

Desarrollo de una herramienta interactiva para apoyar la enseñanza de Fundamentos de Interpretación y Compilación de LP mediante lenguajes orientados por sintaxis

Jota Emilio López Ramírez, Esmeralda Rivas Guzmán

Universidad del Valle

Facultad de Ingeniería

jota.lopez@correounalvalle.edu.co,

esmeralda.rivas@correounalvalle.edu.co

Resumen

El presente anteproyecto propone el diseño y desarrollo de una herramienta interactiva orientada a facilitar la comprensión de los fundamentos de interpretación y compilación de lenguajes de programación. La herramienta permitirá a los estudiantes definir gramáticas, visualizar estructuras sintácticas y explorar de manera interactiva la evaluación de programas, apoyando el aprendizaje autónomo y la comprensión de conceptos complejos mediante representaciones gráficas y mecanismos didácticos.

El proyecto se enmarca en teorías del aprendizaje constructivista y principios de usabilidad para herramientas educativas, buscando servir como un recurso complementario en la asignatura de *Fundamentos de Interpretación y Compilación de Lenguajes de Programación* de la Universidad del Valle.

Palabras clave

compiladores, intérpretes, herramienta educativa, visualización de procesos

1. INTRODUCCIÓN

El presente anteproyecto tiene como propósito establecer las bases teóricas y metodológicas del trabajo de grado propuesto. Se expone el contexto general, la motivación y la relevancia del problema a abordar.

El proceso de enseñanza del diseño de compiladores ha sido abordado desde múltiples enfoques pedagógicos. Por ejemplo, Kundra y Sureka [1] presentan un enfoque basado en aprendizaje por proyectos y casos para mejorar la comprensión de los conceptos de compiladores. De manera complementaria, Vegdahl [2] propone el uso de herramientas de visualización para apoyar el aprendizaje de los procesos internos del compilador.

Otros autores como Baldwin [3] destacan la utilidad de lenguajes simplificados y compiladores modulares para facilitar la enseñanza práctica del tema, mientras que Mernik y Žumer [4] desarrollaron una herramienta educativa específica para la construcción de compiladores.

La importancia de los lenguajes de programación como marco conceptual para el aprendizaje ha sido resaltada desde los primeros trabajos de Feurzeig, Papert y Lawler [5], quienes vinculan la programación con la enseñanza de las matemáticas desde un enfoque constructivista.

Finalmente, investigaciones más recientes como la de Stamenković y Jovanović [6] exploran sistemas web interactivos que permiten la enseñanza de compiladores a través de entornos digitales, integrando elementos de visualización, automatización y simulación educativa.

Estas contribuciones conforman el marco de referencia que sustenta la relevancia y viabilidad del presente proyecto, al evidenciar la evolución de las estrategias pedagógicas aplicadas al ámbito de los compiladores y su enseñanza.

2. FORMULACIÓN DEL PROBLEMA

2.1. Descripción del Problema

En el programa de Ingeniería de Sistemas de la Universidad del Valle, la asignatura *Fundamentos de Interpretación y Compilación de Lenguajes de Programación* (FLP) es un componente esencial para el desarrollo de competencias relacionadas con la comprensión de los procesos internos de los lenguajes de programación. Sin embargo, se ha evidenciado que los estudiantes presentan dificultades para asimilar y aplicar los conceptos abordados, los cuales presentan un alto nivel de abstracción y demandan una sólida comprensión teórica.

Entre las principales causas de esta situación se encuentra la falta de herramientas didácticas que faciliten la visualización práctica de los entornos de ejecución, los cambios en variables y los mecanismos de interpretación durante la evaluación de un programa. Aunque la asignatura se apoya en el lenguaje *Racket* y en la librería *EOPL*, que permiten representar estructuras internas como árboles de sintaxis abstracta (AST) y explorar el funcionamiento del intérprete, estas funcionalidades resultan útiles pero su aprovechamiento requiere un dominio técnico previo del entorno y carecen de un componente visual o interactivo que fomente el aprendizaje autónomo.

