Extensión de funcionalidades al entorno de desarrollo de MateFun

Informe de Proyecto de Grado presentado por

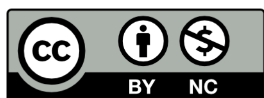
Facundo Barboza y Elizabeth Bennett

en cumplimiento parcial de los requerimientos para la graduación de la carrera de Ingeniería en Computación de Facultad de Ingeniería de la Universidad de la República

Supervisores

Sylvia da Rosa Zipitría
Federico Gómez Frois

Montevideo, 15 de abril de 2026



Extensión de funcionalidades al entorno de desarrollo de MateFun por Facundo Barboza y Elizabeth Bennett tiene licencia CC Atribución - No Comercial 4.0.

Resumen

Este informe expone parte del trabajo realizado en el contexto de la asignatura Proyecto de Grado de la carrera Ingeniería en Computación, centrado en la extensión de funcionalidades del entorno MateFun, un lenguaje de programación funcional diseñado con el propósito específico de introducir el modelado computacional de soluciones a problemas trabajados en la educación en ciencias. El objetivo del trabajo es mejorar la experiencia de uso del entorno incorporando funcionalidades inspiradas en entornos de desarrollo integrados modernos, manteniendo la simplicidad y el enfoque didáctico del entorno original del lenguaje.

A partir del análisis del estado del arte, se identificaron oportunidades de mejora en la visualización gráfica, la interacción con el código y la navegabilidad del entorno. Como resultado, se implementaron funcionalidades que permiten la coexistencia de gráficos y figuras en dos dimensiones, el uso de zoom general y por eje en gráficos 2D, la navegación hacia la definición de funciones incluidas desde otros archivos y la reorganización del menú del editor.

Las extensiones desarrolladas se integran sobre la base existente de MateFun, respetando su arquitectura y contribuyendo a su evolución como un recurso didáctico más flexible y alineada con prácticas actuales en entornos de desarrollo.

Palabras clave: entorno de desarrollo integrado, programación funcional, recursos didácticos, visualización gráfica, usabilidad.

Índice general

1. Introducción **1**

1.1. Objetivos planteados y logrados 2

2. Estado del arte **4**

2.1. Características de MateFun 5

2.2. Definición de IDE 8

2.3. Enfoque adoptado en la investigación de IDEs 9

2.3.1. Aspectos específicos considerados para la clasificación de IDEs 10

2.4. Presentación de IDEs analizados 11

2.5. Análisis de funcionalidades 13

2.5.1. Interfaz visual y experiencia de usuario (UX) 13

2.5.2. Evaluación y ejecución en tiempo real (REPL) 27

2.5.3. Visualización gráfica integrada 30

2.5.4. Soporte para autocompletado y ayuda contextual 33

2.5.5. Depuración visual y manejo de errores 35

2.5.6. Colaboración y trabajo en grupo 37

2.6. Propuestas de nuevas funcionalidades para MateFun 38

3. Análisis y Diseño de la solución **44**

3.1. Descripción general del sistema 44

3.2. Requerimientos del sistema 44

3.3. Casos de uso 45

3.4. Arquitectura del sistema 47

3.5. Diseño de la solución 48

4. Implementación y Pruebas **50**

4.1. Metodología de trabajo 50

4.2. Entorno de desarrollo 51
4.3. Implementación de casos de uso 52
4.3.1. CU001 - Coexistencia de gráficos y figuras 2D 52
4.3.2. CU002 - Aplicar zoom en gráficos 2D 53
4.3.3. CU003 - Navegar a la definición de una función 53
4.3.4. CU004 - Ejecutar acciones del editor mediante el menú 54
4.4. Pruebas realizadas 55
4.4.1. Estrategia de pruebas 56
4.4.2. Pruebas funcionales 56
4.5. Problemas encontrados durante el desarrollo 57
4.6. Dedicación 58

5. Conclusiones y Trabajo Futuro **60**
5.1. Conclusiones 60
5.2. Trabajo futuro 61

Referencias **62**

Anexos **63**
.1. Documento de requerimientos y casos de uso 63
.2. Documento de arquitectura 63

Capítulo 1

Introducción

Los entornos de desarrollo integrados (IDEs) constituyen una herramienta fundamental en el proceso de aprendizaje y desarrollo de software, ya que facilitan la escritura, comprensión y depuración de programas. En contextos educativos, la disponibilidad de entornos adecuados resulta especialmente relevante, dado que una interfaz compleja o poco intuitiva puede convertirse en una barrera adicional para estudiantes que se encuentran en etapas iniciales de formación.

El lenguaje MateFun cuenta con un entorno educativo desarrollado con el objetivo de ser adecuado para la educación en ciencias computacionales, ofreciendo una interfaz simple y orientada a la experimentación. El entorno permite definir funciones, trabajar con figuras bidimensionales y tridimensionales y visualizar resultados de forma gráfica, constituyendo un recurso valioso para introducir el modelado de soluciones computacionales en la educación en ciencias. No obstante, al igual que otros entornos educativos, su evolución plantea el desafío de incorporar nuevas funcionalidades sin perder la simplicidad y claridad que lo caracterizan.

En los últimos años, los IDEs modernos han incorporado una amplia variedad de herramientas orientadas a mejorar la productividad y la experiencia del usuario, tales como mecanismos avanzados de visualización, navegación entre definiciones y acceso simplificado a acciones del editor. La ausencia de algunas de estas características en entornos educativos puede limitar su uso a escenarios simples y dificultar el trabajo con programas de mayor complejidad.

En este contexto, el presente Proyecto de Grado se propone extender las funcionalidades del entorno MateFun, incorporando mejoras inspiradas en

prácticas habituales de IDEs modernos, pero adaptadas a un entorno educativo. El foco del trabajo se centra en la visualización gráfica, la interacción con el código y la navegabilidad del entorno, buscando mejorar la experiencia de uso sin comprometer el enfoque didáctico original.

El desarrollo del proyecto incluye el análisis del estado del arte, la definición de requerimientos funcionales y no funcionales, la especificación de casos de uso y la implementación de las funcionalidades propuestas sobre la base existente del sistema. De esta forma, se busca contribuir a la evolución de MateFun como un recurso didáctico flexible, fácilmente usable y alineado con las necesidades actuales de la enseñanza de las ciencias.

En este marco, cabe destacar que la mayoría de las sugerencias de mejora y de las nuevas funcionalidades consideradas en este proyecto surgen de la experiencia directa de los usuarios del entorno MateFun, fundamentalmente profesores de ciencias. Estas contribuciones, basadas en el uso del entorno en situaciones reales de enseñanza, permitieron identificar necesidades concretas y orientar las decisiones de diseño hacia soluciones alineadas con los objetivos didácticos del sistema.

1.1. Objetivos planteados y logrados

El objetivo principal de este proyecto es la ampliación y mejora de las funcionalidades del entorno MateFun, orientadas tanto a la experiencia de uso como a la potencia de sus herramientas. En particular, se busca:

- Optimizar la manipulación de figuras en dos dimensiones (2D), incorporando la posibilidad de realizar zoom independiente en los ejes X e Y, dado que la versión actual únicamente permite un acercamiento y alejamiento general sin discriminar por eje.

- Permitir la visualización de figuras y gráficas que coexisten en un mismo archivo.

- Incorporar mecanismos de navegabilidad entre funciones, de manera que el usuario pueda acceder directamente a la implementación de una función utilizada en un archivo, aunque su definición no se encuentre allí.

Asimismo, el proyecto contempla una investigación comparativa de distintas funcionalidades presentes en entornos de desarrollo integrados (IDEs,

Integrated Development Environments) conocidos, con el fin de identificar y proponer aquellas características que resulten más pertinentes y beneficiosas para incorporar al IDE de MateFun.

Capítulo 2

Estado del arte

En este capítulo se describe el análisis realizado sobre el estado de MateFun al momento de realizar este proyecto y sobre las distintas herramientas de desarrollo que sirvieron como referencia para orientar las mejoras propuestas en este proyecto. La etapa inicial estuvo centrada en comprender las funcionalidades disponibles en la versión heredada del sistema y las limitaciones que motivaron la definición de nuestros objetivos, como se explica en la sección 4.6 y se muestra en la figura 4.1. En la sección 2.1 se describe y analiza brevemente el IDE de MateFun.

A partir de este análisis, se llevó a cabo un relevamiento exhaustivo de entornos de desarrollo integrados (IDEs), con el fin de identificar funcionalidades relevantes que pudieran adaptarse a MateFun. Como guía para este relevamiento se consideraron aspectos como la interfaz visual, la ejecución en tiempo real, la visualización gráfica, el soporte para autocompletado, la depuración de errores y las capacidades de colaboración.

En el resto del capítulo se describe la metodología de investigación seguida para el relevamiento, los criterios de selección de IDEs, así como una descripción detallada de cada uno. El capítulo finaliza con el resultado del relevamiento, que consiste en un conjunto de propuestas que, sumadas a las que puedan aportar los usuarios, constituyen una base a tener en cuenta en el desarrollo del IDE de MateFun.

2.1. Características de MateFun

MateFun cuenta con un entorno de desarrollo integrado (IDE) diseñado específicamente para introducir la competencia de modelado computacional en la enseñanza de las ciencias. El IDE de MateFun, implementado en Angular, está pensado para facilitar la definición, ejecución y depuración de funciones en un lenguaje propio inspirado en Haskell. A continuación, se detallarán algunas de sus funcionalidades principales.

El editor de código está construido sobre la librería CodeMirror, una herramienta ampliamente utilizada para crear editores en aplicaciones web. Su implementación en MateFun ofrece funcionalidades de autocompletado mediante la combinación Ctrl + Espacio (Ver figura 2.1).

