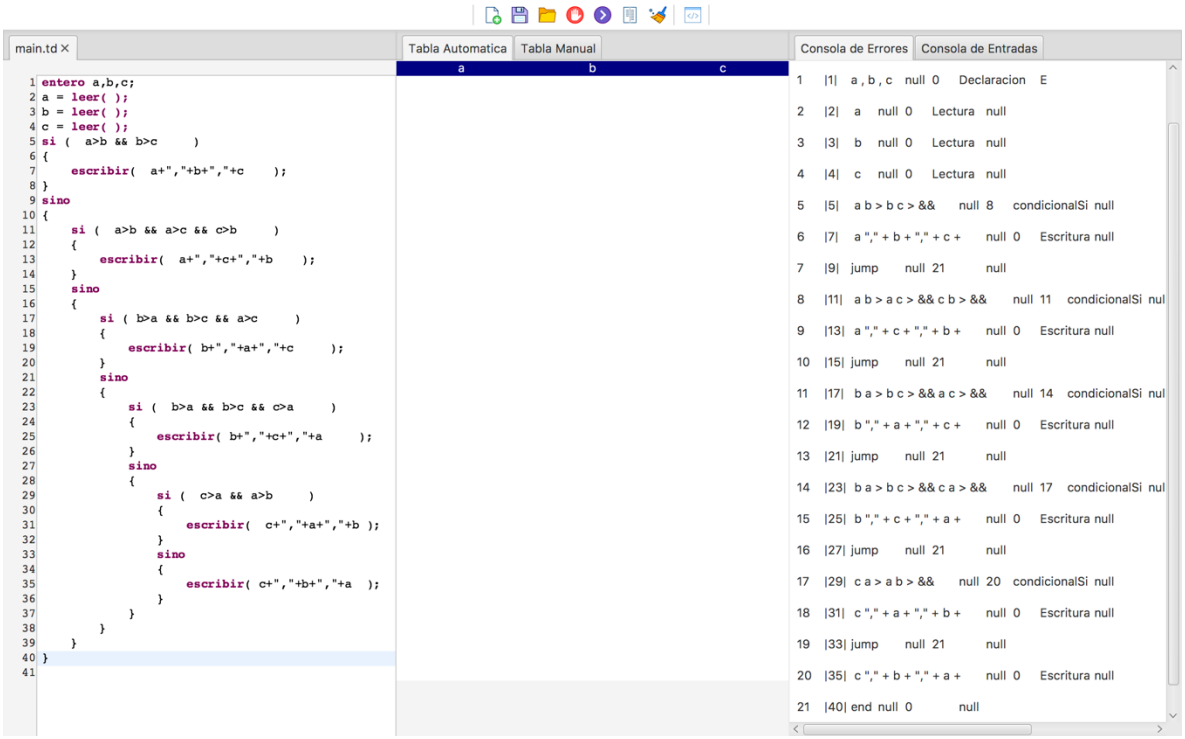
En esta etapa, una vez validado el algoritmo en términos léxicos, sintácticos y semánticos, se procede a secuenciar el pseudocódigo el cual es un proceso que en términos formales transcribe el algoritmo en lenguaje ensamblador. Este proceso suele ser riguroso ya que el resultado de la transcripción es una serie de instrucciones (granulares) dependientes del computador, microprocesador, microcontrolador u otro circuito integrado para el cual se esté desarrollando ya que este puede variar en el número de bits que constituye la arquitectura de la CPU. Este lenguaje ensamblador es la representación cercana al lenguaje máquina. Aun así, debido a que la herramienta expuesta constituye un intérprete y no un compilador. Para Alfonseca[108] un compilador y un intérprete, difiere en su ejecución, ya que un compilador recorre el algoritmo en lenguaje máquina sin analizar cada expresión, contrario a un intérprete. Además de ello, teniendo en cuenta que se está elaborando en un lenguaje de alto nivel (Java); se desarrolló un simulador de conjunto de instrucciones compilado. Cabe aclarar la diferencia pues el propósito de este simulador es reproducir la ejecución del algoritmo manteniendo el entorno en un alto nivel respecto al procesador, es decir, excluyendo el lenguaje máquina o lenguajes intermedios. Es por ello que se generaron instrucciones simples que simularán este proceso netamente para Java.

Para poder reproducir estructuras básicas de alto nivel como lo son condicionales y ciclos (a nivel de hardware y en lenguaje ensamblador) no es posible reproducir limpiamente en el concepto teórico la composición de estos. Es por ello que internamente se simula su ejecución con una lista de instrucciones que incluyan

[108] ALFONSECA, Op. cit., p. 319.

saltos. Estos saltos son ejecutados dependiendo del resultado de las expresiones. En donde un condicional o un iterador condicional tendría definido su salto dependiendo del resultado de la expresión que lo compone. Por el contrario, en una iteración definida (ciclo *for*) internamente opera bajo la condición intermedia la cual define si el índice aún cumple con la cláusula mayor a 0. En la *Figura 19* se puede apreciar un ejemplo de pseudocódigo y la lista de instrucciones generada partiendo del mismo.

Figura 19. Ejemplo de pseudocódigo y de lista de instrucciones del mismo.



Fuente: *Autor propia*

En la *Figura 19* se puede apreciar una lista de instrucciones en el área de *Consola de Errores* la cual se constituye de 7 campos, el primero es el número de la instrucción, el segundo es el numero de linea al cual corresponde, el tercer y cuarto campo corresponde a las partes de la expresión (por ejemplo, en una asignación existen dos partes mientras que en una declaración solo existe una parte), el quinto campo corresponde al salto, posteriormente está el campo donde se denota el tipo de instrucción y finalmente, de ser una declaración, el último campo constituye el primer caracter del tipo de dato declarado. Para la generación de esta lista de

instrucciones se utilizaron objetos de 2 tipos, nodos secuenciadores y nodos de cierre. Donde los nodos secuenciadores representan la instrucción granular, mientras que los nodos de cierre representan los saltos indicados.

13.3.7.1 Nodo secuenciador.

Los nodos secuenciadores se componen en este sistema de atributos como número de línea, número de línea para el salto, número de instrucción, tipo de instrucción, primera y segunda parte de la expresión. Para las instrucciones que corresponden a bloques condicionales o iterativos, el valor del salto equivale al nodo de cierre (concepto explicado a continuación) mientras que para instrucciones simples como la asignación o declaración este valor constituye 0 ya que posee fin en la misma línea.

13.3.7.2 Nodo de cierre.

Los nodos de cierre expresan el fin de un bloque, siendo este útil para indicar el fin de un condicional simple o compuesto, o el fin de alguno de los iteradores. Siendo este vital para el retorno del inicio del bloque e identificar si este ha culminado en su evaluación. Por ejemplo, al ejecutar la última iteración de un ciclo condicional es necesario verificar si la cláusula declarada en el inicio del bloque sigue siendo válida. Este nodo posee dos atributos, el número de instrucción y el tipo que corresponde al cierre.

Una vez explicado esto, la generación de la lista de instrucciones parte del pseudocódigo donde se analiza cada línea y se limpian las líneas vacías. En el análisis, si la línea comprende a una inicio de bloque entonces corresponde a un nodo de cierre y nodo secuenciador. Se procede a instanciar los nodos, el nodo secuenciador se adjunta a la lista de instrucciones mientras que el nodo de cierre se integra en una pila. En el momento de encontrar una línea que indique el cierre de un bloque, entonces se depura el último nodo de la pila y se le indica el salto (índice actual más uno de instrucciones) a la instrucción que denotaba el inicio del bloque, posteriormente se integra un nodo secuenciador con el tipo JUMP el cual en su atributo de salto indica el valor que poseía el nodo depurado de la pila. De esta manera en la ejecución, el intérprete se encarga de la evaluación de expresiones y obedece según su resultado a continuar con la siguiente instrucción o al salto indicado.

13.3.8 Ejecutor.

Para la simulación de ejecución, se recorre la lista de instrucciones y dependiendo del tipo de instrucción se evalúa 1 o 2 campos. La evaluación de cada expresión se realiza con la última tabla de símbolos, es decir, la última que registre un cambio. De esta manera se asume el valor de las variables en la instancia actual. La evaluación de las expresiones recae en la evaluacion postfija que se vio anteriormente. En la evaluación de estas notaciones se verifica que los operando sean calculables respecto al tipo de dato esperado, también se verifica que cumpla con las normas aritméticas o lógicas. Por ejemplo la detección de una expresión incalculable *a/b* donde *a* es igual a 1 y *b* es igual a 0. Posteriormente, al obtener valores cuyos tipos de datos sean válidos para la expresión, se procede a tratar con la tabla de símbolos o a decidir el salto de la lista de instrucción (esto depende del tipo de instrucción).

13.3.8.1 Declaración.

En el momento de encontrar una declaración, se busca la variable en la tabla de símbolos y se inicializa su valor. Este valor de inicio depende del tipo de dato al que pertenezca. Una vez asignado el valor, se indica en la prueba de escritorio automática (en la columna de la variable y en la fila de la línea) se indica el valor nuevo valor el cual asume dicha variable.

13.3.8.2 Asignación.

En este caso, busca la variable indicada en el 1 campo y se almacena el tipo de dato correspondiente a la variable encontrada. Posteriormente se evalúa la notación postfija la cual está declarada en el 2 campo, si la evaluación de esta expresión es exitosa entonces se asigna el nuevo valor en la tabla de símbolos según el identificador, se indica en la prueba de escritorio el nuevo valor.

13.3.8.3 Condicional e Iteración condicional.

A razón de que esta instrucción expresa un bloque condicional, se evalúa la expresión postfija en el único campo. Si la evaluación de esta expresión es exitosa y su resultado es de tipo lógico, se decide el salto que tendrá el ejecutor, bien sea en la siguiente instrucción o en la instrucción que procede al cierre del bloque.

11.3.8.4 Iteración definida.

En este caso particular se toma el valor simple expresado en el 1 campo de tipo entero positivo, se resta 1 unidad y se asigna en el campo su nuevo valor. En caso

de que este nuevo valor sea igual a 0, se procede a indicar el salto a la instrucción posterior al fin del bloque, se asigna su valor inicial (denotado en la tabla de símbolos de manera oculta) y efectúa el salto. De lo contrario, solo se asigna el valor en la expresión y se continúa en la siguiente línea. No se declara ningún valor en la prueba de escritorio.

13.3.8.5 Lectura.

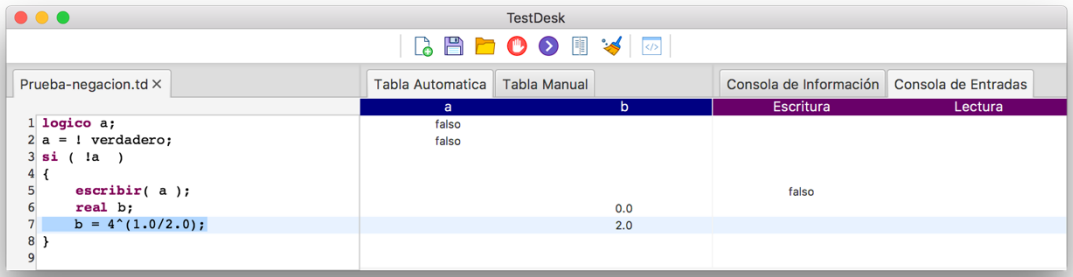
En caso de que la expresión sea de tipo lectura, se verifica si el la línea correspondiente a la línea de la instrucción en la columna de Lectura en la consola de entradas posee algún valor, en caso contrario se frena la ejecución hasta no cumplir esta cláusula. En caso de estar lleno el campo, se hace búsqueda de la variable perteneciente a la instrucción (denotada en el 1 campo), se almacena el tipo de dato, se transcribe la expresión del campo de lectura en notación polaca inversa, se evalúa y se verifica el tipo de dato del resultado respecto al tipo de dato de la variable. Si este proceso es exitoso, se asigna a la variable su nuevo valor en la tabla de símbolos y se registra el cambio en la prueba de escritorio.