Diversos estudios han señalado que el uso de herramientas interactivas y representaciones visuales favorece la comprensión de los procesos de compilación e interpretación, al permitir que los estudiantes observen de forma tangible las transformaciones que ocurren en las distintas etapas del procesamiento de un lenguaje [1], [2], [4], [6]. Estos enfoques, centrados en la visualización y la experimentación práctica, han demostrado ser efectivos para reducir la abstracción inherente a los contenidos y promover un aprendizaje más significativo. Además, se ha evidenciado que, si bien los lenguajes reales y sus intérpretes ofrecen una base sólida para la enseñanza, su orientación hacia la eficiencia y la complejidad técnica dificulta su aprovechamiento con fines pedagógicos, al no exponer claramente los procesos internos que llevan a la ejecución del código [7].

2.2. Definición del Problema

La ausencia de herramientas didácticas interactivas que complementen los recursos actuales y faciliten la exploración visual de los procesos de interpretación y compilación ha provocado que los estudiantes mantengan dificultades persistentes en la comprensión profunda de los fundamentos de la asignatura. Esta situación se traduce en una dependencia de clases magistrales, una limitada autonomía en el aprendizaje,

la adquisición superficial de conocimientos y un bajo desempeño en proyectos finales o en asignaturas posteriores que requieren estos conceptos.

En este contexto, surge la siguiente pregunta de investigación:

“¿Cómo puede el desarrollo de una herramienta interactiva facilitar la comprensión y aplicación de los contenidos de la asignatura Fundamentos de Interpretación y Compilación de Lenguajes de Programación de la Universidad del Valle, mediante la visualización de procesos internos y la generación de lenguajes orientados por sintaxis?”

3. MARCO DE REFERENCIA

3.1. Marco Teórico

El proyecto se fundamenta en tres áreas de conocimiento: teoría de lenguajes formales y compiladores que proporcionan los fundamentos técnicos; teorías del aprendizaje aplicadas a la educación en ciencias de la computación que informan sobre el diseño pedagógico y determinan las estrategias de representación; y principios de usabilidad e interactividad en herramientas educativas que orientan las decisiones de diseño de interfaz.

3.1.1. Fundamentos de Lenguajes Formales y Teoría de Compiladores: Los lenguajes de programación se definen formalmente mediante gramáticas libres de contexto (Context-Free Grammars, CFG), clasificadas por Chomsky en el tipo dos de su jerarquía de lenguajes formales [8]. Estas gramáticas establecen reglas de producción que determinan cómo los símbolos no terminales pueden reemplazarse por secuencias de símbolos terminales y no terminales, proporcionando una descripción precisa y estructurada de la sintaxis de un lenguaje.

El análisis sintáctico (parsing) verifica si una cadena pertenece al lenguaje definido por la gramática y construye una representación estructural del código fuente. Este proceso genera el árbol de sintaxis abstracta (AST), el cual pasa por alto detalles superficiales y conserva la estructura semántica esencial del programa [9]. El AST actúa como puente entre la sintaxis y las etapas posteriores del compilador, facilitando el análisis semántico, la optimización y la generación de código.

La interpretación constituye una forma de ejecución en la cual un intérprete recorre el AST evaluando cada nodo según reglas semánticas y utilizando entornos de ejecución (environments) que asocian identificadores con valores [10]. Finalmente, los lenguajes orientados por sintaxis (syntax-directed languages) establecen una relación formal entre la estructura gramatical y el significado del programa mediante definiciones y esquemas de traducción dirigidos por la sintaxis (Syntax-Directed Definitions y Syntax-Directed Translation Schemes) [9].

En conjunto, estos fundamentos constituyen la base técnica sobre la cual se estructura la enseñanza de los compiladores, permitiendo que los estudiantes comprendan el proceso de traducción y ejecución desde una perspectiva formal y visual.

3.1.2. Teorías del Aprendizaje Aplicadas a la Educación en Computación: El diseño pedagógico de este proyecto se fundamenta en teorías del aprendizaje que promueven una enseñanza activa y significativa. El constructivismo, propuesto por Piaget y Vygotsky, plantea que el conocimiento se construye a través de la interacción con el entorno y la reflexión sobre la experiencia [11], [12]. En la educación en computación, esto se traduce en metodologías centradas en la experimentación y la creación de artefactos funcionales,

donde los estudiantes comprenden mejor los conceptos de compiladores al definir gramáticas, implementar intérpretes y observar los resultados de sus decisiones.