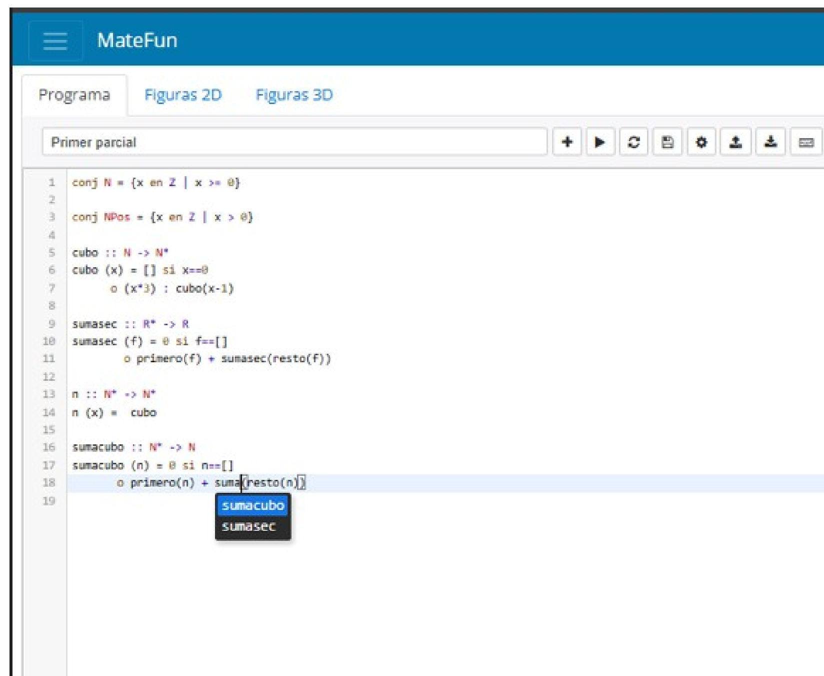


Figura 2.1: Funcionalidad de autocompletado en Matefun

Además, el usuario puede elegir entre una gran variedad de temas visuales, como dracula, monokai o eclipse, y configurar el tamaño de fuente del editor según sus preferencias. Como puede verse en las figuras 2.3 y 2.2, el editor permite que las opciones de configuración elegidas puedan ser guardadas o no, mediante la activación o desactivación del botón *guardar*.

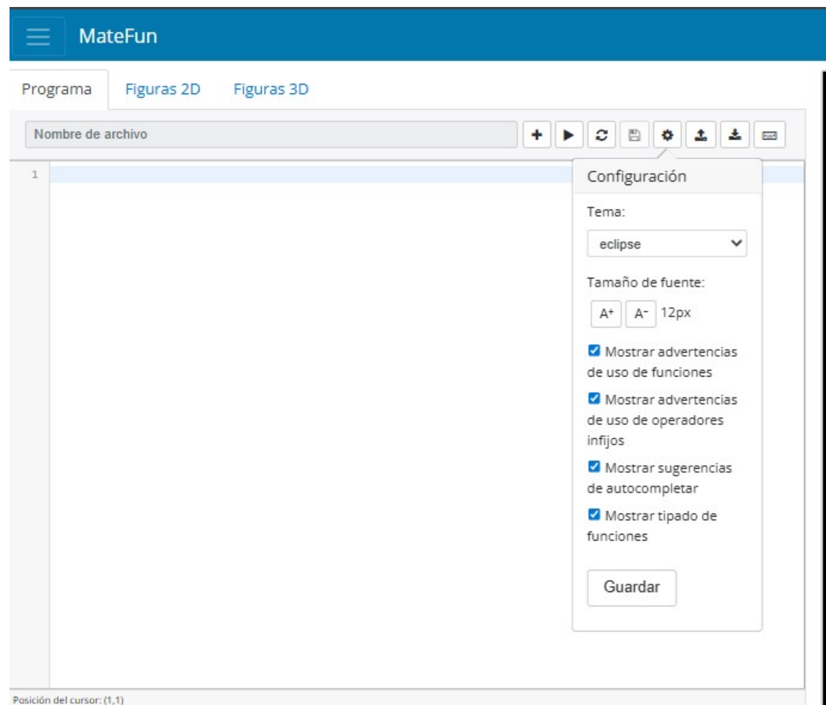


Figura 2.2: Cambiar tema - eclipse

Por otra parte, el editor cuenta con un mecanismo de detección automática de modificaciones, que mantiene una copia en memoria del contenido original del archivo. Esto permite identificar cambios no guardados y habilitar acciones como el guardado manual por parte del usuario.

La consola integrada está conectada a un intérprete funcional tipo GHCI, lo que permite ejecutar código sin salir del entorno. Esta consola puede mostrarse u ocultarse según la preferencia del usuario, y también admite el reinicio del intérprete (Ctrl + R) sin necesidad de recargar la página, facilitando así la interacción fluida con el sistema. El entorno también cuenta con una serie de atajos de teclado que permiten realizar operaciones comunes de forma rápida, como guardar (Ctrl + G), ejecutar (Ctrl + P), descargar (Ctrl + E) o seleccionar un directorio (Ctrl + A).

Entre otras funcionalidades destacadas se encuentra la posibilidad de previsualizar la signatura de funciones externas. Para ello, basta con posicionar el cursor sobre el nombre de la función y hacer clic sobre el mismo, lo que permite visualizar información relevante de su definición, como su nombre,

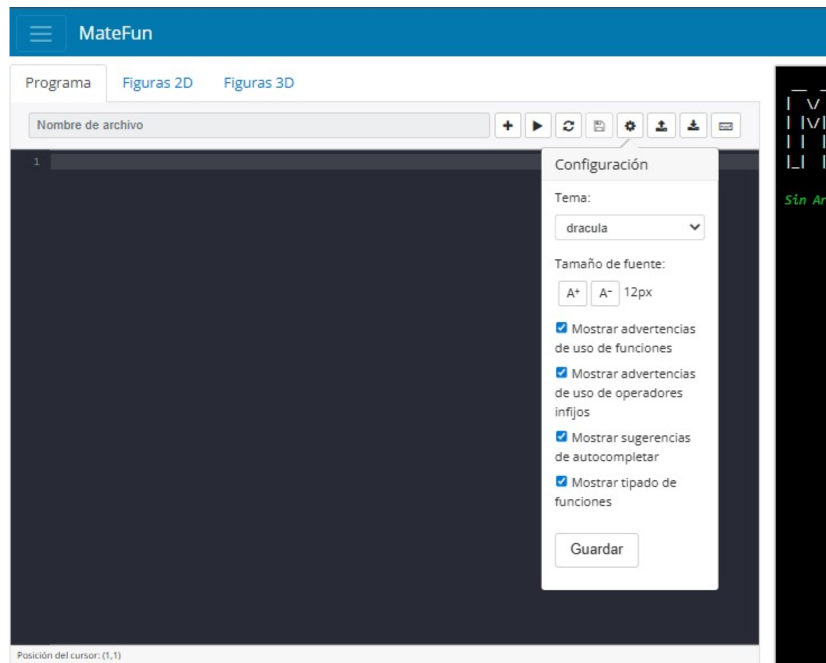


Figura 2.3: Cambiar tema - dracula

parámetros y tipos, sin necesidad de navegar al archivo correspondiente, ver figura 2.4. Además, se ofrecen herramientas para la visualización de gráficos en 2D y 3D mediante componentes dedicados, integrados como pestañas dentro del entorno.

A pesar de sus ventajas como recurso educativo, el sistema presenta actualmente algunas limitaciones que justifican la necesidad de mejoras. Por ejemplo, una de las restricciones identificadas es la imposibilidad de visualizar gráficas y figuras al mismo tiempo. Además, el comportamiento del zoom en las figuras 2D es diferente al de las figuras 3D, mientras que en estas últimas es posible hacer zoom de forma independiente en cada eje, en las figuras 2D el zoom afecta ambos ejes de forma simultánea. Este comportamiento ha sido señalado como dificultoso por algunos profesores de ciencias, por lo que se propone que las gráficas 2D adopten un comportamiento consistente con el de las figuras 3D.

Por otro lado, el entorno de desarrollo integrado (IDE) que proporciona MateFun presenta una funcionalidad limitada. Las herramientas disponibles resultan básicas, lo cual puede restringir la fluidez del trabajo y la explora-

Figura 2.4: Funcionalidad de ver definicion de funcion

ción por parte del usuario. Estas observaciones, junto con los comentarios recabados de usuarios del entorno, en particular de profesores de ciencias, y el análisis de distintos IDEs realizado en este capítulo, fundamentan el presente trabajo. A partir de ello, se proponen y desarrollan mejoras orientadas a optimizar la usabilidad y el funcionamiento del sistema.

2.2. Definición de IDE

Según la definición del libro de Sommerville, I. (2016). Software Engineering (9th ed.), "Un IDE es un conjunto de herramientas de software que apoyan diferentes aspectos del desarrollo de software, dentro de cierto marco común e interfaz de usuario. Por lo común, los IDE se crean para apoyar el desarrollo en un lenguaje de programación específico, como Java. El lenguaje IDE puede elaborarse especialmente, o ser una ejemplificación de un IDE de propósito general, con herramientas de apoyo a lenguaje específico."

La importancia de un IDE reside en que no solamente permite escribir código, sino que proporciona al desarrollador herramientas adicionales que optimizan su flujo de trabajo y mejoran significativamente la eficiencia, productividad y calidad del software resultante. En contextos educativos, la relevancia de los IDEs es aún mayor, ya que una buena elección en cuanto a la herramienta y sus funcionalidades puede contribuir notablemente al proceso de enseñanza-aprendizaje de conceptos técnicos, haciendo la experiencia

educativa más interactiva y atractiva para los estudiantes.

Dado el contexto de este proyecto y el público objetivo, resulta importante realizar previamente una investigación profunda acerca de IDEs existentes, especialmente aquellos que han sido diseñados o adaptados para su uso educativo o que poseen características afines a los objetivos didácticos planteados.