13.3.8.6 Escritura.

Si la instrucción es de tipo escritura, solo se evalúa la expresión en notación polaca inversa del método ubicada en el 1 campo, y su resultado se convierte en un tipo de dato texto. Posteriormente se registra su resultado.

En la *Figura 20* se puede ver una instantánea en el momento de ejecución.

Figura 20. Instantánea de ejecución.



Fuente: *Autor propia*

14. VALIDACIÓN

El presente capítulo pretende documentar el proceso de pruebas al cual se le aplicó a la herramienta, denotando sus resultados. Para ello se le aplicaron pruebas funcionales de diferentes tipos en los cuales se denotaron *bugs* y posteriormente a la corrección de los mismos. Las pruebas unitarias se realizaron a lo largo del desarrollo, sin embargo estas no fueron documentadas debido al periodo de desarrollo y la alta probabilidad de cambio. Asimismo, se aplicó una prueba en un grupo focal de 7 estudiantes de ingeniería en sistemas para poder denotar la percepción en el usuario, evaluando factores que se detallarán más adelante.

14.1 PRUEBAS FUNCIONALES

Entre las primeras pruebas funcionales que se aplicaron en la herramienta fue la detección de errores léxicos, sintácticos y semánticos. Estas pruebas se refieren a la evaluación de expresiones correctas y erradas en los diferentes bloques que se encuentran en la herramienta. Para ello se estableció un set de pruebas de 20 expresiones con resultados esperados. Entre los resultados esperados no se encontraron incongruencias debido a que este analizador lexico-sintactico-semántico, fue una de los componentes más trabajados a lo largo de la construcción del aplicativo. A continuación se denota un resumen del set de pruebas aplicado y el tipo de error esperado/notificado en la *Tabla 8*.

Tabla 8. Set de pruebas de lexicas .

Índice	**Expresión**	**Tipo**
1	5 + 6 – 8 * ç	Léxico
2	58 +* 13	Léxico
3	$58 - 1	Léxico
4	58 *= 14	Léxico
5	58 % sino	Léxico
6	[512	Léxico

7	- 58 / 14	Sintáctico
8	/14	Sintáctico
9	5 + 6 *	Sintáctico
10	(7 –) * 6	Sintáctico
11	67 5 –12 + 85 % 3 – 6	Sintáctico
12	45 + 93 / 7 % + 69	Sintáctico
13	5 + 6 – 8 **[espera real]**	Semántico
14	58.1 + 14.3 **[espera lógico]**	Semántico
15	5.318 - 1.14 **[espera texto]**	Semántico
16	falso || verdadero **[espera real]**	Semántico
17	verdadero && falso **[espera entero]**	Semántico
18	verdadero || verdadero && falso **[espera texto]**	Semántico
19	"hola, usuario "+ 13 **[espera entero]**	Semántico
20	"¡Hola mundo!" **[espera texto]**	Semántico

Fuente: *Autor propia*

Posteriormente a estas evaluaciones de análisis, se procedió a evaluar la operación de las expresiones, es decir, a verificar si los respectivos resultados concretaban el verdadero resultado de la expresión. En principio se realizó bajo expresiones enteras, y por ende reales, las cuales permitieron evaluar adecuadamente cada composición en las expresiones como se puede ver en la *Tabla 9*. Debido a que en primera instancia no se obtuvo el resultado esperado, se decidió entrar en un proceso de depuración el cual con cada cambio entre las diferentes entregas se le aplicaba nuevamente el set completo.

Tabla 9. Set de pruebas de expresiones aritméticas.

	Expresión	**Resultado Java (real)**	**Resultado TestDesk (real)**	**Resultado Java (entero)**	**Resultado TestDesk (entero)**
1	5 + 6 – 8	3.0	3.0	3	3

2	58 + 14	72.0	72.0	72	72
3	58 - 14	44.0	44.0	44	44
4	58 * 14	812.0	812.0	812	812
5	58 % 14	2.0	2.0	2	2
6	58 / 14	4.0	4.0	4	4
7	58.0 / 14	4.0	4.0	4	4
8	5 + 6 * 3	23.0	23.0	23	23
9	(7 – 4) * 6	18.0	18.0	18	18
10	67 * 5 –12 + 85 % 3 – 6	318.0	318.0	318	318
11	45 + 93 / 7 % 8 + 69	119.0	119.0	119	119
12	89 + (65 – 32) * (212 % (5 + 7) – 4) % 43 *21	152.0	152.0	152	152
13	–(17-3)/8+1	0.0	0.0	0	0
14	45*7-6*14%6+5*8-12	343.0	343.0	343	343
15	12+85-6*7+85+36-78*9	-526.0	-526.0	-526	-526
16	((85-43*6)%15-25*12	-308.0	-308.0	-308	-308
17	25-(48*25/(6+47)*3+78)	-120.9245283018	-120.924528301886	-119	-119
18	45-8+96+71-25+53+8+32-9	263.0	263.0	263	263
19	78/3*96%8*25/12*96	0.0	0.0	0	0
20	8*5+12/4	43.0	43.0	43	43
21	123%10*100+123/10%10*10+123/100	324.23	324.23	321	321
22	4589%100*100+4589/100	8945.0	8945.0	8945	8945

Fuente: *Autor propia*

Entre los problemas encontrados, principalmente estaba el valor en la jerarquía de los operadores. También se determinó el problema en la depuración de la pila a la hora de evaluar la expresión postfija, esto debido a que se estaba depurando toda la pila y a razón de ello no se evaluaba a completitud la expresión. Junto con errores a la hora de tomar los operandos cuando existe un operador unitario.

Otra perspectiva para el tipo de pruebas está en las expresiones lógicas, las cuales se componen de resultados verdaderos o falsos dependiendo de su evaluación haciendo uso de tablas de verdad y operadores comparativos. Para esta funcionalidad se evaluó con un set de pruebas en conjunto con expresiones aritméticas, partiendo de valores enteros como: a = 5, b = 12, c = 0, d = 23, e = 15, f = 4.

En función de estas declaraciones, se aplicaron los siguientes casos de prueba presentes en la *Tabla 10*.

Tabla 10. Set de pruebas de expresiones lógicas.

	Entrada	**Resultado**	**Salida**
1	8 > 6 || 9 > 8 && 7 > 19	VERDADERO	VERDADERO
2	(8 > 6 || 9 > 8) && 7 > 19	FALSO	FALSO
3	(3+5)/3+9%4 >= 9+((5+6)*3) || 18%4*9-4 != 15*4/3-6 && 12*6-18 == 45-52+8	FALSO	FALSO
4	a>=b-f&&c<d-e	FALSO	FALSO
5	(57-d+f<=a*(25-b)+14||67-d>(f*12)+15)&&a>c*10	VERDADERO	VERDADERO
6	57-d+f<=a*(25-b)+14||(67-d>(f*12)+15&&a>c*10)	VERDADERO	VERDADERO
7	e*(d-f)>45+c&&a>=85-d*a||27<=a*e-6	VERDADERO	VERDADERO
8	falso || verdadero && verdadero && falso || falso && verdadero	FALSO	FALSO

Fuente: *Autor propia*

Exitosamente se desarrolló de manera adecuada la evaluación de estas expresiones. La evaluación de expresiones cuyo producto resulta en la concatenación de valores para un resultado de tipo texto se llevó a cabo de manera independiente, identificando problemas como el intento de operar valores que se encuentran dentro de la expresión regular del tipo de dato textual al contener símbolos como = o ! los cuales son comunes en algunas salidas. Estos problemas fueron depurados y tenidos en cuenta dentro del sistema.

Así mismo, se determinaron errores en componentes tales como el área de codificación. Esto a razón de que el componente trabajado ha sido apreciado desde el 2015 (razón explicada en el capítulo anterior), aun así se intentó mitigar la gran mayoría de estos. Problemas como el control del cursor dentro del área de codificación o estilizado parcial del contenido. Estas pruebas se elaboraron de manera simultánea haciendo la construcción de los algoritmos puestos estos en ejecución. Todos estos ejemplos fueron almacenados en la carpeta (*https://github.com/BrandonRodriguezC/TestDesk/tree/main/Pruebas*). En él se encontraron errores en la limpieza de espacios, la espera de lectura después de haber culminado la ejecución, la construcción indentada de los bloques, la salida en el método de escritura, la inicialización de la tabla de símbolos después de una ejecución, el cálculo de espacios dentro de las columnas, remoción de bloques, indentación dentro del bloque, inserción de saltos de línea el lugares restringidos, filtro adecuado de inserción de expresiones, notificación de errores en la consola de información, entre otras. Todos estos problemas fueron resueltos sin mayor dificultad ya que solo requería el llamado de métodos ya creados los cuales no se tuvieron en cuenta en primera instancia. Así mismo, se integró un código que analizaba de manera más adecuada las particularidades del caso de uso. Cada una de las pruebas ejecutadas fue evaluada y corregida en caso de no cumplir exitosamente el resultado. Una vez corregido, se aplicaban nuevamente cada una de las pruebas.

14.2 GRUPO FOCAL

Para el desarrollo de las pruebas se evaluó un grupo focal de 7 estudiantes pertenecientes al programa de ingeniería en sistemas que estaban o ya habían cursado el curso introductorio a la programación, 3 de ellos son de la Universidad Minuto de Dios y 4 de ellos son de la Universidad Piloto de Colombia. En la reunión del grupo focal se discutió el contexto de la investigación llevada a cabo, mencionando el planteamiento del problema, la justificación y el alcance. Posteriormente se dio introducción a la herramienta y al uso de esta, explicando sus principales aspectos y funcionalidades. Una vez disipadas las dudas frente al uso de la herramienta se llevó a cabo un ejercicio de resolución de problemas desde las primitivas básicas, este ejercicio constó de la resolución de 3 problemas sencillos. A continuación se describe cada uno de ellos:

Primer ejercicio: Desarrolle un algoritmo en pseudocódigo capaz de leer un número entero positivo y denotar su inverso en la salida por consola.

Segundo ejercicio: Desarrolle un algoritmo en pseudocódigo capaz de leer un número entero positivo y posteriormente denote en la salida el producto de dicho número con la serie consecutiva de números del 1 al 10 por consola.

Tercer ejercicio: Desarrolle un algoritmo en pseudocódigo capaz de leer dos números (x,y) y posteriormente indique si su posición en un plano cartesiano corresponde al 1, 2, 3 o 4 cuadrante.

Entre los resultados de estas pruebas se obtuvo una completitud entre los participantes a la hora de usar la herramienta para la resolución del problema, siendo cada participante capaz de diseñar una respuesta adecuada para el problema descrito.

