Prototipo web para apoyar la enseñanza de Fundamentos de Interpretación y Compilación de Lenguajes de Programación mediante lenguajes orientados por sintaxis

Jota Emilio López Ramírez, Esmeralda Rivas Guzmán
Universidad del Valle
Facultad de Ingeniería
jota.lopez@correounivalle.edu.co,
esmeralda.rivas@correounivalle.edu.co

Resumen

El presente anteproyecto propone el diseño y desarrollo de un prototipo web interactivo orientado a facilitar la comprensión de los fundamentos de interpretación y compilación de lenguajes de programación. El prototipo permitirá a los estudiantes definir gramáticas y visualizar tanto las estructuras sintácticas resultantes (como árboles de sintaxis abstracta) como el ambiente y los cambios en el estado durante el proceso de evaluación conceptual de programas, sin incluir mecanismos de ejecución paso a paso ni funcionalidades de depuración.

El proyecto se enmarca en principios de usabilidad y en enfoques constructivistas del aprendizaje, y busca funcionar como un recurso complementario para la asignatura *Fundamentos de Interpretación y Compilación de Lenguajes de Programación* de la Universidad del Valle, apoyando la comprensión de conceptos clave mediante representaciones gráficas y recursos interactivos.

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 desarrollar un prototipo web que apoye la comprensión 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?"

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 se han desarrollado un conjunto de herramientas orientadas a reducir la abstracción inherente a la enseñanza de compiladores e intérpretes mediante visualizaciones, simulaciones paso a paso y entornos web interactivos, estos sistemas permiten manipular gramáticas, observar la construcción de árboles de sintaxis y seguir la ejecución o el proceso de traducción de manera controlada, facilitando la comprensión de conceptos como análisis léxico, parsing, construcción de AST y evaluación dirigida por sintaxis [17]–[19].

Entre las propuestas recientes destacan los entornos de simulación diseñados específicamente para cursos de compiladores, en particular, *ComVIS* integra simulaciones de autómatas, parsing LL/LR y visualización paso a paso del flujo de transformación, lo que ha demostrado utilidad pedagógica tanto en clases como en ejercicios guiados [17], [20]. Trabajos basados en tutores interactivos también muestran mejoras en la comprensión cuando se combinan implementaciones reales con visualizaciones que conectan teoría y práctica [18]. Asimismo, herramientas como *Compiler Explorer* aportan valor didáctico al permitir inspeccionar representaciones intermedias y código ensamblador, aunque están más orientadas a usuarios avanzados [21].

La literatura reciente incluye además estudios comparativos de generadores de analizadores, como las evaluaciones empíricas entre ANTLR y Lex/Yacc [19], así como análisis sobre el valor educativo de marcos modernos como ChocoPy o CakeML [22], [23]. Finalmente, revisiones sistemáticas sobre diseño y usabilidad en tecnologías educativas enfatizan la importancia de la retroalimentación inmediata, la reducción de carga cognitiva y la evaluación empírica de interfaces [16].

3.3. Antecedentes

En la asignatura *Fundamentos de Interpretación y Compilación de Lenguajes de Programación* de la Universidad del Valle se han utilizado históricamente herramientas como Racket y los materiales de *Essentials of Programming Languages* (EOPL), que permiten implementar intérpretes y explorar estructuras internas como árboles de sintaxis abstracta (AST), si bien estos enfoques han permitido vincular teoría y práctica, requieren familiaridad con entornos de programación específicos, lo que puede representar una barrera inicial para estudiantes sin experiencia previa. Este contexto local ha motivado la búsqueda de alternativas más accesibles, visuales e independientes de un entorno de desarrollo particular.

Desde una perspectiva más amplia, la enseñanza de compiladores ha estado tradicionalmente estructurada alrededor de proyectos de construcción de compiladores simplificados, lenguajes docentes y el uso de herramientas clásicas como Lex/Yacc. Estos enfoques se apoyan en fundamentos teóricos de larga trayectoria, como los modelos de descripción de lenguajes formales [8], los métodos clásicos de análisis sintáctico descritos en obras de referencia como el texto de Aho et al. [9] y los enfoques pedagógicos constructivistas que enfatizan la exploración activa, la carga cognitiva adecuada y el aprendizaje a partir de representaciones múltiples [11]–[15].