2.3. Enfoque adoptado en la investigación de IDEs

Para llevar adelante una investigación organizada y significativa, se definió una estrategia que permitiera evaluar diversos IDEs según criterios claramente establecidos. La investigación no se centró exclusivamente en identificar IDEs populares o ampliamente utilizados, sino que buscó analizar aquellas funcionalidades específicas que podrían adaptarse y resultar útiles en el contexto particular de MateFun.

Dicho análisis se realizó en tres etapas claramente diferenciadas:

- **Exploración inicial:** En esta fase se revisaron IDEs ampliamente conocidos, utilizados tanto a nivel profesional como educativo. El objetivo fue identificar un panorama general sobre las características básicas y avanzadas que típicamente ofrecen estas herramientas.

- **Categorización y selección específica:** Se realizó una selección enfocada en IDEs que tuvieran algún tipo de relevancia educativa, visual o interactiva, debido al público objetivo (estudiantes y docentes). Se consideraron especialmente herramientas orientadas a lenguajes de programación funcional o matemática, así como aquellas que destacaran en aspectos visuales y en simplicidad de uso.

- **Análisis detallado por funcionalidad:** Finalmente, se identificaron las características específicas de los IDEs seleccionados, prestando especial atención a las funcionalidades que se consideraron deseables y relevantes para mejorar la experiencia de aprendizaje dentro de MateFun.

2.3.1. Aspectos específicos considerados para la clasificación de IDEs

Para la evaluación detallada y posterior clasificación de cada IDE, se definieron dimensiones concretas que permitieron sistematizar el análisis realizado. A continuación se mencionan brevemente estas dimensiones, explicando la importancia y el criterio utilizado para evaluar cada una:

- **Interfaz visual y experiencia de usuario (UX):** Se consideró fundamental evaluar la simplicidad y claridad visual del entorno, ya que los usuarios finales suelen no tener experiencia previa en programación. El diseño minimalista, el uso apropiado de colores y la organización visual intuitiva fueron criterios clave.

- **Evaluación y ejecución en tiempo real (REPL):** Se evaluó la posibilidad de que el usuario reciba *feedback* inmediato al escribir una expresión o función matemática. Esta característica resulta particularmente importante para facilitar el aprendizaje exploratorio y para reducir la frustración en los estudiantes principiantes.

- **Visualización gráfica integrada:** Se consideró crucial que el IDE de MateFun disponga de herramientas gráficas que permitan visualizar funciones de manera clara e interactiva (por ejemplo, gráficas 2D y 3D), debido a la importancia de dichas representaciones en la enseñanza de las ciencias.

- **Soporte para autocompletado y ayuda contextual:** Se valoró especialmente la presencia de autocompletado inteligente y sugerencias contextuales sobre errores comunes, ya que estas herramientas pueden acelerar el aprendizaje y reducir errores frecuentes durante el desarrollo.

- **Depuración visual y manejo de errores:** Se buscó evaluar el soporte visual para depuración (*debugging*) y la presentación de errores de manera clara y amigable para usuarios con poco o ningún conocimiento técnico previo.

- **Colaboración y trabajo en grupo:** Finalmente, se analizó la capacidad de los IDEs para permitir la colaboración simultánea entre usuarios (por ejemplo, alumnos y docentes), lo que podría facilitar el trabajo en grupo y la supervisión educativa.

En base a estos criterios se estructuró una investigación rigurosa, orientada a seleccionar y adaptar las mejores prácticas de IDEs existentes para considerar la posibilidad de ser implementadas en MateFun, garantizando así una experiencia educativa eficaz y atractiva para sus usuarios finales.

2.4. Presentación de IDEs analizados

A partir de la metodología y el enfoque descritos previamente, se seleccionaron diversos IDEs para ser analizados. Antes de abordar en profundidad las funcionalidades específicas de cada entorno, resulta pertinente introducir brevemente cada una de estas herramientas, con el fin de contextualizar y comprender mejor su relevancia dentro del análisis posterior.

A continuación, se presentan brevemente estos entornos analizados, ofreciendo una visión general de cada uno:

Visual Studio Code

Se trata de un entorno altamente personalizable y extensible, ampliamente utilizado por desarrolladores profesionales debido a su soporte para una gran variedad de lenguajes y frameworks. Entre sus principales características se destacan el autocompletado inteligente mediante IntelliSense, la gestión visual de errores y la posibilidad de integrar numerosas extensiones.

Visual Studio

Es una plataforma de lanzamiento creativa que se puede utilizar para editar, depurar y compilar código y, finalmente, publicar una aplicación. Además del editor y depurador estándar que ofrecen la mayoría de los IDEs, Visual Studio incluye compiladores, herramientas de completado de código, diseñadores gráficos y muchas más funciones para mejorar el proceso de desarrollo de software.

Eclipse

Es un entorno flexible y abierto, utilizado en contextos tanto profesionales como educativos. Se destaca por ofrecer una amplia gama de plugins y herramientas complementarias que facilitan tareas como la depuración, gestión

de proyectos complejos y soporte multilenguaje, lo que permite adaptarlo fácilmente a diversas necesidades de desarrollo.

BlueJ

Se presenta como un entorno diseñado específicamente con fines educativos, centrado principalmente en la programación orientada a objetos utilizando Java. Su valor educativo radica en su interfaz visual simple y directa, la facilidad para interactuar con objetos del programa y la posibilidad de ejecutar código directamente, lo cual resulta especialmente adecuado para estudiantes que inician su aprendizaje en programación.

DrRacket

Se enfoca particularmente en la enseñanza de la programación funcional utilizando el lenguaje Racket. Este entorno ofrece una interfaz clara y sencilla que integra un editor de código con una consola interactiva, y brinda herramientas visuales que permiten evaluar y depurar el código de manera inmediata, favoreciendo el aprendizaje exploratorio.

Replit

Es una plataforma de desarrollo integrada (IDE) completamente web, diseñada para facilitar la creación, ejecución y colaboración en proyectos de programación de manera accesible y sin necesidad de instalaciones locales. Además, incorpora herramientas de inteligencia artificial diseñadas para asistir al usuario durante toda la experiencia de desarrollo.

Thonny

Corresponde a un entorno pedagógico diseñado específicamente para principiantes en la programación con Python. Sus fortalezas residen en la simplicidad de uso y sus herramientas pedagógicas integradas, incluyendo la depuración visual paso a paso, visualización clara del estado de las variables y evaluación gráfica detallada de expresiones, facilitando así una comprensión profunda de los conceptos de programación.

Cursor

Se caracteriza por ser un entorno asistido mediante inteligencia artificial, que busca mejorar significativamente la productividad en el desarrollo de software. Basado en Visual Studio Code, ofrece funcionalidades avanzadas como la generación automática de código, asistentes conversacionales que permiten interactuar mediante lenguaje natural, y potentes herramientas de autocompletado contextual, adecuadas tanto para usuarios principiantes como avanzados.

Con esta presentación general de los IDEs seleccionados, se establece una base sólida que facilitará el análisis posterior de sus funcionalidades específicas, permitiendo una comparación clara y efectiva orientada a la selección de mejoras concretas para MateFun.

2.5. Análisis de funcionalidades

Una vez contextualizados los entornos de desarrollo seleccionados y presentadas las categorías de análisis utilizadas en esta investigación, se procederá a detallar de manera sistemática las funcionalidades observadas en cada IDE.

El objetivo de esta sección es documentar con precisión cuáles características concretas ofrece cada entorno, organizadas según las dimensiones previamente establecidas: interfaz visual y experiencia de usuario, ejecución en tiempo real, visualización gráfica integrada, soporte para autocompletado, disponibilidad de ejemplos, depuración visual y capacidades de colaboración.

La información que se presenta a continuación fue extraída directamente de la documentación oficial de cada herramienta y complementada con pruebas prácticas realizadas durante la investigación. Esta sistematización permite identificar con claridad las funcionalidades que podrían ser consideradas para su implementación o adaptación en el entorno MateFun, las cuales se detallan en la sección 2.6

2.5.1. Interfaz visual y experiencia de usuario (UX)

Uno de los aspectos más determinantes en la adopción y efectividad de un entorno de desarrollo, especialmente en contextos educativos, es su interfaz visual y la experiencia general que ofrece al usuario. Esta categoría evalúa la

claridad, simplicidad, organización visual y personalización del entorno, así como la forma en que facilita la navegación y comprensión del código.

Organización del código y navegación

Una de las funcionalidades más valoradas en esta categoría es la posibilidad de contraer y expandir bloques de código, presente en entornos como **Visual Studio**. Esta herramienta permite ocultar temporalmente secciones completas del código fuente, dejando visible únicamente la cabecera del bloque —por ejemplo, la firma de un método—. Esta característica no solo mejora la legibilidad, sino que facilita la navegación en archivos extensos, permitiendo al usuario enfocarse en las secciones relevantes sin perder de vista la estructura general del programa.

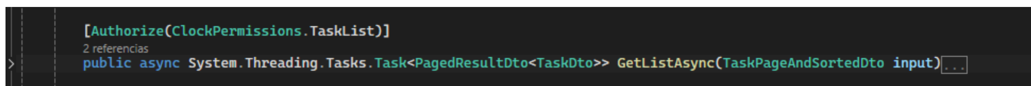


Figura 2.5: Bloque contraído en Visual Studio

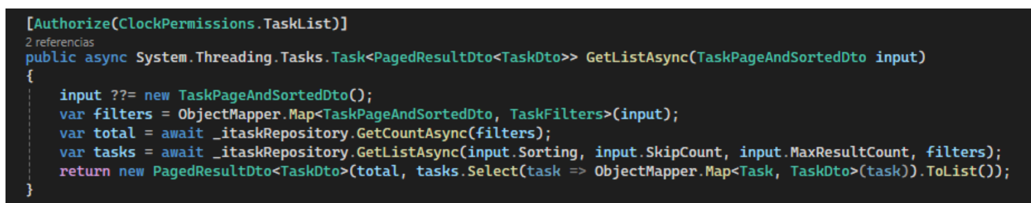


Figura 2.6: Bloque expandido en Visual Studio

Complementariamente, el mismo entorno permite al usuario acceder a la definición de símbolos como tipos, métodos o variables, sin abandonar el contexto actual del código. Esta característica permite al desarrollador inspeccionar la implementación de un símbolo directamente desde su uso, facilitando la comprensión y el análisis del código.