Además de este ejercicio, se le otorgó al estudiante el prototipo para su uso libre a lo largo de una semana. Posteriormente a esta se le envió a cada participante una encuesta de aceptación para recolectar datos sobre la perspectiva del estudiante. En dicha encuesta se incluyeron las siguientes preguntas:

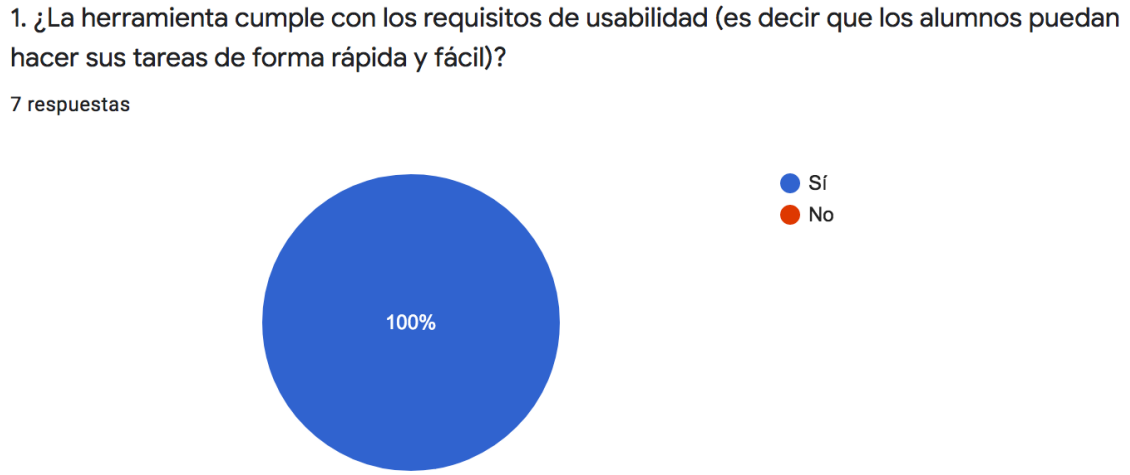
1. ¿La herramienta cumple con los requisitos de usabilidad (es decir que los alumnos puedan hacer sus tareas de forma rápida y fácil)?. Justifique su respuesta valorando aspectos incluidos en el prototipo.
2. ¿La herramienta es amigable con los alumnos para la resolución de problemas? Justifique su respuesta valorando aspectos incluidos en el prototipo que apoyen la resolución de problemas.
3. ¿La herramienta realiza comprobación de errores? Justifique su respuesta valorando aspectos incluidos en el prototipo.
4. ¿La herramienta explica paso a paso cada instrucción ejecutada? Justifique su respuesta valorando aspectos incluidos en el prototipo.
5. La herramienta es fácil de instalar en distintas plataformas.
6. Para las siguientes preguntas, evalúe el desempeño del aplicativo en una escala del 1-5. Siendo 1 muy bajo y 5 muy alto.

a. De manera general, ¿Cómo calificaría el aplicativo?
b. Desde el punto de vista académico, ¿Qué calificación daría al aplicativo como instrumento para fortalecer los conceptos de primitivas de programación?
c. Sobre la interfaz del prototipo, ¿Qué calificación le daría respecto a la facilidad de su manejo, su claridad, etc.?

7. ¿Considera la finalidad de la herramienta como un recurso de apoyo didáctico en la enseñanza-aprendizaje de algoritmos estructurados? Justifique su respuesta valorando aspectos incluidos en el prototipo.
8. Si tiene algún comentario sobre el aplicativo que pueda mejorarlo o corregirlo por favor escríbalo a continuación

Entre los resultados obtenidos se obtuvo un resultado general positivo frente al aplicativo. Algunas de las respuestas frente a la usabilidad fueron las siguientes "*En mi opinión si los cumple, ya que permite a los estudiantes implementar los algoritmos que se le planteen (que el programa permite hacer) de una manera entendible y rápida*" también comentarios como "*Cuenta con todos los aspectos que se usan en un pseudocódigo, todas las unciones marchan correctamente, es sencillo de utilizar*" y "*(...) la aplicación es muy intuitiva y fácil de entender a la hora de construir y visualizar la ejecución de los algoritmos*". Además de ello la completitud de los estudiantes encuestados se encontraron de acuerdo respecto a este aspecto como se puede evidenciar en la *Figura 21*.Con lo cual se puede afirmar que su usabilidad es lo suficientemente intuitiva para los estudiantes que se encuentran en la introducción a la programación.

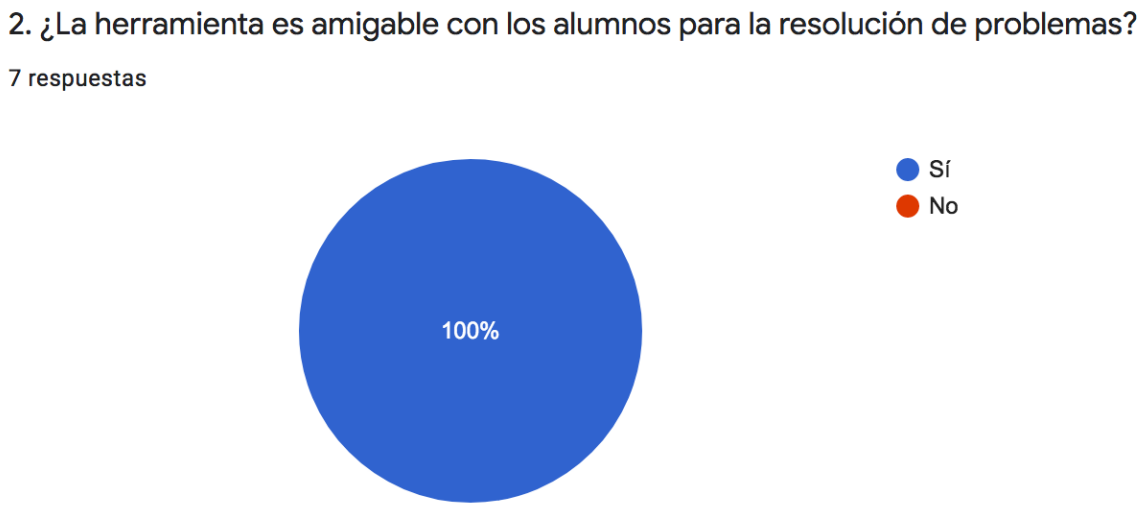
Figura 21. Resultado general respecto a la pregunta de usabilidad en la encuesta.



Fuente: *Autor propia*

Respecto a la pregunta derivada de la afabilidad de la herramienta en la resolución de problemas se identificó de manera general una aprobación de está en dicho aspecto como se puede evidenciar en la *Figura 22*. Asimismo, se encontraron comentarios como "*No requiere de conocimientos sobre un lenguaje específico, es difícil cometer errores a la hora de escribir el código debido a las restricciones del programa*" y "*Si lo es, mientras el estudiante tenga una previa introducción de la función de cada uno de los botones y en el momento en que se empieza a usar se hace más entendible cada uno de sus aspectos*".

Figura 22. Resultado general respecto a la pregunta de afabilidad en la resolución de problemas en la encuesta.

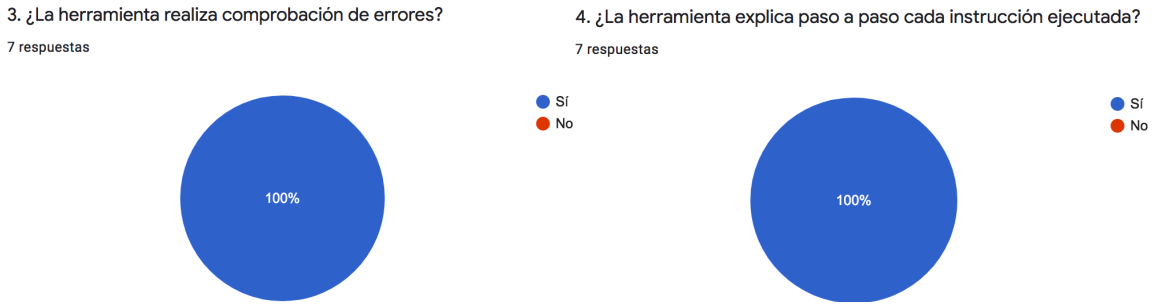


Fuente: *Autor propia*

Por otro lado, se evidenció un adecuado uso de la mecánica detectora de errores y de la ejecución paso a paso donde se apreció una general aprobación por éstas como se ve en la *Figura 23.* Donde cada uno de los participantes las identificaron como recursos de gran ayuda y validación como se puede observar en los comentarios a continuación: "*(...) al momento de escribir el código, la misma muestra si está bien escrito o no, y en dado caso de que no, esta misma suministra la información necesaria para poder identificar rápidamente el problema y darle solución*", "*(...)* sobre todo en los errores que las personas principiantes suelen cometer como (uso incorrecto de operadores, falta de ',', usar variables sin declarar,

etc)", "*Si lo hace y permite ver cada uno de los valores que se trabajan en cada instrucción*" y "*Es muy intuitiva la herramienta porque puedo ver el estado de mis variables a medida que el algoritmo se ejecuta*".

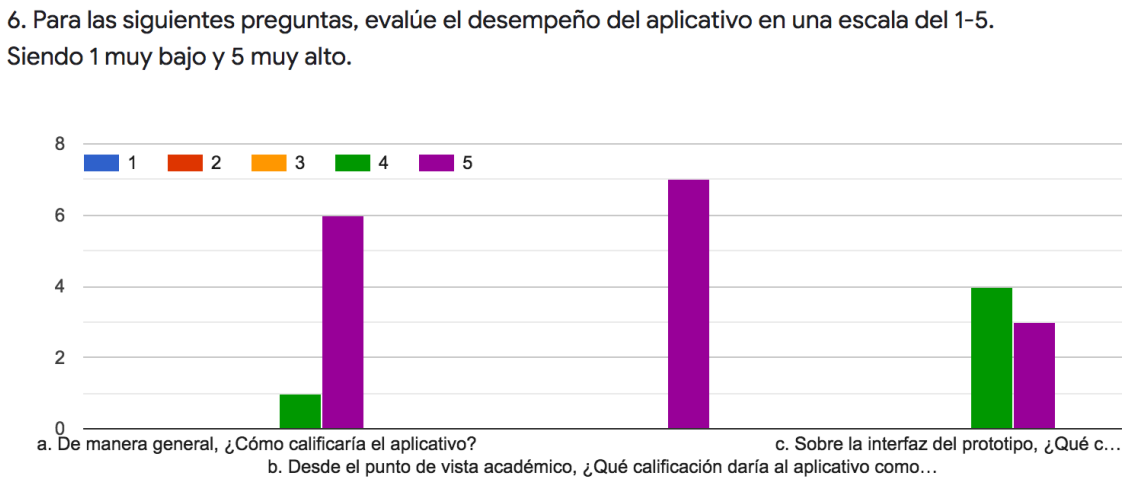
Figura 23. Resultado general respecto a las preguntas de detección de errores y ejecución paso a paso en la encuesta.



Fuente: *Autor propia*

Además de ello, los participantes identificaron en completitud la facilidad a la hora de instalar la herramienta. En la calificación del aplicativo se obtuvo una puntuación más dispareja donde algunos participantes hicieron observaciones respecto a la herramienta como se puede ver en la *Figura 24.* En cuanto a la calificación general de la herramienta se obtuvo únicamente una observación: "*Aunque cuenta con varias funciones de gran utilidad todavía faltan por añadir otros elementos de un código, estructuras de datos por ejemplo*". Aunque esta observación fue valorada y apreciada, no fue tenida en cuenta pues esta solicitud se desborda del alcance definido previamente y sobrepasa lo acordado en el proyecto de grado. Por otro lado, la calificación del aplicativo desde el punto de vista académico fue plenamente positiva con aspectos como "*Sirve de maravilla para entender el funcionamiento de un código sencillo, es ideal para programadores novatos*" y "*Bajo un punto de vista académico creo que es increíblemente útil para los primeros semestres y creo que le sería de utilidad a muchos estudiantes*". Finalmente, en la pregunta relacionada a la interfaz del aplicativo se obtuvieron 3 calificaciones excelentes y 4 buenas, estas ultimas apuntaban a observaciones estéticas del prototipo con comentarios como "*Creo que la interfaz es bueno y bastante entendible, pero creo que si se empieza a utilizar de manera a mayor escala hay algunos aspectos de la interfaz que en mi opinión se pueden mejoran un poco estéticamente*" y "*Le doy un 4, ya que considero que el programa en cuanto a interfaz puede mejorar un poco más*".

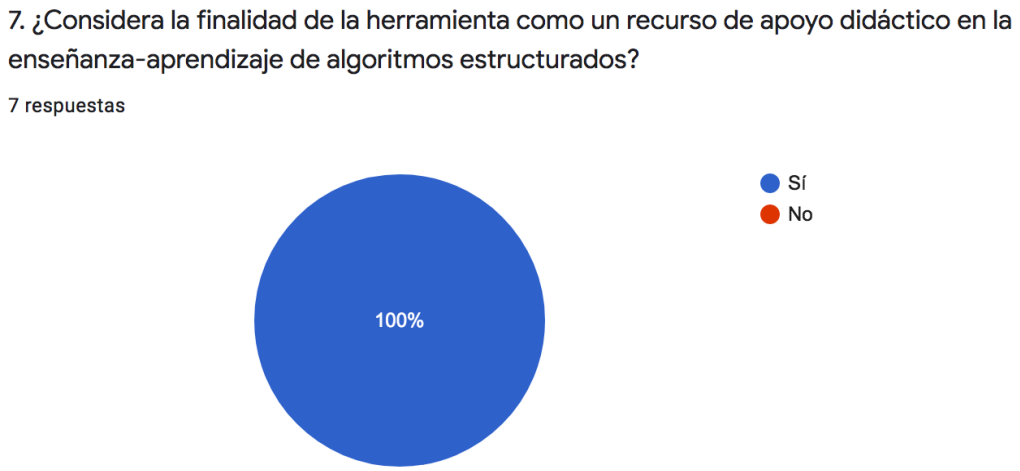
Figura 24. Resultado general respecto a las preguntas calificativas del prototipo en la encuesta.