Autor / Año	Título	Aporte principal	Limitaciones / Vacíos
Jovanović et al. (2021)	*ComVIS: A Web-based Tool for Teaching Compiler Construction*	Herramienta web para visualizar parsing y AST con orientación didáctica.	No permite edición flexible de gramáticas; cubre solo ciertas fases del compilador; carece de un flujo completo integrado.
Lu et al. (2022)	*Usability of Educational Visualization Tools for Programming Languages*	Analiza heurísticas de usabilidad educativa para herramientas visuales; aporta criterios de diseño centrado en estudiantes.	No aborda compiladores directamente; no ofrece propuestas técnicas integradas; alcance conceptual limitado.
Del Vado et al. (2023)	*Interactive Teaching Tutors for Programming Language Foundations*	Tutores interactivos con visualización y apoyo pedagógico explícito.	Funcionalidad cerrada; poca flexibilidad técnica; no permite explorar gramáticas ni componentes avanzados.
Daleman et al. (2024)	*ChocoPy: A Pedagogical Python Subset for Compiler Courses*	Lenguaje educativo con herramientas para mostrar AST, análisis y ejecución.	No está diseñado como entorno guiado; limitado para principiantes; visualización no centrada en andamiaje pedagógico.
Ortin et al. (2022)	*Teaching Compiler Construction with ANTLR and Modern Tools*	Buenas prácticas para enseñar compiladores usando herramientas profesionales.	Curva de aprendizaje alta; herramientas muy complejas para novatos; escasa interactividad visual pedagógica.

Tabla 1
SÍNTESIS COMPARATIVA DE LAS PRINCIPALES FUENTES UTILIZADAS, INCLUYENDO SUS APORTES Y LIMITACIONES EN RELACIÓN CON LA ENSEÑANZA DE COMPILADORES.

3.3.0.1. 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 [16], [17], [22].

3.3.0.2. 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 [16]–[18].

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

Este proyecto comprende el diseño y desarrollo de un *prototipo web educativo* orientado a apoyar la comprensión de conceptos fundamentales de interpretación y compilación de lenguajes de programación. El alcance técnico incluye: (1) la definición y edición de *gramáticas con sintaxis configurable* por el usuario, (2) la visualización de *árboles de sintaxis abstracta (AST)* generados a partir de dichas gramáticas, y (3) la representación de los *cambios en el ambiente* durante la evaluación de programas escritos bajo la gramática definida.

4.1. Declaración del alcance

El prototipo se limita a funcionalidades demostrativas orientadas al aprendizaje, sin la intención de soportar un lenguaje completo de propósito general ni mecanismos avanzados de ejecución.

La investigación se desarrolla en el contexto de la asignatura *Fundamentos de Interpretación y Compilación de Lenguajes de Programación* del programa de Ingeniería de Sistemas de la Universidad del Valle, sede Tuluá, con una población de estudio compuesta por los estudiantes matriculados en la asignatura y el docente responsable, no obstante, debido al tiempo disponible para la realización del trabajo de grado, la evaluación del prototipo se llevará a cabo con un grupo reducido de entre 10 y 20 estudiantes, quienes participarán en pruebas de funcionalidad, usabilidad y retroalimentación cualitativa, lo que exige planear, ejecutar y analizar las pruebas en un período acotado.

El proyecto adopta un enfoque *experimental y evaluativo*, que comprende el análisis documental, el diseño e implementación del prototipo web y la realización de pruebas cualitativas centradas en la comprensión conceptual y la experiencia de uso. Por otro lado, quedan fuera del alcance aspectos como la depuración paso a paso, la compilación a código máquina, la autenticación o persistencia de usuarios, el despliegue en infraestructura productiva, la optimización de rendimiento y las evaluaciones pedagógicas de impacto a largo plazo.