Complementariamente, la teoría de la carga cognitiva, propuesta por Sweller, plantea que la memoria de trabajo posee una capacidad limitada para procesar nueva información [13]. Distingue entre carga cognitiva intrínseca (complejidad del contenido), extrínseca (forma de presentación) y germana (procesos mentales que favorecen el aprendizaje). En el caso de herramientas educativas para compiladores, la carga intrínseca es alta debido a la abstracción del contenido. Por ello, es fundamental que el diseño reduzca la carga extrínseca mediante interfaces claras y visualizaciones que actúen como apoyo externo a la memoria de trabajo.

Asimismo, el enfoque del aprendizaje por descubrimiento guiado, propuesto por Bruner, sostiene que los estudiantes adquieren conocimientos de manera más duradera cuando los descubren por sí mismos bajo una guía estructurada [14]. En el contexto de una herramienta educativa, esto implica permitir la exploración controlada mediante ejemplos incrementales, plantillas predefinidas y retroalimentación constante, facilitando que los estudiantes comprendan los efectos de sus decisiones al modificar gramáticas o estructuras sintácticas.

Finalmente, la teoría del aprendizaje multimedia de Mayer establece principios sobre cómo combinar texto, imágenes y elementos interactivos para optimizar la comprensión [15]. Entre ellos se destacan la modalidad (usar canales visuales y verbales complementarios), la contigüidad (presentar simultáneamente información relacionada), la coherencia (eliminar elementos irrelevantes) y la señalización (resaltar lo esencial). Estos principios orientan el diseño de visualizaciones didácticas manteniendo una correspondencia clara entre acciones y efectos.

3.1.3. Principios de Diseño de Herramientas Educativas Interactivas: El desarrollo de herramientas educativas interactivas requiere integrar criterios de usabilidad, interactividad y progresión pedagógica. La usabilidad, entendida como la capacidad de un sistema para ser utilizado de forma efectiva, eficiente y satisfactoria, resulta fundamental en entornos de aprendizaje digital.

Investigaciones recientes destacan que la usabilidad no solo incide en la eficiencia técnica, sino también en la motivación y la retención del aprendizaje, al permitir que los usuarios concentren sus recursos cognitivos en la comprensión del contenido y no en la manipulación de la interfaz [16]. En este sentido, una interfaz clara, accesible y coherente contribuye a reducir la carga cognitiva extrínseca y a fomentar la autonomía en el aprendizaje.

La interactividad se concibe como un proceso de acción, retroalimentación y reflexión en el que el usuario explora y construye conocimiento activamente [14]. Las herramientas que proporcionan retroalimentación mediante la visualización del estado de un programa o la señalización de errores en tiempo real favorecen la corrección autónoma y el aprendizaje autorregulado.

Por último, la progresión y el andamiaje, derivados del constructivismo social de Vygotsky, enfatizan el apoyo temporal que se ofrece al estudiante durante el aprendizaje, el cual se retira gradualmente a medida que alcanza una comprensión más profunda [12]. En una herramienta para la enseñanza de intérpretes, estos principios pueden materializarse mediante ejemplos guiados, ejercicios incrementales y restricciones adaptativas, promoviendo un aprendizaje autónomo y sostenido.

3.2. Estado del Arte

En la última década han surgido múltiples iniciativas orientadas a reducir la abstracción inherente a la enseñanza de compiladores e intérpretes mediante visualizaciones, simulaciones paso a paso y entornos web interactivos que permiten a los estudiantes manipular gramáticas, construir árboles de sintaxis y seguir la evaluación de programas de manera controlada. Estas soluciones buscan transformar conceptos teóricos (análisis léxico, parsing, construcción de AST, entornos de ejecución y traducción dirigida por sintaxis) en objetos interactivos con los cuales el estudiante puede experimentar [17]–[19].

Entre las contribuciones recientes más relevantes se encuentran los entornos de simulación y visualización diseñados específicamente para la docencia en compiladores, las herramientas de exploración de compiladores empleadas en contextos educativos y los estudios empíricos sobre generadores de analizadores léxicos y sintácticos. En particular, el sistema *ComVIS* ofrece un conjunto integrado de simulaciones (construcción de autómatas, tablas LL/LR, simulación de parsing y visualización del flujo de transformación) pensado para su uso en cursos y ejercicios guiados [17], [20]. De igual modo, trabajos recientes sobre tutores interactivos muestran beneficios en la comprensión cuando se combinan implementaciones concretas con visualizaciones que conectan teoría y práctica [18]. Por su parte, herramientas ampliamente utilizadas por la comunidad, como *Compiler Explorer*, aportan valor didáctico al permitir observar las etapas intermedias de compilación y el código ensamblador, aunque están orientadas principalmente a usuarios avanzados y no a procesos de aprendizaje guiado paso a paso [21].