Los IDEs suelen ofrecer dos formas principales de acceder a esta funcionalidad: mediante la opción de 'Ir a definición', que redirige al archivo fuente correspondiente, o a través de una vista emergente de definición en línea. Esta

última alternativa, está disponible mediante atajos de teclado, por ejemplo Alt + F12, permite abrir una ventana superpuesta que muestra la definición del símbolo sin interrumpir el flujo de trabajo en el archivo actual.
La ventana emergente permite desplazarse por el contenido e incluso navegar por definiciones adicionales desde el mismo entorno. Una vez finalizada la inspección, puede cerrarse fácilmente, manteniendo la productividad y la continuidad del desarrollo. Opción seleccionando Alt + F12: permite ver la definición del método en la misma pantalla. Ver figura 2.7

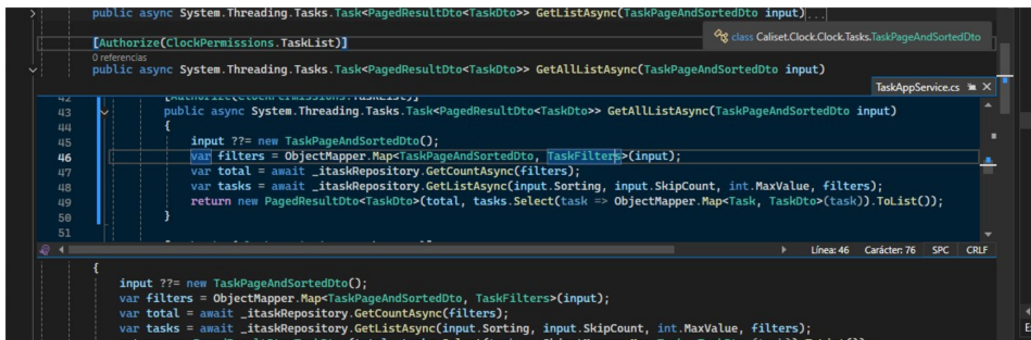


Figura 2.7: visualización de definiciones de símbolos

Diseños adaptados a entornos educativos

En contraposición al enfoque altamente configurable de entornos como VS Code o Eclipse, IDEs como **BlueJ** o **Thonny** han sido diseñados específicamente para facilitar el aprendizaje, especialmente entre estudiantes sin experiencia previa.

BlueJ, por ejemplo, ofrece una interfaz reducida y centrada en los conceptos de programación orientada a objetos. Su diseño intencionalmente simplificado facilita que los estudiantes principiantes se concentren en los fundamentos sin distracciones innecesarias (ver figura 2.8).
Una de sus principales innovaciones es el *object bench*, que permite crear e inspeccionar objetos de forma interactiva, visualizando sus atributos y estados internos (ver figura 2.10). Además, la funcionalidad *scope colouring* facilita la lectura del código mediante el resaltado visual de bloques, lo cual ayuda a detectar errores estructurales como llaves mal colocadas. Ver figura 2.9.

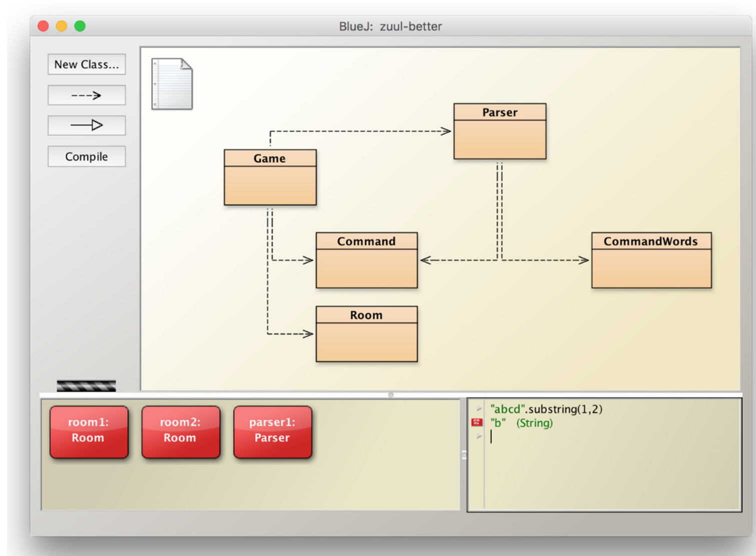


Figura 2.8: Interfaz BlueJ.

```
import java.awt.*;

/**
 * A square that can be manipulated and th
 *
 * @author  Michael Kolling and David J. B
 * @version 2008.03.30
 */

public class Square
{
    private int size;
    private int xPosition;
    private int yPosition;
    private String color;
    private boolean isVisible;

    /**
     * Create a new square at default posi
     */
    public Square()
    {
        size = 30;
        xPosition = 60;
        yPosition = 50;
        color = "red";
        isVisible = false;
    }
```

Figura 2.9: Resaltado de bloques de código en BlueJ

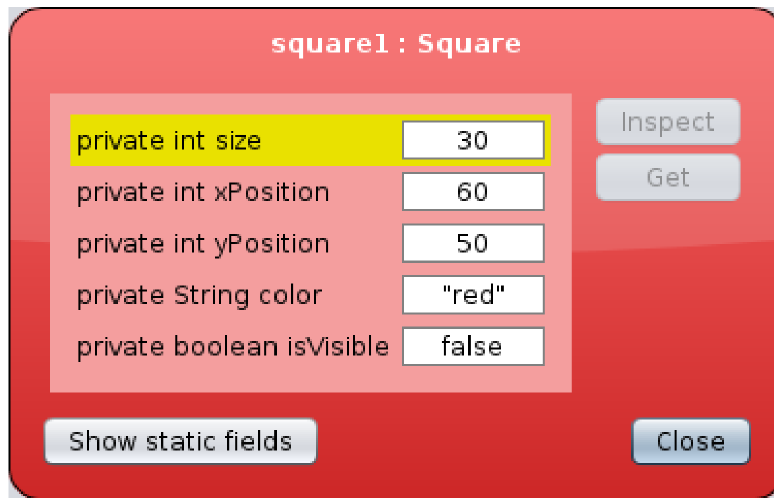


Figura 2.10: Detalles de objeto (object bench) en BlueJ.

Una funcionalidad destacada es la posibilidad de ejecutar métodos y fragmentos de código directamente desde el entorno, sin necesidad de definir un método `main` ni compilar el programa completo. Esto hace que BlueJ sea una especie de intérprete visual para Java, permitiendo al usuario experimentar en forma directa con el comportamiento de sus clases y objetos.Ver figura 2.11

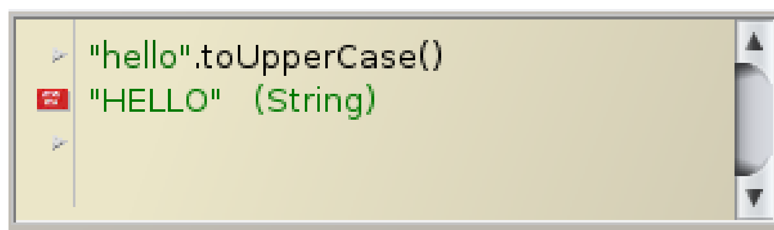


Figura 2.11: Code Pad en BlueJ.

Además, el entorno incorpora un *code pad*, donde el estudiante puede probar expresiones rápidamente, y ofrece mensajes visuales claros para guiar la comprensión del flujo de ejecución. Estas herramientas convierten a BlueJ en un entorno especialmente potente para enseñar los principios de la programación orientada a objetos desde una perspectiva visual y exploratoria.

Por su parte, **Thonny**, se destaca por su fuerte enfoque pedagógico y accesibilidad desde el primer uso. El entorno incluye una instalación simplificada que incorpora Python de forma predeterminada, eliminando barreras técnicas para usuarios principiantes. Su interfaz oculta funcionalidades avanzadas hasta que son necesarias, reduciendo la sobrecarga cognitiva. Es por eso que inicialmente se la ve deliberadamente simplificada (figura 2.12).