4.1.1. Entregables del Proyecto:

1. **Prototipo web funcional:** Despliegue en un entorno de pruebas, con las capacidades definidas en el alcance.
2. **Documentación técnica:** Descripción de la arquitectura, tecnologías utilizadas, decisiones de diseño y guía de instalación/despliegue.
3. **Manual de usuario:** Guía de uso dirigida a estudiantes y docentes, que explique cómo utilizar las funcionalidades del prototipo.
4. **Código fuente:** Repositorio con el código completo, debidamente documentado y versionado.

4.2. Objetivos

4.2.1. Objetivo General: Desarrollar un prototipo web que apoye la comprensión 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.

4.2.2. Objetivos Específicos:

1. Identificar las dificultades conceptuales que enfrentan los estudiantes de la asignatura, con el propósito de orientar la selección de ejemplos, casos de uso y mecanismos de visualización.
2. Caracterizar los fundamentos teóricos y técnicos relacionados con las gramáticas formales, el análisis sintáctico y la generación de árboles de sintaxis abstracta, que servirán como base conceptual para el diseño del prototipo.
3. Diseñar la arquitectura web y las interfaces de usuario, integrando estrategias de visualización que favorezcan la comprensión de las estructuras sintácticas.
4. Implementar un prototipo funcional que permita definir gramáticas, visualizar los árboles de sintaxis abstracta y los cambios en el estado durante la evaluación de los programas escritos por el usuario.

5. Evaluar la usabilidad y funcionalidad del prototipo mediante pruebas con estudiantes inscritos en la asignatura y con el docente responsable.

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.

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: El prototipo debe funcionar en navegadores web modernos sin requerir instalaciones adicionales, garantizando compatibilidad con los dispositivos comúnmente utilizados por los estudiantes, asimismo, 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 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. Por lo tanto, la evaluación se restringirá a funcionalidad técnica y usabilidad, mediante cuestionarios o herramientas similares, y se prevé una muestra entre 10 y 20 estudiantes, para identificar problemas de interacción pero no para análisis estadísticos sobre efectividad pedagógica.

4.2.3.5. 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.

4.2.3.6. 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.

4.2.3.7. 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.

4.2.3.8. 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.

4.2.4. Resultados Esperados:

4.2.4.1. Resultados del Objetivo Específico 1: Se espera obtener una caracterización documentada de las dificultades conceptuales que enfrentan los estudiantes, mediante revisión de literatura, cuestionarios y análisis de material académico disponible, con el propósito de orientar la construcción de ejemplos pedagógicos, logrando identificar al menos cinco conceptos que servirán como base para la biblioteca de casos predefinidos que se incluirá en el prototipo.

4.2.4.2. Resultados del Objetivo Específico 2: Se definirá y documentará un conjunto priorizado de entre 8 y 12 requisitos funcionales del sistema y entre 5 y 8 requisitos no funcionales acompañados de criterios de aceptación específicos que permitan verificar su cumplimiento.

4.2.4.3. Resultados del Objetivo Específico 3: Se determinará la arquitectura del prototipo especificando los componentes principales, sus responsabilidades y el flujo de datos a través del sistema. Además, se producirán prototipos de interfaz de usuario y se definirán las estrategias de visualización seleccionadas, incluyendo bocetos o diagramas ilustrativos.

Igualmente, se realizará la selección de tecnologías a utilizar, justificando cada elección en términos de la curva de aprendizaje del equipo, disponibilidad de documentación, viabilidad de implementar las funcionalidades requeridas, y compatibilidad entre tecnologías seleccionadas.

4.2.4.4. Resultados del Objetivo Específico 4: Se obtendrá un prototipo funcional desplegado en un entorno de pruebas dando cumplimiento a los requisitos funcionales definidos en el *objetivo específico 2*.

El código fuente estará alojado en un repositorio GitHub con historial de commits, debidamente documentado y acompañado de archivo README con instrucciones de instalación y ejecución.