La literatura reciente también presenta evaluaciones comparativas y estudios empíricos sobre herramientas educativas. Algunos trabajos analizan el uso de generadores de analizadores como ANTLR frente a Lex/Yacc y evalúan marcos educativos como ChocoPy o CakeML, documentando ventajas y limitaciones para la docencia práctica de compiladores [19], [22], [23]. Adicionalmente, revisiones sistemáticas sobre usabilidad en tecnologías educativas destacan la importancia del diseño centrado en el usuario, la evaluación empírica de interfaces y la necesidad de instrumentar métricas que midan carga cognitiva y aprendizaje efectivo [24].

3.2.0.1. Limitaciones y vacíos: A partir del análisis de la literatura y de los sistemas existentes se identifican vacíos relevantes para proyectos docentes: (1) la mayoría de las soluciones aborda fases concretas del proceso de compilación, sin ofrecer un flujo integrado que combine definición de gramáticas, visualización de AST y evaluación interactiva con andamiaje pedagógico; (2) persiste una tensión entre potencia técnica y usabilidad didáctica: los entornos profesionales son robustos pero poco accesibles para estudiantes novatos, mientras que las herramientas didácticas suelen carecer de flexibilidad para explorar características avanzadas; y (3) la evidencia empírica contextualizada en universidades latinoamericanas o de habla hispana es escasa, lo que demanda mayor trabajo de adaptación y validación local [17], [22], [24].

3.2.0.2. Contribución proyectada: A partir de las brechas identificadas en la literatura, este proyecto propone el desarrollo de una herramienta interactiva con fines exclusivamente pedagógicos, orientada a apoyar la enseñanza de los fundamentos de interpretación y compilación de lenguajes de programación. La propuesta busca integrar en un entorno web accesible la definición de gramáticas, la visualización de estructuras sintácticas, permitiendo a los estudiantes explorar de forma guiada los procesos internos de interpretación y compilación.

La herramienta se fundamenta en principios de aprendizaje constructivista y en lineamientos de usabilidad educativa, con el propósito de reducir la carga cognitiva asociada a los contenidos abstractos y favorecer la comprensión significativa de los conceptos. Además, busca fomentar el aprendizaje autónomo mediante actividades interactivas y visualizaciones didácticas que sirvan como complemento a las clases teóricas y prácticas de la asignatura.

En conjunto, esta contribución pretende ofrecer un recurso educativo contextualizado al entorno académico de la Universidad del Valle, que promueva el aprendizaje activo de los estudiantes en temas de compiladores e intérpretes [17], [18], [24].

3.3. Antecedentes

En la asignatura *Fundamentos de Interpretación y Compilación de Lenguajes de Programación* de la Universidad del Valle se han empleado tradicionalmente entornos y bibliotecas —como Racket y la colección EOPL— que permiten a los estudiantes implementar intérpretes y explorar estructuras internas como árboles de sintaxis abstracta (AST). Aunque estas herramientas han facilitado el trabajo práctico, su uso requiere familiaridad con entornos de programación específicos, lo que puede representar una barrera inicial para algunos estudiantes. Este escenario ha impulsado la búsqueda de soluciones más accesibles y visuales adaptadas al contexto académico local.

A nivel internacional, los desarrollos recientes mencionados en el Estado del Arte (como *ComVIS*, tutores interactivos, herramientas de exploración de compiladores y estudios empíricos sobre generadores de parsers) constituyen antecedentes tecnológicos y metodológicos directos que orientan las decisiones de diseño de la propuesta: arquitectura web, simulaciones paso a paso y mecanismos de retroalimentación inmediata. La conjunción entre el contexto formativo local y la evidencia internacional justifica el desarrollo de una herramienta interactiva que integre definición de gramáticas, visualización de AST y ejecución paso a paso con un enfoque didáctico [17], [18], [22].