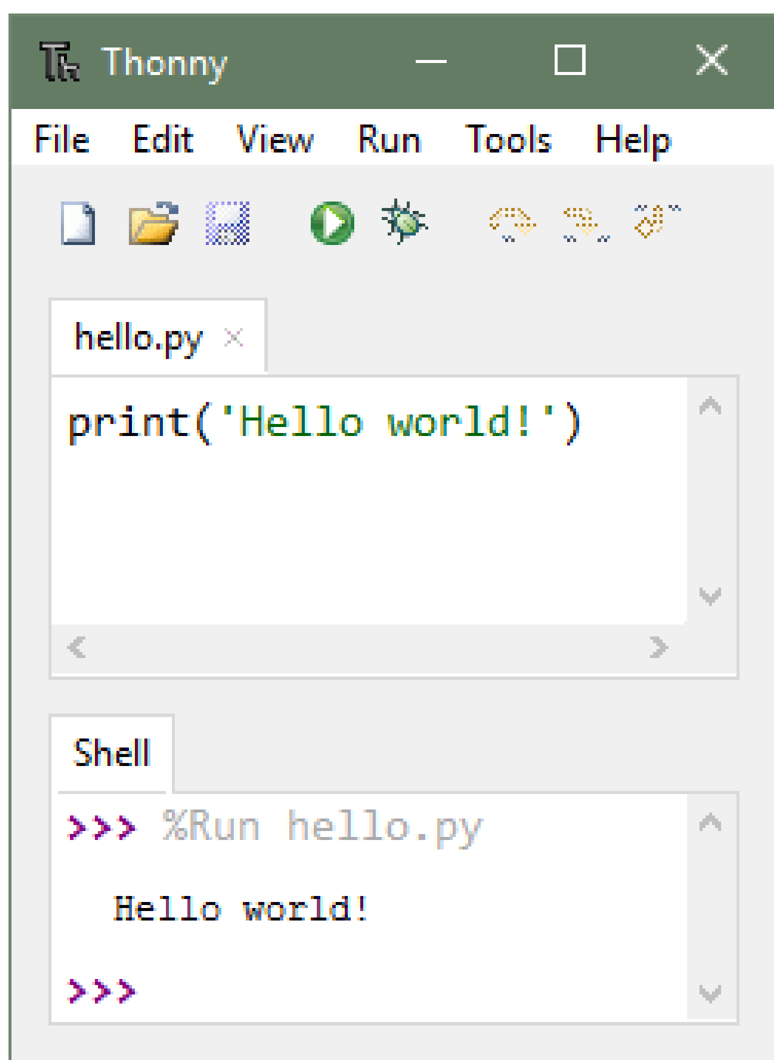


Figura 2.12: Interfaz inicial de ejemplo en Thonny.

Una de sus funcionalidades más valoradas es la **visualización del esta-**

do de las variables, que puede activarse desde el menú *Ver* → *Variables*. Esta vista permite al usuario observar en tiempo real cómo las instrucciones afectan a las variables en memoria, promoviendo una comprensión profunda del flujo del programa (figura 2.13).

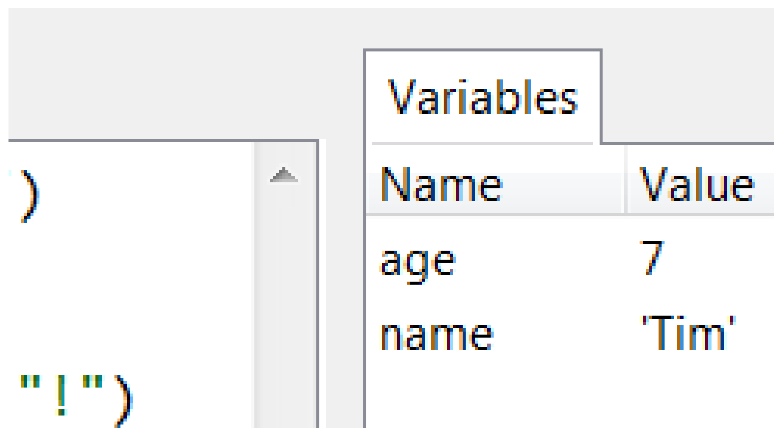


Figura 2.13: Visualización de variables en Thonny.

Además, este entorno en particular también implementa una representación visual diferenciada para ámbitos de variables (scopes), destacando con colores distintos las variables locales y globales, y permitiendo así detectar ambigüedades semánticas o errores lógicos. Esto se puede ver como ejemplo en la figura 2.14.

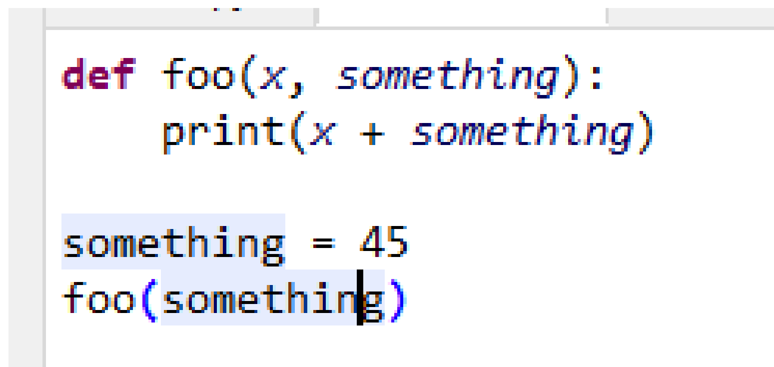


Figura 2.14: Visualización de alcance variables en Thonny.

La depuración visual paso a paso también está integrada de forma

natural. Al ejecutar un programa con `Ctrl+F5`, el entorno permite recorrer el código sin necesidad de breakpoints, utilizando pasos grandes (`F6`) o pequeños (`F7`). Este último modo revela el proceso de evaluación de expresiones subcomponente por subcomponente, simulando el razonamiento manual del estudiante. Esto se ve en la siguiente figura 2.15:

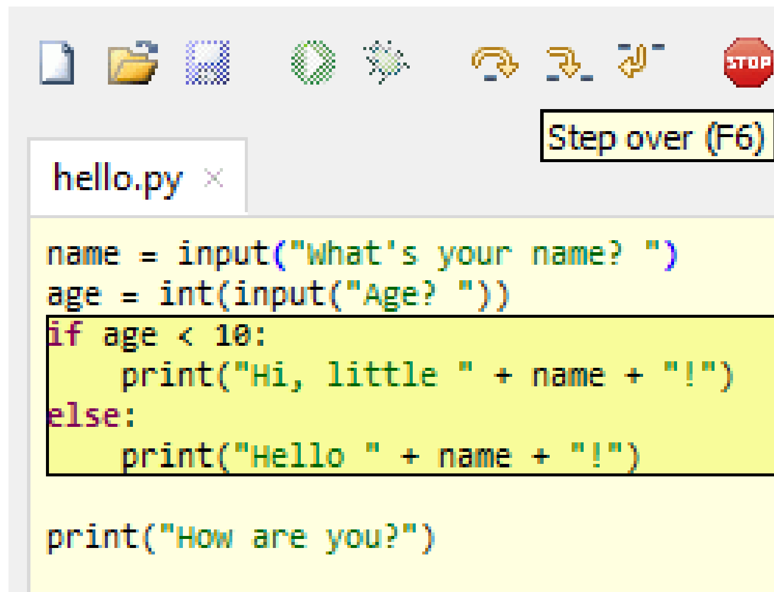


Figura 2.15: Depuración visual en Thonny.

Por su parte, **la segmentación de llamadas a funciones** se presenta de forma visual clara ya al ingresar en una función, se abre una nueva ventana con las variables locales y el estado actual del puntero del programa. Esta vista resulta particularmente útil para entender procesos recursivos y la propagación de parámetros como se puede ver a continuación (figura 2.16) :

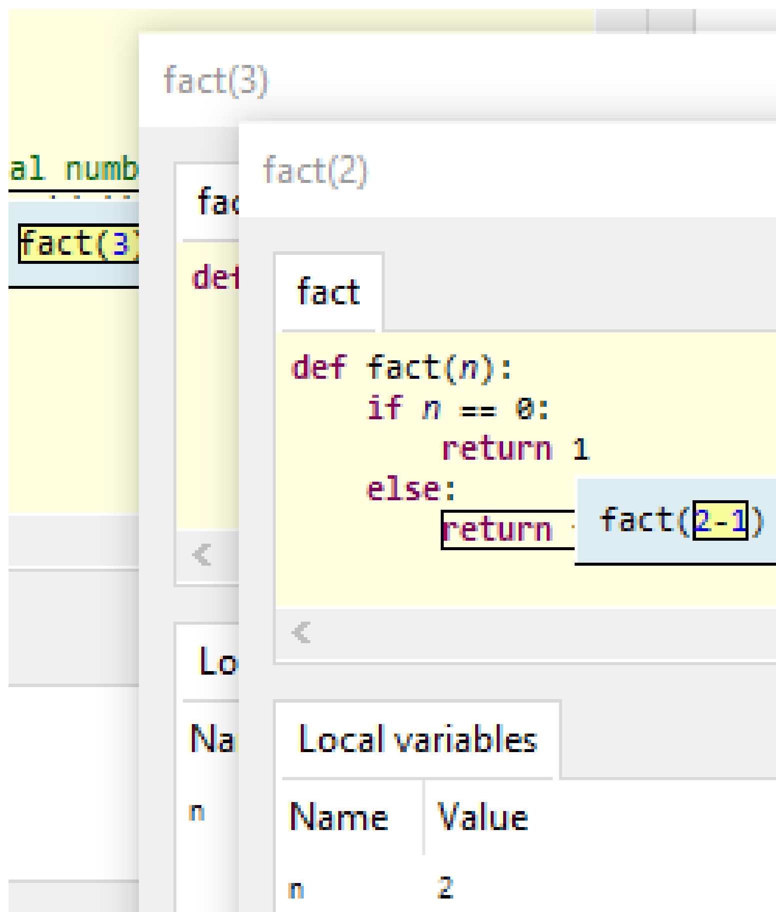


Figura 2.16: Visualización de alcance variables en Thonny.

Finalmente, Thonny detecta y resalta de forma automática errores sintácticos comunes, como comillas sin cerrar o paréntesis desbalanceados, y proporciona mensajes inmediatos y accesibles. Todo esto convierte a Thonny en una herramienta potente y clara para entornos educativos.Esto se puede ver ilustrado como ejemplo en la figura 2.17.

Figura 2.17: Errores de sintáxis en Thonny.

Navegación estructurada y personalización

En cuanto a la organización del trabajo dentro del entorno, **DrRacket** y **Replit** también aportan ideas relevantes.