4.2.4.5. Resultados del Objetivo Específico 5: Se realizará una verificación básica del funcionamiento técnico y una valoración preliminar de su usabilidad, con un conjunto limitado de pruebas que permitan confirmar que el prototipo se ejecuta correctamente sus funciones principales.

Además, se obtendrá una percepción inicial de uso por parte del docente y de un grupo reducido de estudiantes, reflejada en un conjunto de observaciones sobre la facilidad para completar tareas básicas y los aspectos que resulten útiles o que requieran mejoras para desarrollos futuros.

4.2.5. Marco lógico de la propuesta:

Objetivos	**Indicadores Verificables**	**Medios de Verificación**	**Supuestos**
Fin Contribuir a la comprensión de conceptos fundamentales de interpretación y compilación mediante el uso de un prototipo web.	- Evidencia documental del uso pedagógico. - Interés del docente y estudiantes en utilizar el prototipo para apoyar la asignatura.	- Informe final del trabajo de grado. - Retroalimentación del docente responsable.	- El docente mantiene apertura al uso de recursos digitales.

Objetivos	**Indicadores Verificables**	**Medios de Verificación**	**Supuestos**
Propósito Disponer de un prototipo web educativo funcional que permita definir gramáticas, visualizar árboles de sintaxis abstracta y representar cambios de ambiente para apoyar el aprendizaje.	- Prototipo desplegado y operativo en entorno de pruebas. - Participación de entre 10 y 20 estudiantes en pruebas de usabilidad.	- Acceso al prototipo. - Cuestionarios o herramientas similares.	- Disposición del docente y estudiantes para participar en la evaluación.
Componentes			
Componente 1 Diagnóstico de dificultades conceptuales de los estudiantes.	- Documento con al menos cinco conceptos clave identificados.	- Informe de diagnóstico y revisión de literatura.	- Acceso a material académico y colaboración del docente.
Componente 2 Especificación de requisitos funcionales y no funcionales del sistema.	- Lista priorizada de 8–12 requisitos funcionales. - Lista de 5–8 requisitos no funcionales con criterios de aceptación.	- Documento de requisitos validado con el docente.	- Colaboración del docente en la revisión y validación.
Componente 3 Diseño de la arquitectura, prototipos de interfaz y selección tecnológica.	- Documento de arquitectura definido. - Prototipos de interfaz diseñados. - Justificación de las tecnologías seleccionadas.	- Diagramas, mockups y documentación técnica.	- Acceso a herramientas de diseño y documentación.
Componente 4 Prototipo web funcional y desplegado en entorno de pruebas.	- Prototipo con implementación de requisitos funcionales. - Código alojado en un repositorio documentado.	- Repositorio GitHub. - Acceso al entorno de pruebas.	- Capacidad técnica para implementación de funcionalidades.

Objetivos	**Indicadores Verificables**	**Medios de Verificación**	**Supuestos**
Componente 5 Evaluación de funcionalidad y usabilidad del prototipo.	- Cuestionarios aplicados a 10–20 estudiantes. - Observaciones de uso del prototipo.	- Registros de prueba y análisis cualitativo.	- Disposición de estudiantes para participar.
Actividades			
A1.1 Análisis de material académico y fuentes del curso.	- Revisión documental	- Informe de revisión	- Acceso a material académico
A1.2 Elaboración del diagnóstico de conceptos.	- Identificación de conceptos difíciles	- Selección de ejemplos y observaciones del docente	- Colaboración del docente y disponibilidad de información
A2.1 Identificación preliminar de requisitos.	- Identificación de requisitos funcionales y no funcionales	- Lista preliminar de requisitos	- Comprensión de las necesidades del prototipo
A2.2 Priorización y definición de criterios	- Requisitos priorizados y criterios establecidos	- Documento de criterios de aceptación	- Retroalimentación del docente disponible
A2.3 Validación de requisitos	- Requisitos validados con el docente	- Documento final de requisitos y criterios	- Colaboración del docente
A3.1 Diseño de arquitectura	- Arquitectura definida con componentes y flujo de información	- Documento de arquitectura	- Acceso a herramientas de diseño y prototipado
A3.2 Elaboración de prototipos de interfaz	- Mockups de interfaz creados	- Prototipos y bocetos	- Disponibilidad de software de diseño y revisión docente
A3.3 Selección y justificación de tecnologías	- Tecnologías seleccionadas	- Documento técnico con decisiones y justificación	- Viabilidad técnica y curva de aprendizaje del equipo
A4.1 Implementación del editor de gramáticas	- Editor con funciones básicas	- Repositorio con código y pruebas	- Estabilidad del entorno de desarrollo