Fuente: *Autor propia*

Una de las preguntas pilares en la encuesta fue la penúltima la cual pretendía validar la aceptación por parte de los participantes en cuanto al cumplimiento del objetivo general. La pregunta ¿Considera la finalidad de la herramienta como un recurso de apoyo didáctico en la enseñanza-aprendizaje de algoritmos estructurados? tuvo una aceptación del 100% de los estudiantes involucrados en el proceso como se puede apreciar en la *Figura 25.* Incluso se incluyeron comentarios como "*Si, unifica conceptos sintácticos y de orden que todo programador debe aplicar*", "(...) *es una excelente herramienta para ayudar a mejorar el análisis y entendimiento de los algoritmos de secuencia sobre todo para las personas que apenas están iniciando*", "*Si, ya que este aplicativo se puede emplear para la enseñanza de personas que quieran empezar en el mundo de la programación y mejorar la lógica*" y "*Considero que es una herramienta muy útil que me hubiese gustado utilizar cuando no sabía nada de programación, ya que te permite aprender de manera autónoma y la retroalimentación de lo que propones como solución te ayuda a aprender más rápido*".

Figura 25. Resultado general respecto a la pregunta de aceptación en función al objetivo planteado para el desarrollo de la herramienta en la encuesta.



Fuente: *Autor propia*

Finalmente, en la última pregunta haciendo alusión a mejoras y correcciones, ninguno de los participantes identificó correcciones en el sistema, es decir no se encontraron errores. Mas sin embargo se sugieren mejoras en la integración de nuevos conceptos los cuales no fueron planteados en el alcance como lo expone el siguiente comentario "*En ocasiones puede llegar a sentirse lento y limitante debido a la forma en la que se escribe el código, sin embargo esta función ayuda a que se disminuyen errores. También se recomienda la inclusión de otros elementos como Arrays, ya que estos son pueden ser los más difíciles de entender por novatos*". Por otro lado, en un enfoque en cuanto a la interfaz se resume en la mejora de esta como se aprecia en el siguiente comentario "*Creo que si se empieza a utilizar de manera a mayor escala hay algunos aspectos de la interfaz que en mi opinión se pueden mejorar un poco estéticamente*".

15. CONCLUSIONES

Como se evidenció en el estado actual, la dificultad en la educación introductoria a ciencias de la computación, más específicamente a la programación, es un problema común. Problema el cual recae en la incomprensión de conceptos abstractos los cuales requieren de una experiencia práctica para entender sus comportamientos en un entorno dinámico. Diversas soluciones se han llevado a cabo dentro del aula con el fin de mitigar la confusión en los estudiantes, entre las propuestas más comunes encontramos el software educativo como una herramienta efectiva a la hora de brindar una experiencia más concreta de los conceptos incorporados. Así mismo, existen diferentes presentaciones con diferentes enfoques como la presentación de conceptos, la motivación del estudiante, el testeo o puesta en práctica de las soluciones diseñadas por el estudiante, entre otros. Siendo cada una de estas soluciones enfocadas a diferentes públicos o usuarios según el entorno donde se desarrolla. Es por ello que es evidente que esta propuesta es indicada a la hora de apoyarlo desde el enfoque práctico-experimental como medio de retroalimentación según el temario delimitado y definido desde los conceptos incorporados en el alcance.

Posteriormente se elaboró un análisis respecto a las dificultades y variables involucradas en los cursos introductorios a la programación y con ello se realizó un levantamiento de requerimientos para un entorno asistido que pretende evadir errores sintácticos, léxicos y semánticos en el estudiante y de esta manera realizar un enfoque en los errores lógicos. Además de establecer requerimientos funcionales, se establecieron requerimientos no funcionales los cuales constituirán la presentación de la solución y su modo de interactuar con el estudiante. Detallando una justificación para cada requerimiento, incluyendo un análisis de la relación que tiene con la problemática. Siendo este aspecto importante como un fundamento para el desarrollo del prototipo.

En el desarrollo del prototipo se implementó la metodología SCRUM, donde los eventos y principales características de esta metodología fueron detallados en el periodo temporal del proyecto, donde se constataron los roles, las reuniones y las historias de usuario basadas en los requerimientos planteados anteriormente. Para el desarrollo del proyecto, a un nivel más técnico, se llevaron a cabo los diseños y modelados de vistas y diagramas. Los cuales fueron un recurso importante y dinámico en el desarrollo del proyecto, estas vistas definieron los componentes, sus

responsabilidades y sus relaciones o interacciones con los demás. Estableciendo la interacción de las partes con un todo en función de los requerimientos establecidos.

Hablando referente al prototipo construido, el sistema se comprende como un intérprete simulado de pseudocódigo, esto a razón de la esencia principal de la herramienta a la hora de simular la ejecución del pseudocódigo creado. Donde se analiza la veracidad del pseudocódigo de carácter léxico, sintáctico y semántico, generando un listado de instrucciones por medio de nodos secuenciadores los cuales simulan un lenguaje ensamblador (de manera funcional) y posteriormente se interpreta esta analogía de lenguaje ensamblador. En dicha interpretación se denotan espacios en memoria fuertemente tipados en la tabla de símbolos, y se registra cada cambio en memoria. Propiamente esta herramienta en un análisis de ingeniería inversa puede resultar en un modelo dinámico para la educar en conceptos básicos de intérpretes y cómo estos funcionan en composición. Además de cumplir su propósito al desarrollar los requerimientos establecidos, los cuales se establecieron con el fin de apoyar el proceso de aprendizaje en el entorno y los conceptos delimitados en el alcance.

El proceso de pruebas descrito en el capítulo anterior permite evidenciar la presencia de errores y consigo la corrección de estos. Este proceso de depuración fue importante a la hora de mejorar la calidad del prototipo según los requerimientos establecidos. Así mismo, bajo el set de pruebas declaradas se logró el objetivo de validar su funcionalidad de manera tal que fuera posible presentarse a un grupo focal de participantes que puedan utilizar el aplicativo. Estos participantes asimismo se enfrentaron a los problemas descritos anteriormente donde la solución de los ejercicios no tuvo mayor problema entre los participantes pues identificaron las primitivas necesarias para la resolución de los problemas, además de ello pudieron incorporar de manera íntegra y en conjunto cada uno de los conceptos declarados en el alcance como medio suficiente para la solución del problema. Donde en conclusión se puede evidenciar que la herramienta soporta aspectos capaces de apoyar al estudiante en la resolución de problemas por medio de los conceptos ya mencionados, así mismo se evidencio que los estudiante hacían constante uso del modo ejecución paso a paso para disipar dudas mayormente en la evaluación de expresiones. Con el prototipo, en función al requerimiento de generar y remover estructuras de pseudocódigo se evidenció el hecho de que los estudiantes tomaban mucho más tiempo a la hora insertar una primitiva, debido a que el estudiante analiza y diseña con mayor cuidado el algoritmo solución del problema. La razón principal se debe a la mecánica de remoción, ya que esta borra el bloque incluyendo los contenidos con lo cual le puede interponer al estudiante un mayor trabajo a la

hora de querer reinsertar las primitivas que se encontraban dentro del bloque. Asimismo, dentro del proceso fue evidente que la integración del software educativo en conjunto al acompañamiento pedagógico ha permitido identificar y orientar el papel del docente respecto al proceso educativo, y mejor aun permitiéndole al estudiante construir conocimiento de manera cooperativa y autónoma. Finalmente se realizó una encuesta a cada uno de los participantes del grupo focal, la cual tuvo una aceptación por parte de cada uno de los participantes afirmando que se cumplia con el objetivo general del proyecto ya que la herramienta los apoyó en el ejercicio llevado a cabo y así mismo identificaron aspectos de vital importancia en la introducción a la programación, validando la hipótesis anteriormente planteada. Sin embargo, dicha encuesta también incluye comentarios de mejora como la interfaz desde un factor estético.

Un aporte de esta propuesta al entorno medio ambiental es la sustitución de los medios físicos y estáticos que se mencionaron al inicio de este documento, por un entorno digital y reutilizable en el espacio de clase donde se estaba llevando el proceso pedagógico, posibilitando la reducción de estos de estos recursos mencionados. Esto es de gran ayuda a la digitalización del conocimiento y a la transformación de la educación (presencial a remota) que se ha venido dando en los últimos años, la cual se aceleró a razón de las medidas de bioseguridad designadas debido a la pandemia. Además de ello, esta transformación abre la oportunidad y el espacio a una mayor accesibilidad a la educación. Este último factor es claro e importante ya que se encuentra alineado con los objetivos de desarrollo sostenible de la ONU. Donde la situación actual, dado el cierre temporal de las escuelas, la ONU[109] reporta una afectación del 91% de los estudiantes en el mundo alterando su aprendizaje y asimismo sus vidas. Es por ello que esta herramienta busca mitigar, en cierta parte, este cambio al abrir un nuevo espacio de desarrollo.

Finalmente, la presente investigación puede concluirse con la aceptación del cumplimiento de cada uno de los propósitos descritos en capítulos anteriores y validando la hipótesis planteada. Así como el cumplimiento del objetivo general desde la perspectiva de cada uno de los participantes inmersos en el grupo focal. Es decir, se logró identificar las variables involucradas en el problema, con base a ello se realizó un levantamiento de requerimientos (funcionales y no funcionales) fundamentado en las necesidades identificadas, luego (aplicando la metodología

[109] ORGANIZACIÓN DE LAS NACIONES UNIDAS. Objetivo 4: Garantizar una educación inclusiva, equitativa y de calidad y promover oportunidades de aprendizaje durante toda la vida para todos [en línea]. ONU; [Consultado: 11 de mayo de 2021]. Disponible en: https://www.un.org/sustainabledevelopment/es/education/

SCRUM) se diseñaron vistas y modelos, continuamente a ello se construyó el aplicativo. Toda la documentación se encuentra expuesta de manera digital junto con un manual de usuario, manual de instalación y descarga del aplicativo en la página (*https://brandonrodriguezc.github.io/testdesk-documentacion/*). Por último, se aplicó la etapa de pruebas planificadas en el cronograma, dichas pruebas fueron funcionales y con el grupo focal. Concluyendo de manera satisfactoria la resolución a la pregunta problema validando la hipótesis y consigo la investigación.

16. RECOMENDACIONES

Si bien el proyecto cumple bajo los requerimientos funcionales y no funcionales, dado el alcance proyectado al inicio de la investigación. Este proyecto se puede transformar de diferentes maneras, como una versión móvil o un nuevo entorno gráfico. Así mismo como la inclusión de nuevos conceptos que no se incluyeron en el alcance como la adición de librerías externas, la inclusión de métodos secundarios (llamado y creación), la inclusión de la recursión, entre otras.