DrRacket estructura su interfaz en una división clara entre el área de definición de código (Definitions Window) y el área de interacción (Interactions Window), lo que permite una separación visual inmediata entre el código fuente y su ejecución (figura 2.18). Además, permite abrir múltiples pestañas de trabajo (Tabbed Editing), facilitando la gestión de archivos o ejercicios simultáneos. Con esto lo que se logra es poder editar múltiples archivos dentro de la misma ventana, cada uno con su propia pestaña e interacción asociada (figura 2.19).

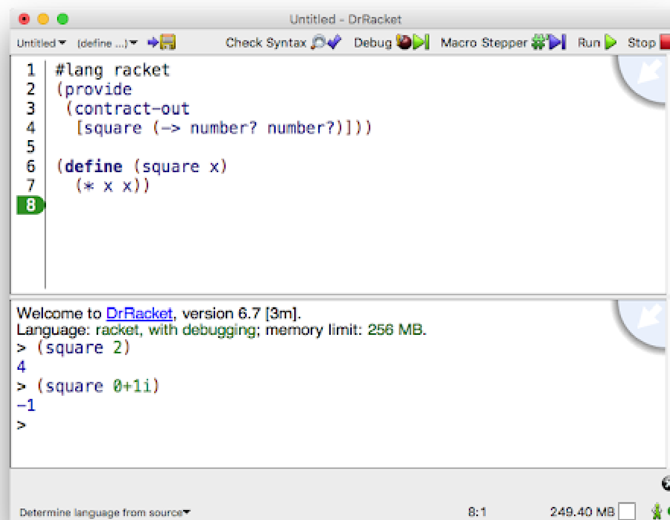


Figura 2.18: Interfaz visual DrRacket.

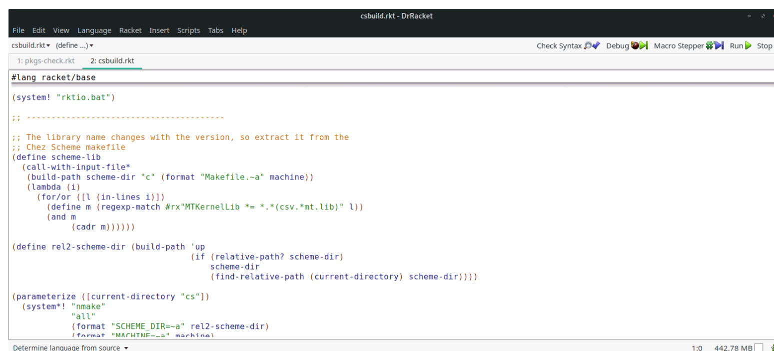


Figura 2.19: Pestañas en DrRacket.

Replit, por su parte, ofrece una interfaz moderna (figura 2.20) basada en el navegador, con soporte para pestañas y paneles configurables, integración con consola y vista previa de resultados en tiempo real. Además, permite a los usuarios alternar entre modos claro y oscuro según su preferencia visual, mejorando la accesibilidad y la comodidad en sesiones prolongadas.

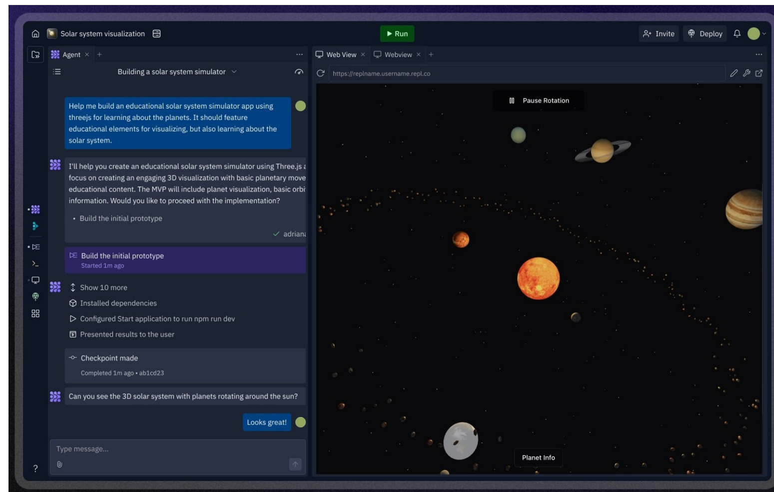


Figura 2.20: Interfaz de Replit.

Extensibilidad y personalización avanzada

Finalmente, cabe mencionar entornos como **Eclipse** y **Cursor**, que se destacan por sus capacidades de personalización y extensibilidad, ofreciendo enfoques distintos pero complementarios respecto a la experiencia de usuario.

Por un lado, *Eclipse* es un entorno pensado para proyectos de gran escala y estructuras complejas. Su interfaz se organiza mediante el concepto de perspectivas (Perspectives), las cuales son configuraciones visuales predefinidas adaptadas a distintos flujos de trabajo, como desarrollo, depuración, diseño gráfico o modelado (figura 2.21). Esto permite al usuario alternar rápidamente entre vistas personalizadas según la tarea a realizar, mostrando u ocultando paneles según necesidad.

Además, la vista de proyecto (Package Explorer) ofrece una representación jerárquica detallada de la estructura del proyecto, facilitando la navegación eficiente en proyectos que involucren múltiples módulos, bibliotecas externas y recursos asociados. Los paneles de consola, problemas y tareas permiten monitorear en tiempo real la salida de compilación, mensajes de error y actividades pendientes, configurables a gusto del usuario.

Sin embargo, esto también conlleva una curva de aprendizaje más pronunciada para usuarios principiantes, especialmente.

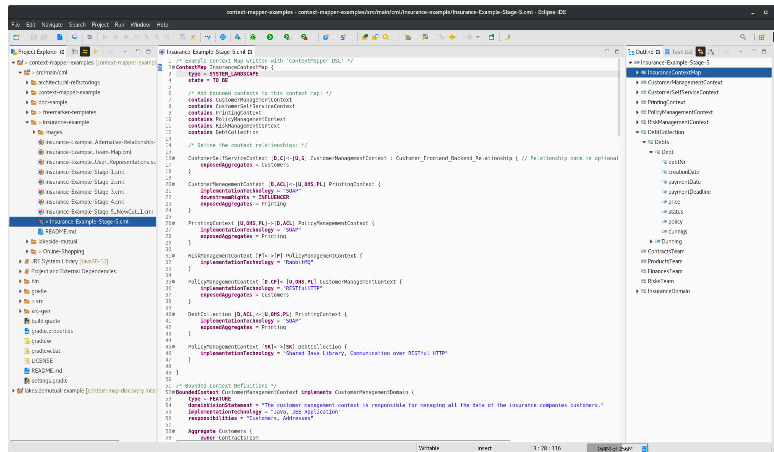


Figura 2.21: Interfaz de Eclipse. Package Explorer en el lado izquierdo.

Por otra parte, *Cursor* presenta una evolución del entorno de desarrollo basado en Visual Studio Code, al integrar de forma nativa asistentes de inteligencia artificial que enriquecen la experiencia sin sobrecargar la interfaz.

Una de sus particularidades es la inclusión de un *sidebar AI chat* (figura 2.22), un panel lateral desde el cual el usuario puede interactuar directamente con la IA, realizar consultas sobre el código, solicitar refactorizaciones, o generar nuevos bloques de código a partir de descripciones en lenguaje natural. Esta interacción se realiza sin perder de vista el contexto visual del proyecto, ya que el chat coexiste de manera no intrusiva con el árbol de archivos y el editor central.

Además, este entorno presenta sugerencias inteligentes en línea (*inline AI suggestions*), integradas directamente en el flujo de edición, evitando ventanas emergentes o modales que interrumpan el trabajo. La plataforma también permite realizar búsquedas contextuales sobre todo el proyecto utilizando lenguaje natural, reduciendo la necesidad de navegación manual dentro del árbol de archivos.

Al heredar la estructura visual de VS Code, Cursor mantiene la familiaridad de un entorno limpio y adaptable, sumando capas de asistencia que optimizan la productividad sin comprometer la simplicidad de la interfaz.

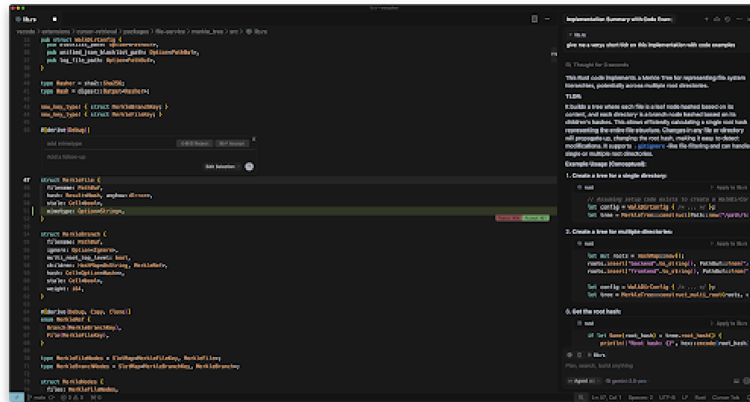


Figura 2.22: Interfaz de Cursor.

En conjunto, estos entornos demuestran enfoques diversos pero complementarios sobre la experiencia de usuario: desde interfaces reducidas pensadas para facilitar el aprendizaje inicial, hasta entornos altamente configurables y extensibles para contextos más avanzados. Este abanico de soluciones permite identificar buenas prácticas y funcionalidades clave que podrían considerarse en futuras mejoras del entorno MateFun.

2.5.2. Evaluación y ejecución en tiempo real (REPL)

La capacidad de evaluar fragmentos de código y obtener resultados inmediatos (Read–Evaluate–Print Loop, REPL) es una funcionalidad clave, ya que favorece la comprensión a través de la experimentación directa. Esta sección analiza cómo los IDEs evaluados permiten al usuario interactuar con el código en tiempo real, facilitando un aprendizaje exploratorio.