Objetivos	Indicadores Verificables	Medios de Verificación	Supuestos
A4.2 Implementación del generador y visualizador de AST	- AST generados correctamente	- Repositorio y pruebas funcionales	- Capacidad técnica del equipo
A4.3 Implementación del visualizador de cambios de ambiente	- Visualización de cambios en tiempo de ejecución	- Repositorio y pruebas	- Entorno compatible y estable
A4.4 Integración y despliegue del prototipo	- Prototipo funcional en entorno de pruebas	- Repositorio y demostración funcional	- Acceso a servicios de alojamiento
A5.1 Diseño del protocolo de pruebas	- Protocolo definido con objetivos claros	- Documento de protocolo	- Disponibilidad de docentes y estudiantes para revisión
A5.2 Aplicación de pruebas	- Pruebas ejecutadas y resultados registrados	- Cuestionarios y observaciones	- Participación voluntaria de estudiantes y docente

Tabla 2: Marco lógico de la propuesta

5. METODOLOGÍAS

5.1. Metodología de investigación

La metodología de investigación adoptada en este proyecto se enmarca en un enfoque mixto, combinando revisión sistemática de literatura y desarrollo experimental de un prototipo educativo, con el fin de identificar dificultades conceptuales y validar la utilidad pedagógica de la herramienta. Para guiar el proceso, se utiliza el modelo de Niveles de Madurez Tecnológica (Technology Readiness Levels, TRL), desarrollado por la NASA en la década de 1970 como un sistema estandarizado para evaluar la madurez de tecnologías durante su evolución desde conceptos básicos hasta aplicaciones operativas [24].

A continuación, se presenta una tabla que resume los seis niveles de TRL, adaptados al desarrollo de software educativo para compiladores e intérpretes de lenguajes de programación:

TRL	Descripción General	Aplicación en este Proyecto
1	Principios básicos observados y reportados.	Identificación inicial de conceptos teóricos en gramáticas formales y análisis sintáctico mediante revisión bibliográfica.
2	Formulación de concepto tecnológico y/o aplicación.	Análisis de dificultades conceptuales en la asignatura.
3	Prueba de componentes analíticos y experimentales.	Caracterización de requisitos funcionales para el editor de gramáticas y generador de AST.
4	Validación en entorno de laboratorio.	Integración y pruebas iniciales de componentes en un entorno de desarrollo local.
5	Validación en entorno relevante.	Pruebas de prototipo con datos simulados de estudiantes.
6	Demostración de sistema prototipo en entorno relevante.	Evaluación con grupo reducido de estudiantes (10–20) en sesiones simuladas de clase, verificando usabilidad y funcionalidad pedagógica.

Tabla 3: Niveles de Madurez Tecnológica adaptados al proyecto.

En coherencia con el alcance definido para este trabajo de grado, el proyecto se orienta a alcanzar el TRL 6, correspondiente a un prototipo validado en un entorno relevante. Esto implica que el objetivo no es producir un sistema completamente operativo ni desplegarlo en un entorno institucional de uso permanente, sino implementar y evaluar un prototipo funcional que permita demostrar la viabilidad técnica del enfoque.

Los niveles superiores se excluyen, dado que requieren validación en escenarios reales de operación, estudios longitudinales y recursos que exceden el tiempo y alcance del proyecto académico.