3.4. Marco Conceptual

3.4.1. Proceso de Interpretación: La interpretación de lenguajes de programación constituye un proceso central en la ejecución de programas, integrando varios niveles de abstracción. Comienza con la definición formal de la sintaxis mediante gramáticas libres de contexto, las cuales determinan qué secuencias de símbolos forman programas válidos [9]. Estas gramáticas, compuestas por reglas de producción entre símbolos terminales y no terminales, garantizan precisión y evitan ambigüedades que puedan afectar la interpretación del código.

El análisis sintáctico transforma el código fuente en un árbol de sintaxis abstracta (AST), es decir, en una representación estructural que revela la jerarquía semántica del programa [9]. Este modelo facilita la comprensión de cómo las expresiones se componen y relacionan, aportando al aprendizaje de compiladores e intérpretes.

La interpretación consiste en recorrer el AST ejecutando las operaciones definidas en sus nodos [10]. Para ello, se mantienen ambientes de ejecución que asocian identificadores con valores, modelando fenómenos como el alcance léxico y las clausuras. La estructura jerárquica de estos ambientes explica la organización de variables locales y globales, el paso de parámetros y la recursión, aspectos esenciales para comprender el comportamiento del intérprete.

3.4.2. Lenguajes Orientados por Sintaxis como Paradigma Educativo: Los lenguajes orientados por sintaxis definen la semántica del programa en función de su estructura sintáctica mediante atributos asociados a los símbolos de la gramática [9]. Este enfoque, formalizado en las definiciones dirigidas por sintaxis, establece cómo se deriva el significado de cada construcción a partir de sus componentes.

En el ámbito educativo, este paradigma permite visualizar la relación entre la forma del código y su comportamiento. Los atributos sintetizados y heredados representan distintos flujos de información dentro del árbol sintáctico, desde las partes hacia el todo o desde el contexto hacia las partes, facilitando la comprensión de la evaluación y traducción de programas.

4. ALCANCE DEL PROYECTO

4.1. Declaración del Alcance

El presente proyecto tiene como alcance el diseño, desarrollo y evaluación de usabilidad de una herramienta interactiva orientada a facilitar la comprensión de conceptos fundamentales de interpretación y compilación de lenguajes de programación, específicamente como apoyo al proceso de enseñanza-aprendizaje dentro de la asignatura *Fundamentos de Interpretación y Compilación de LP* de la Universidad del Valle.

4.1.1. Alcance del Producto: La herramienta permitirá la definición y análisis de gramáticas que describan la estructura formal de lenguajes orientados por sintaxis. Asimismo, integrará la representación visual de las estructuras sintácticas (AST) derivadas de los programas escritos por el usuario, con el fin de facilitar la comprensión de su organización interna.

La interfaz será accesible y promoverá la exploración autónoma mediante ejemplos predefinidos que ilustren conceptos centrales de la asignatura. Finalmente, se habilitará el almacenamiento local de gramáticas y programas definidos por el usuario, permitiendo su consulta y reutilización en sesiones posteriores.

4.1.2. Alcance del Proyecto: El proyecto se estructura en cuatro fases principales que se ejecutarán dentro del período de ocho meses establecido. La fase de análisis comprende la revisión de literatura relevante, la consulta con el docente de la asignatura y la identificación de los requisitos funcionales y no funcionales del sistema. Esta fase incluye también el reconocimiento de las dificultades más comunes que enfrentan los estudiantes, con el fin de orientar la selección y construcción de ejemplos que acompañarán la herramienta, teniendo en cuenta que las funcionalidades principales como la visualización del AST y el manejo de lenguajes orientados por sintaxis no dependen de fuentes externas adicionales.

La fase de diseño abarca la elaboración de la arquitectura del sistema, el diseño de interfaces de usuario mediante prototipos y la definición de estrategias de visualización adecuadas para explicar conceptos abstractos. Esta fase considerará explícitamente la curva de aprendizaje asociada con las tecnologías seleccionadas, dedicando tiempo suficiente para que el equipo desarrolle competencia técnica antes de proceder a la implementación.

La fase de implementación consiste en el desarrollo del prototipo funcional con base en los requisitos y diseños establecidos, integrando el análisis de gramáticas, la generación de árboles sintácticos abstractos (AST), la interfaz de usuario y los mecanismos de visualización interactiva.