17. BIBLIOGRAFÍA

ALFONSECA MORENO, Manuel et al. Análisis morfológico. En: Compiladores e intérpretes: teoría y práctica [en línea]. PEARSON PRENTICE HALL, 2006. [Consultado el 20 de abril de 2021]. Capítulo 3. 65 p. Disponible en https://d1wqtxts1xzle7.cloudfront.net/51585857/Compiladores-e-interpretes-teoria-y-practica.pdf?1485977162=&response-content-disposition=inline%3B+filename%3DCompiladores_e_interpretes_teoria_y_prac.pdf&Expires=1619016405&Signature=X0MqgGKuxe65w3xbiczHClbBiH9051cOn9xDh8qjZzzHTU6mfJkcbn4dPo1hrCrbWkgCq~w799geheJmWB3JneM3hUnNwNWI0LrzDZgvgZiX1CkHl-j4Mx2DSybOW~NHwlFGpDIXGAUAVpyQp14zvcrQp12XnXVuVQXqWg9xsnpv8~hWVeEBqW-h5W3TB-1yFFW~4-9MM3TqJuOxctDLDj9BnyjNPJTWVrXVnBuOsdwSbtMKSsOzUNWClvvLsTfk2OvLsdlZTwqFdr2~Oi0Oi0~lUBUO04J90dNOfuhJvZ7-QbPHqhjCxcmcwv8xxQwvAZGBwK2MgXxYGnTMv~iMY5a-Q__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA

ALFONSECA MORENO, Manuel et al. Intérpretes. En: Compiladores e intérpretes: teoría y práctica [en línea]. PEARSON PRENTICE HALL, 2006. [Consultado el 20 de abril de 2021]. Capítulo 8. 317 p. Disponible en https://d1wqtxts1xzle7.cloudfront.net/51585857/Compiladores-e-interpretes-teoria-y-practica.pdf?1485977162=&response-content-disposition=inline%3B+filename%3DCompiladores_e_interpretes_teoria_y_prac.pdf&Expires=1619016405&Signature=X0MqgGKuxe65w3xbiczHClbBiH9051cOn9xDh8qjZzzHTU6mfJkcbn4dPo1hrCrbWkgCq~w799geheJmWB3JneM3hUnNwNWI0LrzDZgvgZiX1CkHl-j4Mx2DSybOW~NHwlFGpDIXGAUAVpyQp14zvcrQp12XnXVuVQXqWg9xsnpv8~hWVeEBqW-h5W3TB-1yFFW~4-9MM3TqJuOxctDLDj9BnyjNPJTWVrXVnBuOsdwSbtMKSsOzUNWClvvLsTfk2OvLsdlZTwqFdr2~Oi0~lUBUO04J90dNOfuhJvZ7-QbPHqhjCxcmcwv8xxQwvAZGBwK2MgXxYGnTMv~iMY5a-Q__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA

ALFONSECA MORENO, Manuel et al. Tabla de símbolos. En: Compiladores e intérpretes: teoría y práctica [en línea]. PEARSON PRENTICE HALL, 2006. [Consultado el 20 de abril de 2021]. Capítulo 2. 33 p. Disponible en https://d1wqtxts1xzle7.cloudfront.net/51585857/Compiladores-e-interpretes-teoria-

y-practica.pdf?1485977162=&response-content-disposition=inline%3B+filename%3DCompiladores_e_interpretes_teoria_y_prac.pdf&Expires=1619016405&Signature=X0MqgGKuxe65w3xbiczHClbBiH9051cOn9xDh8qjZzzHTU6mfJkcbn4dPo1hrCrbWkgCq~w799geheJmWB3JneM3hUnNwNWI0LrzDZgvgZiX1CkHl-j4Mx2DSybOW~NHwlFGpDIXGAUAVpyQp14zvcrQp12XnXVuVQXqWg9xsnpv8~hWVeEBqW-h5W3TB-1yFFW~4-9MM3TqJuOxctDLDj9BnyjNPJTWVrXVnBuOsdwSbtMKSsOzUNWClvvLsTfk2OvLsdlZTwqFdr2~Oi0~lUBUO04J90dNOfuhJvZ7-QbPHqhjCxcmcwv8xxQwvAZGBwK2MgXxYGnTMv~iMY5a-Q__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA

ANDERSON, Juan y GONZALES-BAÑALES, Dora. PSeInt como herramienta para mejorar el proceso de enseñanza aprendizaje de algoritmos, pseudocódigo y diagramas de flujo. En: Tecnologías de la Información en Educación: Sistematización de experiencias docentes [en línea]. 1 ed. Durango: Red Durango de Investigadores Educativos, 2020. [Consultado el 1 de noviembre de 2020]. Capítulo 8. 91p. Disponible en https://www.researchgate.net/profile/Dora_Gonzalez-Banales/publication/342243248_Tecnologias_de_informacion_en_educacion_Sistematizacion_de_experiencias_docentes/links/5eea3e96a6fdcc73be84b5aa/Tecnologias-de-informacion-en-educacion-Sistematizacion-de-experiencias-docentes.pdf#page=99

ARELLANO, Jesús et al. Software para la enseñanza-aprendizaje de algoritmos estructurados. En: Revista Iberoamericana de Educación en Tecnología y Tecnología en Educación N°8 [en línea]. La Plata: Universidad Nacional de La Plata, diciembre de 2012. vol. nro. 8. p. 23-33. [Consultado: 29 de septiembre de 2020]. Disponible en http://sedici.unlp.edu.ar/bitstream/handle/10915/25531/Documento_completo.pdf?sequence=1&isAllowed=y. ISSN 1850-9959.

ASTRACHAN, Owen. Concrete Teaching: Hooks and Props as Instructional Technology. En: SIGCSE Bull [en línea]. Nueva York: Association for Computing Machinery, septiembre de 1998. vol. 30 nro. 3. p. 21-24. [Consultado el 1 de noviembre de 2020]. Disponible en: https://users.cs.duke.edu/~ola/papers/propsfinal.pdf. ISSN: 0097-8418.

BENNEDSEN, Jens y CASPERSEN, Michael E. Failure Rates in Introductory Programming. En: SIGCSE Bull [en línea]. Nueva York: Association for Computing Machinery, junio de 2007. vol.39, nro. 2. p. 32–36. [Consultado: 29 de septiembre de 2020]. Disponible en https://www.researchgate.net/publication/220613179. ISSN: 0097-8418.

BLAKE, Jonathan D. The Journal of Computing Sciences in Colleges [en línea]. En: Papers of the Sixteenth Annual CCSCNortheastern Conference. (15-16, noviembre. Massachusetts, Estados Unidos). Language considerations in the first year CS curriculum. Springfield: Western New England College. 2011. p. 124-129. [Consultado: 25 de septiembre de 2020]. Disponible en: https://www.researchgate.net/publication/234781683

BUCK, Duane y STUCKI, David. JKarelRobot: A Case Study in Supporting Levels of Cognitive Development in the Computer Science Curriculum. En: SIGCSE Bull [en línea]. Nueva York: Association for Computing Machinery, marzo de 2001. vol. 33, nro. 1. p. 16–20. [Consultado: 1 de noviembre de 2020]. Disponible en https://www.researchgate.net/publication/221536705. ISSN: 0097-8418.

CAMANA, Roberto y SALGUERO, Anita. Congreso Internacional “Ciencia, Sociedad e Investigación Universitaria” [en línea]. En: II Congreso Internacional “Ciencia, Sociedad e Investigación Universitaria”. (18-20, octubre. Ecuador). Validación y evaluación de VisualWebMedia para la enseñanza de conceptos básicos deprogramación estructurada. Ecuador. 2017. p. 1-16. [Consultado el 8 de marzo de 2021]. Disponible en: https://repositorio.pucesa.edu.ec/handle/123456789/2321

CARRIZO, Rosa; CORSO, Lorena y OLMEDO, Adriana. Congreso Argentino de Ingeniería Industrial [en línea]. En : VIIIº Congreso Argentino de Ingeniería Industrial. (12-13, noviembre. Córdoba, Argentina). Aplicación de una herramienta basada en Software Libre para la enseñanza de Algoritmos y Lógica de Programación. Córdoba. 2015. p. 1-9. [Consultado: 1 de noviembre de 2020]. Disponible en: http://www.edutecne.utn.edu.ar/coini_2015/trabajos/F030_COINI2015.pdf

CLIFTON, Mitchell. A technique for making structured programs more readable. En: SIGPLAN Not [en línea]. Nueva York: Association for Computing Machinery, abril de 1978. vol.13 nro. 4. p. 58–63.[Consultado el 1 de noviembre de 2020]. Disponible en: https://dl.acm.org/doi/abs/10.1145/953411.953415. ISSN: 0362-1340.

Code.org. Lo que la mayoría de las escuelas no enseñan [video]. Estados Unidos: YouTube, Code.org. (23 de febrero de 2013). 5:43 minutos. [Consultado: 19 de abril de 2021]. Disponible en: https://www.youtube.com/watch?v=nKIu9yen5nc

CONNOLLY, Thomas. Using Games-Based eLearning Technologies in Overcoming Difficulties in Teaching Information Systems. En: Journal of Information Technology Education [en línea]. Estados Unidos: Informing Science Institute, 2006. vol. 5, nro. 1. p. 459-476. [Consultado el 12 de abril de 2021]. Disponible en https://www.researchgate.net/publication/220590643_Using_Games-Based_eLearning_Technologiesin_Overcoming_Difficulties_in_Teaching_Information_Systems

COOPER, Gina. World Congress in Computer Science, Computer Engineering, & Applied Computing [en línea]. En: Proceedings of The 2014 World Congress in Computer Science, Computer Engineering, and Applied Computing. (21-24, julio. San Diego, Estados Unidos). Using Visual Logic with Pseudocode to Teach an Introductory Programming Course. San Diego. 2014. p. 58-63. [Consultado: 1 de noviembre de 2020]. Disponible en: http://worldcomp-proceedings.com/proc/p2014/FEC2198.pdf

DAROS, William. ¿Qué es un marco teórico?. En: Enfoques [en línea]. Bolivia: Universidad Adventista del Plata, 2001. vol. 14, nro. 1-2. p. 73-112. [Consultado el 12 de abril de 2021]. Disponible en http://www.publicaciones.uap.edu.ar/index.php/revistaenfoques/article/view/348/338. ISSN: 1669-2721

DEITEL, Paul y DEITEL, Harvey. Cómo programar en Java. 7 ed. Naucalpan de Juárez: PEARSON PRENTICE HALL, 2008. XXVIII p. [Consultado el 20 de abril de 2021]. Disponible en: http://www.mfbarcell.es/docencia_uned/fund_inf_ing/libros/-%20Como%20Programar%20en%20Java%20Deitel.pdf. ISBN 978-970-26-1190-5

DEL PRADO, Ana y LAMAS, Nancy. Alternativas para la enseñanza de pseudocódigo y diagrama de flujo. En: Revista Electrónica Iberoamericana de Educación en Ciencias y Tecnología [en línea]. Catamarca: Universidad Nacional de Catamarca, diciembre de 2014. vol. 5, nro. 3. p. 102-113. [Consultado: 1 de noviembre de 2020]. Disponible en http://www.exactas.unca.edu.ar/riecyt/VOL%205%20NUM%203/F%20%20SI%203%2014%20Trabajo%20Completo%20Fundamentos.pdf, ISSN: 1852-852X

DERUS, Siti y ALI, Mohamad. International Conference on Current Issues in Education [en línea]. En: 1st International Conference oo Current Issues in Education. (19-21, noviembre. Madrid, España). Difficulties in learning programming: Views of students. Madrid. 2012. p. 1-6. [Consultado el 12 de abril de 2021]. Disponible en: https://www.researchgate.net/publication/267338258_Difficulties_in_learning_programming_Views_of_students

EDWARDS, Stephen. Special Interest Group on Computer Science Education [en línea]. En: Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education. (3-7, marzo. Virginia, Estados Unidos). Using software testing to move students from trial-and-error to reflection-in-action. Virginia. 2004. p. 1-5. [Consultado el 12 de abril de 2021]. Disponible en: https://www.researchgate.net/publication/221537536_Using_software_testing_to_move_students_from_trial-and-error_to_reflection-in-action

FUENTES, Jorge y MOO, Melquizedec. Dificultades de aprender a programar. En: Revista Educación en Ingeniería [en línea]. Colombia: Asociación Colombiana de Facultades de Ingeniería, julio de 2017. vol. 12, nro. 24. p. 76-82. [Consultado: 29 de septiembre de 2020]. Disponible en https://doi.org/10.26507/rei.v12n24.728