5.1.1. Metodología PRISMA para la Revisión Sistemática: Para la fase inicial de revisión de literatura, se aplicará la guía PRISMA (Preferred Reporting Items for Systematic Reviews and Meta-Analyses) [25]. Este enfoque estructurado permitirá identificar, seleccionar y evaluar estudios relevantes sobre herramientas educativas para la enseñanza de compiladores e intérpretes y dificultades conceptuales comunes en estos temas. Los pasos clave incluyen:

1. Definición de preguntas de investigación y criterios de inclusión/exclusión.
2. Búsqueda sistemática en bases de datos académicas (IEEE Xplore, Web of Science, Scopus).
3. Selección de estudios mediante revisión de títulos, resúmenes y textos completos.
4. Extracción y síntesis de datos relevantes.

Esta metodología garantizará una base sólida de conocimiento para informar el diseño del prototipo y asegurar que se aborden las necesidades educativas identificadas en la literatura.

5.2. Metodología de desarrollo de software

Para el desarrollo del prototipo se adoptará Scrum como marco de trabajo ágil. Aunque Scrum no se considera una metodología rígida de desarrollo de software, su enfoque iterativo basado en ciclos cortos de planificación, ejecución y evaluación lo convierte en una opción adecuada para proyectos que requieren retroalimentación continua y una evolución progresiva del producto [26], [27].

La elección de Scrum se sustenta en un análisis previo de metodologías y marcos de trabajo, tomando como referencia criterios de selección establecidos en la literatura, tales como el nivel de formalidad, la flexibilidad, el tamaño del equipo y la naturaleza cambiante de los requisitos [28], [29]. Con base en estos criterios, Scrum se identificó como una alternativa pertinente para un proyecto académico con un equipo pequeño y un alcance sujeto a ajustes sucesivos.

El proceso de desarrollo se organizará mediante *Sprints* de duración fija, dentro de los cuales se definirán objetivos concretos y se desarrollarán incrementos funcionales del sistema. Scrum proporcionará la estructura para gestionar el avance mediante sus artefactos principales: el *Product Backlog*, que consolida los requisitos; el *Sprint Backlog*, que especifica los compromisos de cada iteración; y el *Incremento*, correspondiente al avance verificable generado al finalizar cada Sprint [26].

Dado el tamaño reducido del equipo, los integrantes asumirán combinadamente las responsabilidades de *Scrum Master* y *Development Team*, mientras que el director del trabajo de grado actuará como *Product Owner*, definiendo prioridades y validando los entregables. Diversos estudios han mostrado que Scrum puede adaptarse efectivamente a equipos pequeños y a entornos académicos o experimentales, lo cual respalda su utilización en este proyecto [30], [31].

5.3. Metodología de gestión de actividades

Para gestionar de forma operativa las actividades del proyecto se adoptará el enfoque Kanban, una metodología visual que permite controlar el trabajo en curso (Work In Progress, WIP), identificar cuellos de botella y maximizar la entrega continua [32]. Aunque el desarrollo del software seguirá el marco Scrum, la gestión diaria de tareas se realizará mediante Kanban como herramienta de seguimiento del flujo de trabajo, tanto para las tareas técnicas como de documentación. Esta combinación garantiza trazabilidad, flexibilidad y control del avance del proyecto.

El tablero contendrá como mínimo las siguientes columnas:

1. **Backlog:** tareas identificadas y priorizadas.
2. **To Do:** tareas comprometidas en el sprint actual.
3. **In Progress:** máximo 3-4 tarjetas simultáneas.
4. **Review / Testing:** código o funcionalidad lista para revisión o pruebas.
5. **Done:** completada y aceptada.

Con este enfoque se busca mantener un flujo de trabajo eficiente, evitar la sobrecarga y asegurar la calidad en cada etapa del desarrollo del prototipo web educativo.