La fase de evaluación comprende la validación de usabilidad mediante pruebas con estudiantes y el docente de la asignatura. Dada la restricción temporal y la necesidad de coordinar con el calendario

académico, la evaluación se realizará durante un período académico específico. Se hará a través de cuestionarios de usabilidad y observación de uso con una muestra estimada entre 10 y 20 estudiantes. Esto es suficiente para identificar problemas de interacción y oportunidades de mejora sin pretender medir el impacto en el aprendizaje formal.

4.1.3. *Fuera del Alcance:* El proyecto no contempla la compilación a código ejecutable o generación de binarios. No se incluirán mecanismos de optimización de programas ni análisis de rendimiento de código. Debido a las restricciones temporales del proyecto, no será posible realizar estudios experimentales que midan el impacto en el aprendizaje a largo plazo, ya que estos requieren de seguimiento durante múltiples períodos académicos.

No se implementará un sistema de gestión de usuarios con autenticación o capacidades de colaboración multiusuario, optándose por almacenamiento local para simplificar el desarrollo y reducir requisitos de infraestructura. La visualización del proceso de evaluación de programas paso a paso, queda fuera del alcance debido a su complejidad de implementación. Finalmente, no se contempla infraestructura de despliegue de tipo productivo con alta disponibilidad, limitando el despliegue a un entorno de pruebas suficiente para la evaluación con estudiantes.

4.1.4. *Entregables del Proyecto:*

1. **Herramienta interactiva funcional:** Sistema desplegado en un entorno de pruebas que implementa las capacidades especificadas en el alcance del producto.
2. **Documentación técnica:** Descripción de la arquitectura, tecnologías utilizadas, decisiones de diseño y guía de instalación/despliegue.
3. **Manual de usuario:** Guía de uso de la herramienta orientada a estudiantes y docentes.
4. **Informe de evaluación:** Documento con resultados de las pruebas de usabilidad.
5. **Código fuente:** Repositorio con el código completo del sistema, debidamente documentado y versionado.

4.2. *Objetivos*

4.2.1. *Objetivo General:* Desarrollar una herramienta interactiva que facilite la comprensión y aplicación de los contenidos de la asignatura *Fundamentos de Interpretación y Compilación de Lenguajes de Programación (LP)* de la Universidad del Valle, mediante la visualización de estructuras internas y el uso de lenguajes orientados por sintaxis definidos mediante gramáticas.

4.2.2. *Objetivos Específicos:*

1. Identificar las dificultades conceptuales más comunes que enfrentan los estudiantes en la asignatura, con el fin de orientar la selección y construcción de ejemplos.
2. Caracterizar los elementos teóricos y técnicos necesarios para la definición y análisis de gramáticas, así como para la generación de AST en el contexto de lenguajes orientados por sintaxis.
3. Diseñar la arquitectura del sistema y las interfaces de usuario, integrando estrategias de visualización que favorezcan la comprensión de estructuras sintácticas.
4. Implementar una herramienta interactiva que permita definir gramáticas y visualizar las estructuras sintácticas de programas, incorporando ejemplos y mecanismos de interacción accesibles.

5. Evaluar la funcionalidad técnica y la usabilidad de la herramienta mediante pruebas con estudiantes y el docente de la asignatura.

4.2.3. Restricciones y Supuestos:

4.2.3.1. Restricciones temporales: El proyecto debe completarse en un período total de ocho meses, lo cual limita el alcance de las funcionalidades a implementar y la profundidad de la evaluación. Este tiempo incluye todas las fases: análisis, diseño incluyendo la curva de aprendizaje de tecnologías, implementación y evaluación con estudiantes.

4.2.3.2. Restricciones de recursos humanos: El desarrollo estará a cargo de dos estudiantes de trabajo de grado, lo que restringe la complejidad y cantidad de características que pueden incorporarse. La participación de estudiantes en las pruebas de usabilidad depende del calendario académico y de su disponibilidad voluntaria durante el período de evaluación.

4.2.3.3. Restricciones tecnológicas: La herramienta debe ser accesible sin requerir instalación de software especializado, priorizando una implementación web funcional en equipos de uso común entre estudiantes universitarios. No se cuenta con infraestructura institucional garantizada para el despliegue, por lo que este se limitará a servicios gratuitos o de bajo costo. La selección tecnológica debe considerar la curva de aprendizaje del equipo y privilegiar herramientas con documentación amplia y comunidades activas.