GARCÍA, Jesus. Jornadas de Enseñanza universitaria de la Informática [en línea]. En: Actas de las VII Jornadas de Enseñanza universitaria de la Informática. (16-18, julio. Palma de Mallorca, España). ¿Es conveniente la orientación a objetos en un primer curso de programación?. En: Universidad de les Illes Balear. 2001. p. 291-298. [Consultado el 12 de abril de 2021]. Disponible en: http://bioinfo.uib.es/~joemiro/aenui/procJenui/ProcWeb/actas2001/gaesc43.pdf]

GIANNAKOULAS, A. y XINOGALOS, Stelios. A pilot study on the effectiveness and acceptance of an educational game for teaching programming concepts to primary school students. En: Education and Information Technologies [en línea]. Macedonia: Universidad de Macedonia, 2018. vol. 23, nro. 5. p. 2029-2052. [Consultado el 8 de marzo de 2021]. Disponible en: https://ruomo.lib.uom.gr/bitstream/7000/342/1/EAIT-RunMarco-postprint.pdf

GIMPEL, James. Contour: A Method of Preparing Structured Flowcharts. En: SIGPLAN Not. [en línea]. Nueva York, octubre de 1980. vol. 15, nro. 10. p. 35–41. [Consultado el 1 de noviembre de 2020]. Disponible en https://dl.acm.org/doi/10.1145/947727.947730. ISSN: 0362-1340

GOMES, Anabela y MENDES, Antonio. Conference: Computers and Education [en línea]. En: Towards an Interconnected Society. (21, enero). SICAS: Interactive system for algorithm development and simulation. 2015. p. 1-9. [Consultado: 1 de noviembre de 2020]. Disponible en: https://www.researchgate.net/publication/228328489

GUTIERREZ, Demián. 01 UML Diagramas de Clases (UML ilustrado)[diapositivas]. codecompiling. Venezuela. marzo de 2011, 64 diapositivas. [Consultado 24 de abril de 2021]. Disponible en: https://www.codecompiling.net/files/slides/UML_clase_04_UML_clases.pdf

ISO-IEC-IEEE. Systems and software engineering - System life cycle processes, ISO/IEC/IEEE 15288 [en línea]. Estados Unidos: ISO/IEC JTC 1/SC 7Software and systems engineering. 2015. 108 p. [Consultado 24 de abril de 2021]. Disponible en: https://www.iso.org/standard/63711.html

KEREKI, Ines y SALVETTO, Fernando. Enseñanza Inicial de la Programación: Comparación entre Programación Estructurada y Programación Orientada a Objetos. En: Universidad ORT Uruguay [en línea]. Montevideo: Universidad ORT Uruguay. vol. , nro. p. 1-8. [Consultado el 1 de noviembre de 2020]. Disponible en https://fi.ort.edu.uy/innovaportal/file/5553/2/comparacionpe_poo.pdf

KOVALSCHICK, Ann y DAWSON, Kara. Education and Technology. 1 ed. Santa Barbara, California; Denver, Colorado; Oxford, England: ABC-CLIO, 2003. 544p. ISBN 1-57607-351-3

LEINBAUGH, Dennis. Indenting for the Compiler. En: Association for Computing Machinery [en línea]. Nueva York, mayo de 1980. vol. 15, nro. 5. p. 41–48. [Consultado el 1 de noviembre de 2020]. Disponible en: https://dl.acm.org/doi/10.1145/947639.947644. ISSN: 0362-1340

MARCELINO, María et al. 4th International Conference on Computer Systems and Technologies [en línea]. En: Educational Aspects of Computer Systems and Technologies (e-Learning). (17-18 junio. Rousse, Bulgaria). Using a computer-based interactive system for the development of basic algorithmic and programming skills. 2004. p. IV.8-(1-6). [Consultado: 1 de noviembre de 2020]. Disponible en: https://www.researchgate.net/publication/228328510_Using_a_computer-

based_interactive_system_for_the_development_of_basic_algorithmic_and_programming_skills

MATHRANI, Anuradha; CHRISTIAN, Shelly y PONDER-SUTTON, Agate. PlayIT:Game Based Learning Approach for Teaching Programming Concepts. En: Journal of Educational Technology & Society [en línea]. Taiwan: International Forum of Educational Technology & Society, julio de 2016. vol. 19, nro. 2. p. 5-17. [Consultado el 8 de marzo de 2021]. Disponible en: https://pdfs.semanticscholar.org/98ce/1c523d7617e2b64af12d87f3f1bb69e4f0ca.pdf

MCCARTHY, John. LISP Session. En: Association for Computing Machinery [en línea]. Nueva York: Wexelblat, Richard L, 1978. vol. , nro. pp. 172-185. [Consultado el 1 de noviembre de 2020]. Disponible en: https://www.labouseur.com/courses/tpl/History%20of%20LISP-mccarthy.pdf. ISBN: 0-12-745040-8

MIARA, Richard et al. PROGRAM INDENTATION AND COMPREHENSIBILITY. En: Communications of the ACM [en línea]. Estados Unidos: Universidad de Maryland, noviembre de 1983. vol. 26, nro. 11. p. 861-867. [Consultado el 1 de noviembre de 2020]. Disponible en: https://www.cs.umd.edu/~ben/papers/Miara1983Program.pdf

MILNE, Iain y ROWE, Glenn. Difficulties in Learning and Teaching Programming—Views of Students and Tutors. En: Education and Information Technologies. Estados Unidos: Education and Information Technologies, marzo 2002. vol. 7, nro. p. 55-66. [Consultado: 21 de abril de 2021]. Disponible en: https://doi.org/10.1023/A:1015362608943

MINISTERIO DE TECNOLOGÍAS DE LA INFORMACIÓN Y LAS COMUNICACIONES [sitio web]. Colombia. MINTIC. MinTIC inicia proyecto para formar 100.000 colombianos en lenguaje de programación. [Consultado: 29 de septiembre de 2020]. Disponible en: https://www.mintic.gov.co/portal/inicio/Sala-de-Prensa/Noticias/146273:MinTIC-inicia-proyecto-para-formar-100-000-colombianos-en-lenguaje-de-programacion

MORENO, Julian y MONTAÑO, Edgar. TALLER INTERNACIONAL DE SOFTWARE EDUCATIVO [en línea]. En: XIV TALLER INTERNACIONAL DE SOFTWARE EDUCATIVO, TISE ´09. (1-3, diciembre. Santiago de Chile, Chile). ProBot: Juego para el aprendizaje de lógica de programación. Santiago. 2009. p. 1-

7. [Consultado: 1 de octubre de 2020]. Disponible en: http://www.tise.cl/2009/tise_2009/pdf/1.pdf

MUÑOZ, Roberto et al. TALLER INTERNACIONAL DE SOFTWARE EDUCATIVO [en línea]. En: Memorias del XVII Congreso Internacional de Informática Educativa, Nuevas Ideas en Informátíca Educativa. (Santiago, Chile). Determinando las dificultades en el aprendizaje de la primera asignatura de programación en estudiantes de ingeniería civil informática. Santiago: 2012. p. 120-126. [Consultado: 29 de septiembre de 2020]. Disponible en: http://www.tise.cl/volumen8/TISE2012/17.pdf

NAVARRO, Andrés; FERNÁNDEZ, Juan y MORALES, Jonathan. Revisión de metodologías ágiles para el desarrollo de software. En: PROSPECTIVA [en línea]. Barranquilla: Universidad Autónoma del Caribe, junio-diciembre de 2013. vol. 11, nro. 2. p. 30-39. [Consultado el 1 de noviembre de 2020]. Disponible en https://www.redalyc.org/pdf/4962/496250736004.pdf. ISSN: 1692-8261.

OLMOS, Karla; MORALES, Cristal; ROJAS, Teresa y FERNÁNDEZ, Luis. Objetos de Aprendizaje Enfocados a la Resolución de Problemas para Facilitar la Enseñanza de la Programación. En: Universidad Autónoma de Ciudad Juárez, Instituto de Ingeniería y Tecnología [en línea]. Ciudad Juárez: Universidad Autónoma de Ciudad Juárez, 2010. vol. nro. p. 1-6. [Consultado el 12 de abril de 2021]. Disponible en https://www.academia.edu/1303813/Objetos_de_Aprendizaje_Enfocados_a_la_Resoluci%C3%B3n_de_Problemas_para_Facilitar_la_Ense%C3%B1anza_de_la_Programaci%C3%B3n

ORGANIZACIÓN DE LAS NACIONES UNIDAS. Objetivo 4: Garantizar una educación inclusiva, equitativa y de calidad y promover oportunidades de aprendizaje durante toda la vida para todos [en línea]. ONU; [Consultado: 11 de mayo de 2021]. Disponible en: https://www.un.org/sustainabledevelopment/es/education/

ORJUELA, Ailin y ROJAS, Mauricio. Las Metodologías de Desarrollo Ágil como una Oportunidad para la Ingeniería del Software Educativo. En: Revista Avances en Sistemas e Informática [en línea]. Medellín: Universidad Nacional de Colombia, junio de 2018. vol. 5, nro. 2. p. 159-171. [Consultado 1 de noviembre de 2020]. Disponible en:https://www.redalyc.org/pdf/1331/133115027022.pdf

OUZA, Draylson Micael; ISOTANI, Seiji y BARBOSA, Ellen Francine. Teaching novice programmers using ProgTest. En: International Journal of Knowledge and Learning [en línea]. Switzerland: Inderscience Enterprises Limited, agosto de 2015. vol. 10, nro. 1. p. 1-18. [Consultado el 8 de marzo de 2021]. Disponible en https://www.researchgate.net/profile/Seiji-Isotani/publication/281578819

PINEDA, Evelyn y GONZALEZ, Carlos. Networking Skills in Latin America. En: IDC Analyze the Future [en línea]. mayo de 2016 pp. 1-32. [Consultado: 29 de septiembre de 2020] . Disponible en: https://www.cisco.com/c/dam/assets/csr/pdf/IDC_Skills_Gap_-_LatAm.pdf

POLYA, George. Cómo plantear y resolver problemas. 2 ed. México: Trillas, 1989. 28 p. [Consultado el 20 de abril de 2021]. Disponible en: https://d1wqtxts1xzle7.cloudfront.net/34996114/Polya_-_Como_plantear_y_resolver_problemas.pdf?1412448720=&response-content-disposition=inline%3B+filename%3DPolya_Como_plantear_y_resolver_problemas.pdf&Expires=1619015790&Signature=cjnudU94TRKoD9YsLcpYA9UZYY~Y61rDOKYoURCP6KBWPtmM1TT2lj5wKP8iqn9E6PneiEUeaEv60hhNzzHmjIcZ0UfTwz33adaV7ZDsk9PQmXg-dqcelX2D4M5hkKknM526nqn9ceFdyrgSOyRkhtatFOg-vcjvuTrWzbTD775zafT-xD14EpVwBQ~nHovVYqyKZJoCYQk~gktitCrrupKij5Ulrt0iB5B3ENTiDiCBqg9a2miA0jee5L5CDW3wcaE0Fag8ubpQP2-aHg4PtB4wpys-AZkwbEV0XZqwupj5K8kRKPyBvWybG5~sFtQx0nRnzawPViUVTI3WM7guCQ__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA. ISBN 968-24-0064-3

PORTO, Roberto et al. MARCO DE ENSEÑANZA BASADO EN OVAs PARA EL ADIESTRAMIENTO EN LA INTRODUCCIÓN DE LOS ALGORITMOS (OLIA). En: Revistas de la Universidad de Granada etic@net [en línea]. Granada: Universidad de Granada, 2017. vol. 17, nro. 1. p. 60-75. [Consultado el 8 de marzo de 2021]. Disponible en http://eticanet.org/revista/index.php/eticanet/article/view/126

REDONDO, Miguel; MENDES, Antonio; MARCELINO, María y BRAVO Crescencio. TALLER INTERNACIONAL DE SOFTWARE EDUCATIVO [en línea]. En: 8vo Taller Internacional de Software Educativo. (24-26, noviembre. Santiago, Chile). Planificación colaborativa del diseño para el aprendizaje de la programación. Santiago de Chile. 2003. p. 1-11. [Consultado el 1 de noviembre de 2020]. Disponible en: http://www.tise.cl/2010/archivos/tise2003/papers/planificacion_colaborativa.pdf