Para comenzar, Visual Studio Code no cuenta con un REPL integrado de forma nativa, sin embargo, permite emular esta dinámica mediante extensiones o configuraciones personalizadas. Su funcionalidad de *visualización de definiciones de símbolos* (Alt + F12), como vimos en la sección anterior (figura 2.7), permite inspeccionar implementaciones de métodos o variables en el contexto actual sin abandonar el flujo de trabajo, brindando un acceso rápido a la lógica subyacente sin necesidad de navegar entre archivos. Esta interacción, aunque no es una ejecución en tiempo real, permite optimizar la

comprensión del código durante el desarrollo.

En cuanto a Eclipse, la funcionalidad REPL no está presente de forma directa tampoco, ya que su flujo de trabajo está orientado a la compilación y ejecución de proyectos completos. Sin embargo, ofrece herramientas como *Content Assist* para facilitar la escritura y corrección en tiempo real, mostrando sugerencias contextuales y permitiendo inspeccionar firmas de métodos o estructuras directamente desde el editor.

Por su parte, BlueJ incorpora dos mecanismos clave para la ejecución inmediata de código: el *Code Pad* y la interacción directa con el *object bench*. El *Code Pad* permite escribir y evaluar fragmentos de código Java (expresiones o sentencias completas) sin necesidad de compilar todo el programa (ver figura 2.11). Cada evaluación devuelve el resultado y su tipo de dato, pudiendo referirse tanto a clases estándar como a clases definidas en el proyecto. También admite la creación de variables locales persistentes durante la sesión, la ejecución de secuencias de sentencias y el uso de historial de comandos para reutilizar instrucciones previas. De forma complementaria, el *object bench* (figura 2.10) posibilita crear instancias de clases, invocar métodos y recibir resultados de forma inmediata, funcionando como un REPL visual para Java.

Este es el caso también de DrRacket, quien implementa de manera nativa el paradigma REPL. Su *Interactions Window* permite al usuario evaluar expresiones instantáneamente, observar resultados y modificar entradas sin necesidad de recompilar. La consola interactiva, junto con atajos como Esc-p / Esc-n para repetir comandos, proporciona un flujo de trabajo altamente exploratorio (ver figura 2.18). Esto resulta especialmente útil en la enseñanza de la programación funcional, donde la evaluación de expresiones pequeñas y su resultado visual son fundamentales para la comprensión.

Al ser un entorno completamente web, Replit ofrece una de las experiencias REPL más accesibles. Su consola interactiva permite ejecutar scripts de manera inmediata y visualizar los resultados sin pasos intermedios. Además, cuenta con un panel de vista previa en vivo para proyectos gráficos o web, donde los cambios se reflejan en tiempo real, optimizando la retroalimentación durante el proceso de aprendizaje (ver figura 2.20).

Cuando se pone foco en Thonny, éste lleva la evaluación en tiempo real

a un nivel visual muy didáctico. Este IDE permite recorrer el código paso a paso (utilizando Ctrl+F5) sin necesidad de establecer puntos de interrupción. A través de sus modos de paso grande (F6) y paso pequeño (F7), el entorno muestra cómo se evalúan subexpresiones progresivamente, lo cual refuerza la comprensión de la semántica de ejecución en Python. Además, lo que permite es observar en tiempo real el estado de las variables y su evolución a lo largo de la ejecución (ver figura 2.15).

Por otro lado, Visual Studio Code también ofrece un entorno robusto para la depuración de aplicaciones y el manejo de errores, diseñado para facilitar tanto el análisis como la corrección de problemas en el código. El editor incluye soporte integrado para JavaScript, TypeScript y Node.js, y mediante extensiones disponibles en el Marketplace puede ampliarse a otros lenguajes como Python, C++, Java o C#.

La interfaz de depuración está compuesta por varias secciones que brindan una experiencia visual clara. En la vista Ejecutar y depurar, el usuario puede gestionar la configuración de depuración, iniciar o detener sesiones y acceder a herramientas auxiliares. La barra de herramientas de depuración proporciona controles para continuar, pausar o reiniciar la ejecución, así como recorrer el código paso a paso. Además, la consola de depuración permite visualizar la salida del programa y evaluar expresiones en tiempo real, mientras que la barra lateral de depuración centraliza la información sobre la pila de llamadas, variables activas y puntos de interrupción.

Una de las características más útiles son los breakpoints (puntos de interrupción), que permiten pausar la ejecución en líneas específicas de código. VS Code soporta diversos tipos de breakpoints: básicos, condicionales, de función, de datos, en línea o incluso logpoints, que en lugar de detener el programa registran mensajes en la consola. Esto brinda flexibilidad para diagnosticar comportamientos complejos sin necesidad de modificar manualmente el código con instrucciones de depuración.

Durante una sesión de depuración, el desarrollador puede inspeccionar y modificar el valor de las variables, observar expresiones específicas y navegar en detalle por la pila de ejecución. Asimismo, la consola REPL integrada posibilita ejecutar expresiones adicionales en el mismo contexto del programa en ejecución, lo que facilita probar correcciones rápidas o validar hipótesis sin interrumpir la sesión.

En cuanto al manejo de errores, VS Code detecta y resalta problemas de sintaxis en tiempo real, mostrando advertencias y mensajes en el panel de Problemas. Esto se complementa con el sistema de depuración, que permite

atrapar excepciones en ejecución y analizar de forma interactiva la causa del fallo. Ver figura 2.23

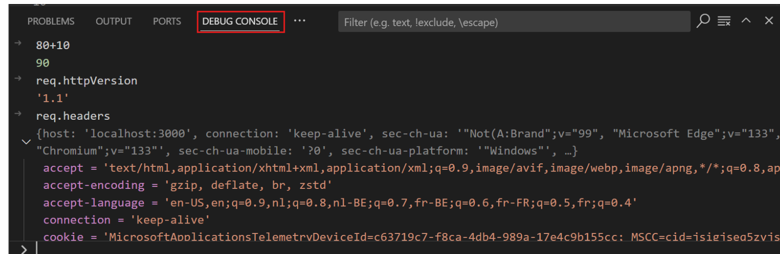


Figura 2.23: Manejo de errores Visual Studio Code.

Por último, Cursor tampoco dispone de un REPL tradicional, pero introduce dinámicas de interacción en tiempo real mediante su integración con inteligencia artificial. A través del *AI Sidebar Chat*, el usuario puede seleccionar bloques de código y solicitar explicaciones, refactorizaciones o generación de nuevos fragmentos, recibiendo respuestas inmediatas sin salir del flujo de trabajo. Esta interacción permite mantener la continuidad en el desarrollo mientras se recibe asistencia contextual (ver figura 2.22).

Como se observa, IDEs como BlueJ, DrRacket, Replit y Thonny ofrecen mecanismos REPL nativos que permiten una interacción directa y continua con el código, optimizando el aprendizaje a través de la experimentación. Por otro lado, entornos como Cursor proponen nuevas formas de interacción en tiempo real mediante inteligencia artificial, adaptándose a flujos de trabajo más avanzados.

2.5.3. Visualización gráfica integrada

A continuación, se describen los IDEs analizados que facilitan la representación visual de funciones de forma clara e interactiva, según su documentación oficial.

Para comenzar, **DrRacket** incluye soporte nativo para graficar funciones mediante la librería `plot`. Esta lo que hace es proporcionar una interfaz flexible para generar gráficos 2D (figura 2.24) como diagramas de dispersión, líneas, contornos e histogramas, así como superficies 3D e isosuperficies.

Además, los gráficos se pueden visualizar directamente como fragmentos interactivos dentro del entorno, donde es posible rotar gráficas 3D (figura 2.25) con el mouse y manipular elementos visuales en tiempo real, lo que permite la vista de diferentes ángulos. También es posible ampliar o reducir los gráficos (zoom), y trabajar con objetos visuales interactivos llamados `plot-snip%`, que soportan overlays o información contextual al pasar el cursor (hover), como mostrar valores numéricos o resaltar regiones del gráfico. Finalmente, esta biblioteca es independiente del backend, lo que permite exportar los gráficos a formatos como PNG, PDF, SVG, entre otros.

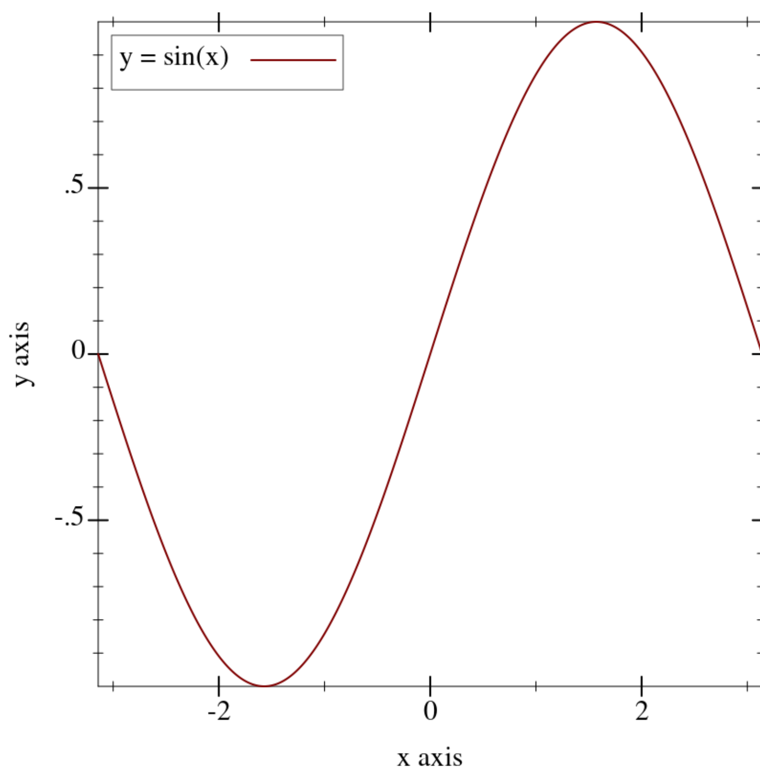


Figura 2.24: Gráfico 2D en DrRacket.