4.2.3.4. Restricciones de alcance funcional: El sistema se enfocará exclusivamente en la interpretación de lenguajes orientados por sintaxis, excluyendo generación de código, compilación a ejecutables, construcción de código intermedio y optimización. Solo se soportará un subconjunto representativo de características de lenguajes de programación, sin pretender construir un entorno de desarrollo completo. La visualización se concentrará en la representación de AST y validación de gramáticas, excluyendo la visualización paso a paso del proceso de evaluación.

4.2.3.5. Restricciones de evaluación: No es posible medir el impacto en el aprendizaje a largo plazo debido a la necesidad de estudios experimentales prolongados, con grupos de control y seguimiento por varios semestres. La evaluación se restringirá a funcionalidad técnica y usabilidad, mediante cuestionarios o herramientas similares, y se prevé una muestra entre 10 y 20 estudiantes, suficiente para identificar problemas de interacción pero no para análisis estadísticos sobre efectividad pedagógica.

4.2.3.6. Supuestos del contexto educativo: Se asume que los estudiantes cuentan con conocimientos básicos de programación y estructuras de datos. También se supone la disposición de un grupo de estudiantes para participar voluntariamente en las pruebas, así como la colaboración del docente para facilitar acceso a la asignatura y permitir la presentación del prototipo.

4.2.3.7. Supuestos técnicos: Se asume la viabilidad técnica de implementar un parser para gramáticas libres de contexto y un generador de AST con rendimiento adecuado para uso interactivo. Se presupone que las tecnologías web modernas permitirán desarrollar visualizaciones dinámicas sin dependencias externas ni instalación adicional en el navegador.

4.2.3.8. Supuestos sobre recursos y disponibilidad: Se asume que será posible acceder a estudiantes de un período académico durante la fase de evaluación. También se presupone que el docente proporcionará material didáctico relevante para orientar el diseño de la herramienta. Asimismo, se considera que existirán

servicios de alojamiento gratuitos o de bajo costo suficientes para desplegar el prototipo durante la etapa de pruebas.

4.2.3.9. Supuestos sobre el problema y la evaluación: Se asume que las dificultades conceptuales identificadas en la literatura y en la consulta con el docente reflejan los obstáculos reales que enfrentan los estudiantes. También se presupone que la visualización de estructuras sintácticas (AST) aporta un valor pedagógico potencial, aun cuando este no se mida formalmente. Finalmente, se asume que los criterios de funcionalidad técnica y de usabilidad constituyen indicadores válidos y suficientes para evaluar el prototipo dentro del marco temporal del proyecto, aun cuando no sustituyen estudios de impacto en el aprendizaje.