SALCEDO, Walter; MÜLLER, Christian y OTERO, Manuel. III Jornadas Argentinas sobre Tecnología, Innovación y Creatividad [en línea]. En: III JATIC 2017. (1-3, noviembre. Buenos Aires, Argentina). UAIDesign: Herramienta Colaborativa Multiplataforma para la enseñanza de Diseño de Algoritmos y Programación Estructurada. Buenos Aires. 2017. p. 1-8. [Consultado el 8 de marzo de 2021]. Disponible en: http://bon-cloud.net/info/pubs/07-jatic-2017.pdf

SAMPIERI, Roberto; FERNÁNDEZ, Carlos y BAPTISTA, María del Pilar. Definiciones de los enfoques cuantitativo y cualitativo, sus similitudes y diferencias. En: Metodología de la Investigación [en línea]. McGraw-Hill Interamericana, 2014. [Consultado el 20 de abril de 2021]. Capítulo 1. 8 p. Disponible en https://www.uca.ac.cr/wp-content/uploads/2017/10/Investigacion.pdf

SANTOS Raúl; CHUC Fernando; CADENA Selene y SILVA Hilda. EL MÉTODO HEURÍSTICO DE POLYA EN UN ESCENARIO DE INVESTIGACIÓN: APLICACIÓN EN UN CASO ESPECÍFICO. En: I. C. Investig@ción [en línea]. Campeche: Instituto Campechano, junio-noviembre de 2018. vol. nro. 14. p. 9-21. [Consultado el 1 de noviembre de 2020]. Disponible en http://instcamp.edu.mx/wp-content/uploads/2018/11/Ano2018No14_9_21.pdf

SATORRE, Rosana; LLORENS, Faraón y PUCHOL, Juan. Enseñar Programación en las Ingenierías Informáticas. En: II Jornadas Nacionales de Innovación en las Enseñanzas de las Ingenierías: resúmenes de comunicaciones [en línea]. Alicante: Universidad de Alicante, 1996. vol. nro. 2. p. 840-847. [Consultado: 1 de noviembre de 2020]. Disponible en http://www.dccia.ua.es/~faraon/docs/programacion.pdf

SCHULTE, Carsten y BENNEDSEN, Jens. International Computing Education Research Workshop [en línea]. En: ICER '06: Proceedings of the second international workshop on Computing education research. (septiembre. Nueva York, Estados Unidos). What do teachers teach in introductory programming?. Nueva York. 2006. p. 17-28. [Consultado el 12 de abril de 2021]. Disponible en: https://www.researchgate.net/publication/234772749_What_do_teachers_teach_in_introductory_programming

SCHWABER, Ken y SUTHERLAND, Jeff. La Guía Definitiva de Scrum: Las Reglas del Juego. En: SCRUM GUIDES [sitio web]. Estados Unidos: ScrumGuides. Serie "SCRUM Guide". [Consultado: 24 de abril de 2021]. Archivo pdf. Disponible en: https://scrumguides.org/docs/scrumguide/v2016/2016-Scrum-Guide-Spanish.pdf#zoom=100

SHACKELFORD, Russell y LEBLANC, Richard. Teaching and Learning in an Era of Change [en línea]. En: Proceedings Frontiers in Education 1997 27th Annual Conference. (5-8, noviembre. Pennsylvania, Estados Unidos). Introducing Computer Science Fundamentals Before Programming. Illinois: Stipes Publishing. 1997. p. 285-289. [Consultado: 29 de septiembre de 2020]. Disponible en: https://ieeexplore.ieee.org/document/644858

SHNEIDERMAN, Ben et al. Experimental Investigations of the Utility of Detailed Flowcharts in Programming. En: Communications of the ACM [en línea]. Estados Unidos: Universidad de Indiana, junio de 1977. vol. 20, nro. 6. p. 373-381. [Consultado el 2 de Octubre de 2020]. Disponible en: https://www.cs.umd.edu/~ben/papers/Shneiderman1977Experimental.pdf

SIEGEL, Eric. Innovation and Technology in Computer Science Education [en línea]. En: ITiCSE '99: Proceedings of the 4th annual SIGCSE/SIGCUE ITiCSE conference on Innovation and technology in computer science education. (junio. Cracow, Polonia). Why Do Fools Fall Into Infinite Loops: Singing To Your Computer Science Class. Cracow. 1999. p. 167–170. [Consultado el 1 de noviembre de 2020]. Disponible en: http://www.predictionimpact.com/Siegel-educational-computer-songs.pdf

STRAWN, George y STRAWN, Candace. Grace Hopper: Compilers and Cobol. En: IT Professional [en línea]. enero-febrero, 2015. vol. 17, nro. 1. p. 62-64. [Consultado el 1 de noviembre de 2020]. Disponible en https://ieeexplore.ieee.org/abstract/document/7030179. DOI: 10.1109/MITP.2015.6

TINOCO, Oscar; ROSALES, Pedro y SALAS, Julio. Criterios de selección de metodologías de desarrollo de software. En: Industrial Data [en línea]. Lima: Universidad Nacional Mayor de San Marcos, julio de 2010. vol. 13, nro. 2. p. 70-74. [Consultado el 1 de noviembre de 2020]. Disponible en: https://www.redalyc.org/pdf/816/81619984009.pdf. ISSN: 1560-9146

Universidad de los Andes. REVIVA EL TERCER FORO: DÉFICIT EN TALENTO TI [video]. Bogotá: YouTube, Universidad de los Andes. (2 de septiembre de 2019). 189:08 minutos. [Consultado: 2 de octubre de 2020]. Disponible en: https://uniandes.edu.co/es/noticias/ingenieria/tercer-foro-deficit-en-talento-ti

UNIVERSIDAD DE LOS ANDES: Departamento de Sistemas. Arquitectura de Software: Punto de Vista Funcional [diapositivas]. Repositorio uniandes. Bogotá, 79 diapositivas. [Consultado 24 de abril de 2021]. Disponible en: https://profesores.virtual.uniandes.edu.co/~arti4001/dokuwiki/lib/exe/fetch.php?media=principal:calendario:modulo5-vistafuncional.pdf

UNIVERSITAT JAUME I. Procesadores de Lenguaje: Estructura de los compiladores e interpretes. En: Universidad Jaime I de Castellón [en línea]. Castellón de la Plana: Universidad Jaime I, 2010-2011. vol. , nro. p. 1-9. [Consultado: 1 de noviembre de 2020]. Disponible en http://repositori.uji.es/xmlui/bitstream/handle/10234/22656/II26_estructura_compiladores.pdf?sequence=1

XIE, Jianjun; LI, Junhong y GENG, Yi. International Conference on Computer Science and Software Engineering [en línea]. En: 2008 International Conference on Computer Science and Software Engineering. (12-14, diciembre. Wuhan, China). Blended Learning in Computer Hardware Course Education. 2008. p. 676-680. [Consultado el 12 de abril de 2021]. Disponible en: https://ieeexplore.ieee.org/document/4723051

18. ANEXOS

18.1 MARCO HISTORICO

18.1.1 Java.

Según Deitel[110] la historia de java parte desde el origen de microprocesadores los cuales permitieron la fabricación de computadoras personales lo cual tuvo consecuencias favorables en las empresas y hogares de todo el mundo. En vista de ello, Sun Microsystems apoyó el proyecto de investigación Green en 1991. Este proyecto produjo un lenguaje basado en C++ el cual fue llamado Java (una variedad de café) por su creador James Gosling. La comercialización de computadoras personales no se desarrolló tan rápido y es por ello que el proyecto Green estuvo a punto de ser cancelado. Dos años más tarde, la tendencia de World Wide Web permitió que se incorporará Java como una herramienta para agregar contenido dinámico en páginas web. Finalmente, para 1995 Java fue anunciado formalmente por Sun Microsystems. A lo largo de los años java ha tenido múltiples contribuciones en productos de software (tanto privados como libres) lo cual ha enriquecido a la comunidad de programadores en usar estos como librerías de desarrollo. Actualmente java se utiliza para el desarrollo de aplicativos de gran escala empresarial, para mejorar la funcionalidad de los servicios Web, para el desarrollo de aplicativos domésticos y otros más.

18.1.2 Intérprete.

Como lo menciona McCarthy[111] Lisp es probablemente el primer intérprete para un lenguaje de alto nivel creado en la historia, este fue implementado por Steve Rusell en 1958 para una computadora IBM 704. Este proyecto se desarrolló con el fin de evaluar notaciones matemáticas y obtuvo una gran repercusión en la comunidad investigadora de inteligencia artificial. Pero su creador fue John McCarthy[112] en 1958 mientras estudiaba en el MIT. A lo largo del tiempo se han desarrollado otros intérpretes con el fin de ser involucrados en el desarrollo de software y por ende con lenguajes de alto nivel. Este elemento será recreado o simulado con el propósito de interpretar pseudocódigo en el proyecto.

18.1.3 Compilador.

---

[110] DEITEL, Op. cit., p. 8

[111] MCCARTHY, John. LISP Session. En: Association for Computing Machinery [en línea]. Nueva York: Wexelblat, Richard L, 1978. vol. , nro. pp. 172-185. [Consultado el 1 de noviembre de 2020]. Disponible en: https://www.labouseur.com/courses/tpl/History%20of%20LISP-mccarthy.pdf. ISBN: 0-12-745040-8

[112] MCCARTHY, Op. cit., p. 173

Según Strawn[113], Admiral Grace Murray Hopper (1906-1992) es una de las mujeres más importantes en la historia de la computación, la cual fue la responsable de construir el primer compilador en el mundo. En su vida académica recibió el título de doctorado en matemáticas y para cuando estalló la segunda guerra mundial se enlistó en la marina con el fin de cooperar en proyectos navales involucrados. Posteriormente trabajó para Harvard donde conoció a Howard Aiken quien luego convencería a IBM de construir un computador electromagnético llamado Mark I. Ella fue la primera persona en tener la idea de que un computador se debe comunicar como un humano y no viceversa, esta idea se materializó en 1952 con la creación del compilador A-0 (lenguaje aritmético - 0). Sus trabajos se desarrollaron posteriormente en los lenguajes verdaderamente compilados como Math-Matic y Flow-Matic, generando un gran impacto en la tecnología ya que IBM en 1957 desarrolló el lenguaje Fortran (formula translation) y su compilador. Por otro lado, un equipo para las ciencias de computo desarrolló el lenguaje Algol y sus compiladores en 1958. Su idea de un lenguaje similar al inglés se extendió hasta el desarrollo de Flow-Matic el cual fue un éxito. IBM entró en el negocio de los lenguajes empresariales con Comtran para aprovechar el uso de un estándar comercial de lenguajes. Y debido a la necesidad de ahorrar costos de producción de software (dada la complejidad de desarrollo en lenguajes máquina) se le incluyó a Hopper en el programa de desarrollo de COBOL (common business-oriented languages). Cobol popularizó esta tendencia de compiladores debido a que este se estableció como estándar. Este elemento se utilizará no en completitud de todas sus partes sino en la aplicación de este concepto funcional para el pseudocódigo.

18.1.3 Analizador Léxico-Morfológico.

Este analizador en principio según Strawn[114] estaba declarado únicamente en expresiones aritméticas en 1952 con el compilador A-0 (creado por Grace Hopper). Y se desarrolló plenamente con enfoque matemático en Math-Matic y Flow-Matic en 1957 (desarrollado por IBM). Pero solo hasta 1958 evolucionó en el sentido el cual se usa actualmente, donde su propósito va dirigido hacia los algoritmos. Su desarrollo formal se estableció en COBOL como lenguaje de desarrollo orientado al ámbito empresarial. Se utilizará este elemento desde el propósito inicial cuya función recae en expresiones lógicas y aritméticas.