5.4. Metodología de evaluación

La evaluación del prototipo combinará un enfoque formativo con el marco de usabilidad definido por la norma ISO 9241-11. Este marco considera la efectividad en el cumplimiento de objetivos, la eficiencia en

términos de esfuerzo y tiempo requeridos, y la satisfacción del usuario, entendida como sus percepciones y actitudes frente al sistema [33]. Este enfoque resulta adecuado para identificar problemas de interacción y para validar la claridad conceptual de las visualizaciones.

La evaluación será de carácter cualitativo predominante y se llevará a cabo en una única ronda al finalizar el ciclo de implementación, con la participación de entre 10 y 20 estudiantes de la asignatura, además del docente responsable, empleando las siguientes herramientas de recolección de datos:

1. **Lista de verificación de funcionalidad**: aplicada por los desarrolladores para verificar el cumplimiento técnico de los requisitos principales.
2. **System Usability Scale (SUS)**: cuestionario estandarizado de 10 ítems que proporciona una métrica cuantitativa comparable de usabilidad percibida [34].
3. **Cuestionario de utilidad pedagógica**: preguntas enfocadas en la claridad de las visualizaciones, la utilidad para comprender conceptos de compilación e interpretación, y sugerencias de mejora.

El análisis de los datos combinará el cálculo del puntaje SUS con una síntesis cualitativa de las observaciones recogidas y los comentarios del docente como usuario experto. A partir de esto, los resultados incluirán recomendaciones priorizadas de mejora. Aunque su implementación quede fuera del alcance temporal del proyecto, los resultados constituirán evidencia suficiente para respaldar el logro del TRL 6.

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] Ó. Salazar. ¿qué es la escala de madurez tecnológica (trl)? [Online]. Available: https://euro-funding.com/es/blog/que-es-la-escala-de-madurez-tecnologica-trl/

[25] M. J. Page, D. Moher, P. M. Bossuyt, I. Boutron, T. C. Hoffmann, C. D. Mulrow, L. Shamseer, J. M. Tetzlaff, E. A. Akl, S. E. Brennan, R. Chou, J. Glanville, J. M. Grimshaw, A. Hróbjartsson, M. M. Lalu, T. Li, E. W. Loder, E. Mayo-Wilson, S. McDonald, L. A. McGuinness, L. A. Stewart, J. Thomas, A. C. Tricco, V. A. Welch, P. Whiting, and J. E. McKenzie, "Prisma 2020 explanation and elaboration: updated guidance and exemplars for reporting systematic reviews," *BMJ*, vol. 372, 2021. [Online]. Available: https://www.bmj.com/content/372/bmj.n160

[26] K. Schwaber and J. Sutherland, "The scrum guide (2020) – spanish latin-south american edition," 2020. [Online]. Available: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-Spanish-Latin-South-American.pdf

[27] Atlassian, "Qué es scrum y cómo empezar," https://www.atlassian.com/es/agile/scrum, 2025.

[28] O. Tinoco Gómez, P. P. Rosales López, and J. Salas Bacalla, "Criterios de selección de metodologías de desarrollo de software," *Industrial Data*, vol. 13, no. 1, pp. 70–74, 2010.

[29] L. Flórez Marín and F. Grisales Tobón, "Formulación de criterios para la selección de metodologías de desarrollo de software," 2014.

[30] Z. Masood, R. Hoda, and K. Blincoe, "Real world scrum: A grounded theory of variations in practice," *ArXiv preprint arXiv:2103.15268*, 2021.

[31] D. A. Díaz Vargas *et al.*, "Implementing scrum to develop a connected robot," in *Proceedings of 2018*, 2018.

[32] H. Kniberg, *Kanban and Scrum - making the most of both*. Lulu.com, 2010.

[33] N. Bevan, J. Carter, and S. Harker, "Iso 9241-11 revised: What have we learnt about usability since 1998?" in *Human-Computer Interaction: Design and Evaluation*, M. Kurosu, Ed. Cham: Springer International Publishing, 2015, pp. 143–151.

[34] J. Brooke, "Sus: A quick and dirty usability scale," *Usability Eval. Ind.*, vol. 189, 11 1995.