REFERENCIAS

- [1] D. Kundra and A. Sureka, “An experience report on teaching compiler design concepts using case-based and project-based learning approaches,” in *2016 IEEE Eighth International Conference on Technology for Education (T4E)*, 2016, pp. 216–219.
- [2] S. R. Vegdahl, “Using visualization tools to teach compiler design,” in *Proceedings of the Fourteenth Annual Consortium on Small Colleges Southeastern Conference*, ser. CCSC ’00. Evansville, IN, USA: Consortium for Computing Sciences in Colleges, 2000, p. 72–83.
- [3] D. Baldwin, “A compiler for teaching about compilers,” in *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE ’03. New York, NY, USA: Association for Computing Machinery, 2003, p. 220–223. [Online]. Available: <https://doi.org/10.1145/611892.611974>
- [4] M. Mernik and V. Zumer, “An educational tool for teaching compiler construction,” *IEEE Transactions on Education*, vol. 46, no. 1, pp. 61–68, 2003.
- [5] W. Feurzeig, S. A. Papert, and B. Lawler, “Programming-languages as a conceptual framework for teaching mathematics,” *Interactive Learning Environments*, vol. 19, no. 5, pp. 487–501, 2011. [Online]. Available: <https://doi.org/10.1080/10494820903520040>
- [6] S. Stamenković and N. Jovanović, “A web-based educational system for teaching compilers,” *IEEE Transactions on Learning Technologies*, vol. 17, pp. 143–156, 2024.
- [7] L. Spigariol, J. Bono, and F. S. Guijarro, “Aprendiendo a desarrollar un intérprete de un lenguaje de programación funcional,” in *Actas del XXVIII Congreso Argentino de Ciencias de la Computación (CACIC)*, Universidad Tecnológica Nacional; Universidad Nacional de General San Martín; Universidad de Buenos Aires; Universidad Nacional de Hurlingham. La Rioja, Argentina: Red de Universidades con Carreras en Informática, Oct. 2023, pp. 166–175. [Online]. Available: <http://sedici.unlp.edu.ar/handle/10915/149494>
- [8] N. Chomsky, “Three models for the description of language,” *IRE Transactions on Information Theory*, vol. 2, no. 3, pp. 113–124, 1956.
- [9] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques, and Tools (2nd Edition)*. USA: Addison-Wesley Longman Publishing Co., Inc., 2006.
- [10] D. P. Friedman and M. Wand, *Essentials of Programming Languages*, 3rd Edition, 3rd ed. The MIT Press, 2008.
- [11] J. Piaget, *Genetic Epistemology*. New York Chichester, West Sussex: Columbia University Press, 1970. [Online]. Available: <https://doi.org/10.7312/piag91272>
- [12] L. Vygotsky, *Mind in Society: Development of Higher Psychological Processes*. Harvard University Press, 1978. [Online]. Available: <http://www.jstor.org/stable/j.ctvjf9vz4>
- [13] J. Sweller, “Cognitive load during problem solving: Effects on learning,” *Cognitive Science*, vol. 12, no. 2, pp. 257–285, 1988. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/0364021388900237>
- [14] J. S. Bruner, “The act of discovery.” *Harvard Educational Review*, vol. 31, pp. 21–32, 1961. [Online]. Available: <https://api.semanticscholar.org/CorpusID:142938071>
- [15] R. E. Mayer, *Multimedia Learning*, 2nd ed. Cambridge University Press, 2009.
- [16] J. Lu, M. Schmidt, M. Lee, and R. Huang, “Usability research in educational technology: a state-of-the-art systematic review,” *Educational Technology Research and Development*, vol. 70, no. 6, pp. 1951–1992, 2022. [Online]. Available: <https://doi.org/10.1007/s11423-022-10152-6>
- [17] N. Jovanović, S. Stamenković, and D. Miljković, “Comvis — interactive simulation environment for compiler learning,” *Computer Applications in Engineering Education*, vol. 30, no. 2, pp. 275–291, 2021. [Online]. Available: <https://onlinelibrary.wiley.com/doi/10.1002/cae.22456>

- [18] R. del Vado Vírseda, “Visualizing compiler design theory from implementation through an interactive tutoring tool: Experiences and results,” in *Proceedings of the 15th International Conference on Computer Supported Education (CSEDU 2023)*, 2023, pp. 333–340. [Online]. Available: <https://doi.org/10.5220/0011709800003470>
- [19] F. Ortín, J. L. Pérez *et al.*, “An empirical evaluation of lex/yacc and antlr parser generation tools,” *PLOS ONE*, vol. 17, no. 10, p. e0264326, 2022. [Online]. Available: <https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0264326>
- [20] S. Stamenković and N. Jovanović, “A web-based educational system for teaching compilers,” *IEEE Transactions on Learning Technologies*, 2024, early access / online. [Online]. Available: <https://doi.org/10.1109/TLT.2023.3297626>
- [21] M. Godbolt and contributors, “Compiler explorer (godbolt) — developer updates and educational uses,” <https://godbolt.org/>, 2022, web; herramienta interactiva para inspeccionar compilación y código intermedio. [Online]. Available: <https://godbolt.org/>
- [22] L. Daleman, “Reviewing the educational value of the chocopy compiler,” Ph.D. dissertation, LIACS / Leiden University (thesis), 2024. [Online]. Available: <https://theses.liacs.nl/pdf/2024-2025-DalemanLLuuk.pdf>
- [23] M. O. Myreen and contributors, “The cakeml project’s quest for ever stronger correctness (overview),” in *Proceedings of ITP / LIPIcs (ITP 2021)*, 2021, cakeML project — verified compiler explorer materials. [Online]. Available: <https://cakeml.org/publications.html>
- [24] J. Lu, M. Schmidt, M. Lee, and R. Huang, “Usability research in educational technology: A state-of-the-art systematic review,” *Educational Technology Research and Development*, vol. 70, no. 6, pp. 1951–1992, 2022. [Online]. Available: <https://doi.org/10.1007/s11423-022-10152-6>