18.1.4 Analizador Sintáctico.

Este analizador como lo comenta Strawn[115] estaba declarado únicamente en expresiones aritméticas 1952 creado por Grace Hopper (compilador A-0) y tuvo versiones posteriores en compiladores matemáticos para lenguajes como Math-

---

[113] STRAWN, George y STRAWN, Candace. Grace Hopper: Compilers and Cobol. En: IT Professional [en línea]. enero-febrero, 2015. vol. 17, nro. 1. p. 62-64. [Consultado el 1 de noviembre de 2020]. Disponible en https://ieeexplore.ieee.org/abstract/document/7030179. DOI: 10.1109/MITP.2015.6

[114] Ibíd., p. 63.

[115] Ibíd., p. 63.

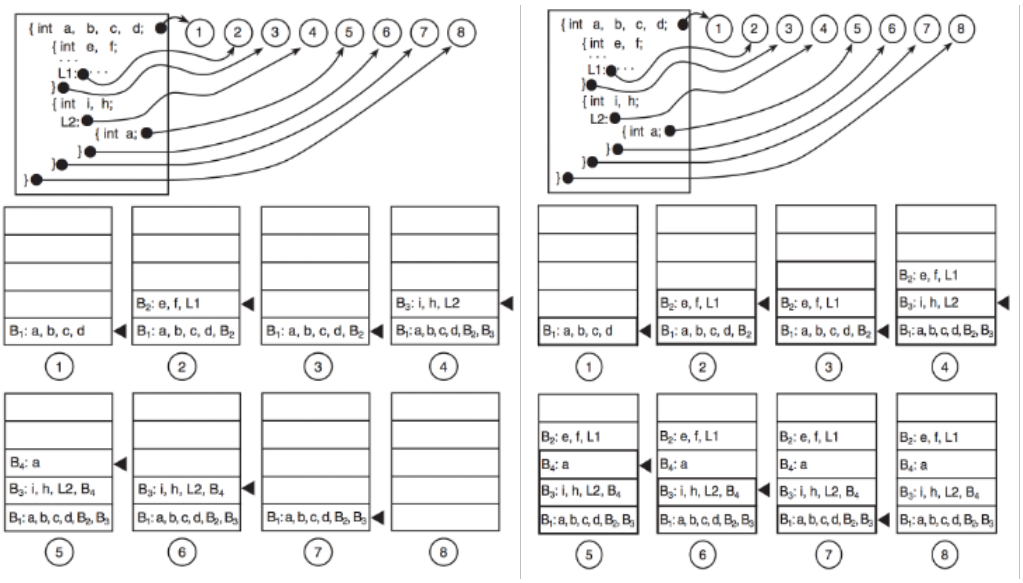
Matic y Flow-Matic (desarrollados por IBM en 1957). Pero dio sus primeros pasos al concepto moderno funcional solo hasta el surgimiento de Fortran en el sentido dirigido hacia los algoritmos. Posteriormente su función fue de gran utilidad hasta el surgimiento de lenguajes de alto nivel. Pero este elemento se utilizará desde su función inicial para expresiones aritméticas y lógicas.

18.1.5 Tabla de Símbolos.

Este recurso como lo menciona Strawn[116] apareció por primera vez como una tabla de identificadores implícito en el compilador A-0 creado por Grace Hopper (1952). Desde entonces no ha tenido muchos cambios debido a que su función no se encuentra limitada al contexto. Es igual de necesario para expresiones aritmeticas y logicas como para lenguajes de alto nivel. Sus variaciones no se encuentran definidas en un contexto histórico claro, pero a pesar de esto se harán mención de estas. Como lo menciona Alfonseca[117] este elemento puede estar definido en una búsqueda por comparación de claves usando árboles AVL o en el uso de tablas hash. El uso más eficiente entre estas dos alternativas es la última, la cual se despliega en diversas presentaciones según Alfonseca[118], estas son: tabla dinámica, tabla estática y tabla con encadenamiento. Para esta propuesta se utilizará una tabla con encadenamiento para evitar colisiones. El uso de esta puede variar según el ámbito como lo muestra la *Figura 25*. Como el presente proyecto tiene un alcance definido para algoritmos cortos (sin invocar métodos externos) se utilizará una tabla para todos los ámbitos.

Figura 25. Uso de Tabla de Simbolos para cada ámbito (izquierda) y uso de Tabla de Simbolos para todos los ámbitos (derecha).

[116] Ibíd., p. 63.
[117] ALFONSECA, Op. cit., p. 37.
[118] ALFONSECA, Op. cit., p. 44.



Fuente ALFONSECA MORENO, Manuel et al. Compiladores e intérpretes: teoría y práctica

18.1.6 Pedagogía en cursos introductorios a la programación.

En los cursos introductorios a la programación se han elaborado diversos métodos con los cuales se busca explicar por medios dinámicos conceptos dinámicos. Ejemplos de estas metodologías son las empleadas por Astrachan[119] en donde se utilizaron frisbees con el objetivo de enseñar el envío de parámetros, se utilizaron bloques de construcción didáctica para la enseñar listas enlazadas y ganchos de goma (icky-poo) para enseñar el funcionamiento de los punteros. Otro ejemplo de estos métodos son los empleados por Siegel[120] donde utilizan canciones para enseñar errores comunes en la programación, el concepto de abstracción y punteros. Sin embargo, en tiempos modernos es más común ver el uso de software educativo ya que genera una gran aceptación en los entornos académicos. Ejemplos de esto se evidencian en documentos escritos por Carrizo[121], Anderson[122], Redondo[123], Del Prado[124], Gomes[125], Marcelino[126] y Buck[127].

18.1.7 Indentación.

---

[119] ASTRACHAN, Op. cit., p. 22.
[120] SIEGEL, Op. cit., p. 167.
[121] CARRIZO, Op. cit., p. 1.
[122] ANDERSON, Op. cit., p. 91.
[123] REDONDO, Op. cit., p.1.
[124] DEL PRADO, Op. cit., p.1.
[125] GOMES, Op. cit., p.1.
[126] MARCELINO, Op. cit., p. IV.8-1.
[127] BUCK, Op. cit., p. 1.

A lo largo de la historia en ciencias de la computación se han definido diversas técnicas de indentación las cuales puedan contribuir con la comprensión del código. Como lo menciona Miara[128], este recurso es ideal para los principiantes en el mundo de la programación, así mismo, el autor expone las líneas conectoras como técnica. Otra técnica utilizada es Countour como lo plantea Gimpel[129] el cual trata de expresar el código en bloques definidos por márgenes (similar a un mapa geológico) para identificar la profundidad del código en su ejecución. Más sin embargo, en este proyecto se utilizará la propuesta de Leinbaugh[130] el cual emplea una única regla para la indentación, esta es común en lenguajes como Python. Su elección se deriva a su simplicidad.

18.1.8 Heurística de George Polya.

Como ya es descrito el concepto de heurística en el marco conceptual, este puede presentarse en diversas metodologías. Una de las más relevantes es la heurística de George Polya[131]. La cual se plantea en 4 etapas según Polya[132], donde la primera se enfoca en comprender el problema tratando de visualizar todo su contexto; la segunda, involucra la concepción de un plan el cual será una estrategia para la solución; la tercera, es la puesta en marcha de la estrategia en la fase anterior; la cuarta, es la retroalimentación de la estrategia la cual puede estar sujeta a la modificación del plan. En este proyecto se utilizará esta metodología como componente pedagógico así como enfoque para el aplicativo (en las 3 últimas etapas), debido a que en las variables obtenidas de las dificultades en cursos introductorios a la programación se encuentra la detección de errores.

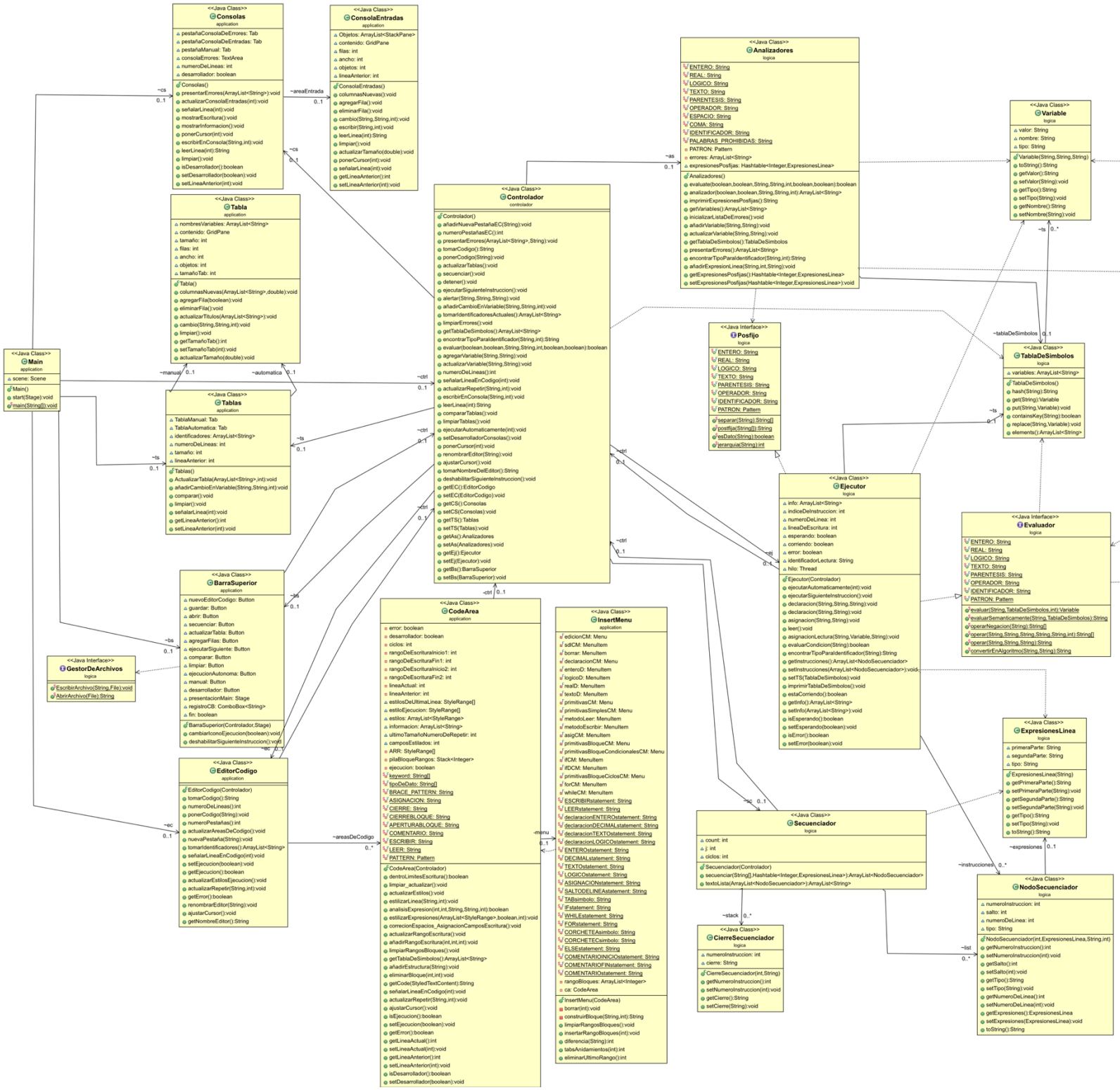
---

[128] MIARA, Op. cit., p. 863.

[129] GIMPEL, James. Contour: A Method of Preparing Structured Flowcharts. En: SIGPLAN Not. [en línea]. Nueva York, octubre de 1980. vol. 15, nro. 10. p. 35–41. [Consultado el 1 de noviembre de 2020]. Disponible en https://dl.acm.org/doi/10.1145/947727.947730. ISSN: 0362-1340

[130] LEINBAUGH, Dennis. Indenting for the Compiler. En: Association for Computing Machinery [en línea]. Nueva York, mayo de 1980. vol. 15, nro. 5. p. 41–48. [Consultado el 1 de noviembre de 2020]. Disponible en: https://dl.acm.org/doi/10.1145/947639.947644. ISSN: 0362-1340

[131] POLYA, Op. cit., p. 28.

[132] POLYA, Op. cit., p. 28, 30, 33, 35.

18.2 DIAGRAMA DE CLASES