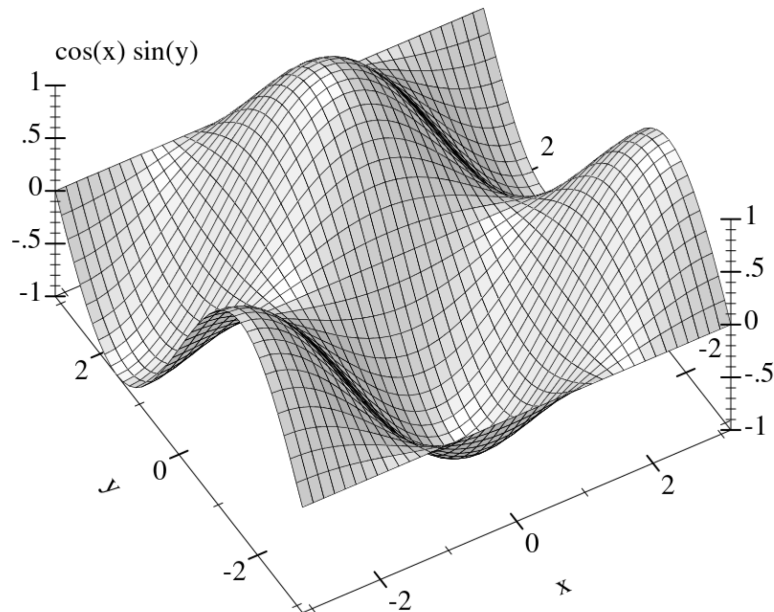


Figura 2.25: Gráfico 3D en DrRacket.

Si nos referimos a **Thonny**, este entorno no integra visualización gráfica matemática de forma predeterminada, sin embargo, su gestor de paquetes facilita la instalación de librerías como `matplotlib` desde la interfaz gráfica (`Tools → Manage packages...`). Una vez instalada, se permite la generación de gráficas procedentes de un código en lenguaje Python que se muestran en ventanas emergentes, sin necesidad de una configuración avanzada.

En cuanto a **Replit**, es posible crear visualizaciones gráficas utilizando librerías comunes como `Plotly`. Estas se ejecutan directamente en el entorno web, mostrando resultados en paneles de vista previa integrados. La integración web permite representar gráficos interactivos 2D y 3D sin requerir configuración local, lo que resulta muy útil para los usuarios. Cuando se genera una gráfica usando la librería mencionada, el entorno mantiene características interactivas navegables mediante el navegador. Los usuarios pueden hacer zoom, desplazarse, seleccionar regiones o puntos, y ver información en

hover o clic sin configuración adicional. Aunque esto depende de librerías externas, su integración dentro del IDE web hace que la experiencia sea casi instantánea y muy accesible desde cualquier dispositivo.

En cambio, si se pone foco en otros entornos de desarrollo vistos, como Visual Studio Code, Eclipse y Cursor, estos no incluyen herramientas gráficas integradas por defecto. Sin embargo, gracias a su extensibilidad, puede añadirse soporte gráfico mediante extensiones o mediante el uso de notebooks interactivos que soporten librerías como `matplotlib` o `Plotly`. Esto posibilita crear visualizaciones, pero requiere configuración adicional y no es un comportamiento nativo del IDE.

2.5.4. Soporte para autocompletado y ayuda contextual

En los entornos de desarrollo integrados (IDE) modernos, una de las funcionalidades más valoradas es el autocompletado inteligente, que permite escribir código de manera más rápida, precisa y con menos errores. En Visual Studio, esta función se conoce como IntelliSense, y proporciona sugerencias contextuales mientras el programador escribe. Estas sugerencias incluyen palabras clave, nombres de variables, métodos, clases y otros símbolos del lenguaje, y se despliegan automáticamente al introducir los primeros caracteres de un identificador. El desarrollador puede insertar la opción deseada simplemente presionando la tecla Tab, lo que acelera la escritura y reduce errores tipográficos, además de favorecer la comprensión del código existente (ver Figura 2.26).

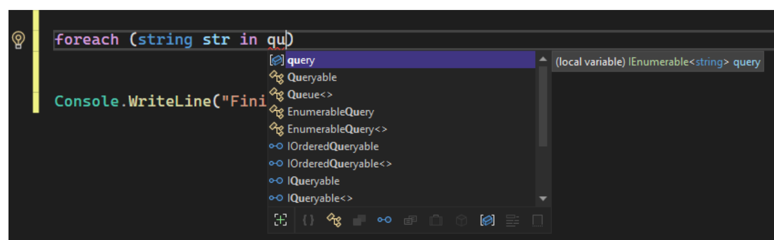


Figura 2.26: Autocompletado de código Visual Studio

En Eclipse, el sistema de asistencia de contenido permite realizar auto-

completado en cualquier parte del documento. Puede activarse manualmente, por ejemplo, mediante la combinación de teclas Ctrl+Espacio, mostrando una lista de elementos relevantes que coinciden con la secuencia de letras ingresada. El orden de las sugerencias sigue un criterio de prioridad que incluye, en primer lugar, campos y variables, seguidos por métodos, funciones, clases, estructuras (structs), uniones (unions), espacios de nombres (namespaces) y enumeraciones. Esta funcionalidad también puede activarse automáticamente al escribir caracteres como".", >" ó "::". Además, al pasar el cursor sobre un elemento sugerido, se muestra una ventana emergente con su firma, y el usuario puede insertarlo directamente en el código fuente (ver 2.27).

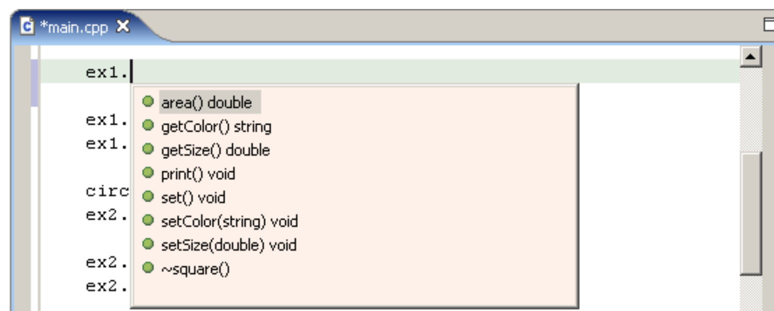


Figura 2.27: Autocompletado de código Eclipse

Por su parte, Thonny ofrece funciones de autocompletado especialmente pensadas para el aprendizaje, ayudando a los estudiantes a explorar las interfaces de programación de aplicaciones (API) disponibles. Esto reduce la necesidad de memorizar métodos y atributos, y favorece la escritura correcta del código. Asimismo, al mostrar únicamente opciones válidas, se minimizan errores por tipografía incorrecta, llamadas inadecuadas o uso incorrecto de nombres de funciones o variables (ver Figura 2.28).

Figura 2.28: Autocompletado de código Thonny

Por último, en Visual Studio Code, la función de autocompletado se potencia mediante IntelliSense, que proporciona sugerencias contextuales más avanzadas para lenguajes como JavaScript, TypeScript, C#, HTML, CSS, SCSS, Less y JSON. Mientras el desarrollador escribe, IntelliSense muestra posibles palabras clave, nombres de variables, métodos y otros elementos del lenguaje, permitiendo insertarlos rápidamente con Tab o Enter, o activarlas manualmente con Ctrl+Espacio.

El sistema incluye filtros inteligentes que permiten, por ejemplo, escribir solo las letras mayúsculas de un método para acotar rápidamente las sugerencias y localizar funciones largas o complejas de manera eficiente.

2.5.5. Depuración visual y manejo de errores

En Visual Studio, la ventana de lista de errores constituye una herramienta central para la depuración del código. Allí se muestran no solo los errores de sintaxis detectados por IntelliSense, sino también los errores de compilación, advertencias y mensajes derivados de análisis estáticos o de políticas empresariales previamente definidas. Cada entrada incluye información detallada acerca de la naturaleza del problema, el archivo involucrado y la ubicación exacta dentro del código fuente. Una de sus funcionalidades más prácticas es la navegación directa: al hacer doble clic sobre un mensaje, el IDE lleva automáticamente al usuario a la línea correspondiente, lo que facilita la corrección inmediata. Además, esta ventana ofrece opciones de filtrado

para visualizar únicamente determinados tipos de mensajes, permite personalizar las columnas visibles según las necesidades del desarrollador y admite la realización de búsquedas específicas dentro del archivo actual, del proyecto activo o de toda la solución. De esta forma, contribuye a mantener un flujo de trabajo ágil y orientado a la calidad del software. En la figura 2.29 se observa la ventana de errores en un proyecto sin incidencias. Al introducir un error —por ejemplo, eliminando un ";"— la ventana muestra la notificación correspondiente, como se aprecia en la figura 2.30.

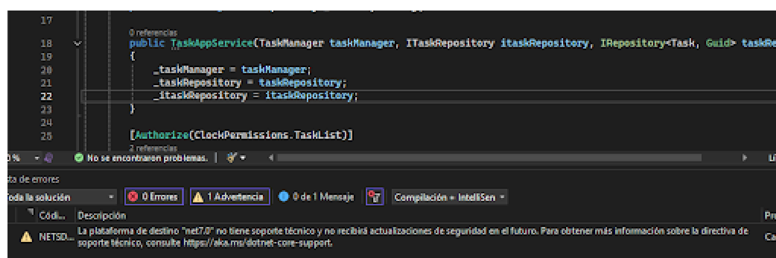


Figura 2.29: Ventana sin errores Visual Studio

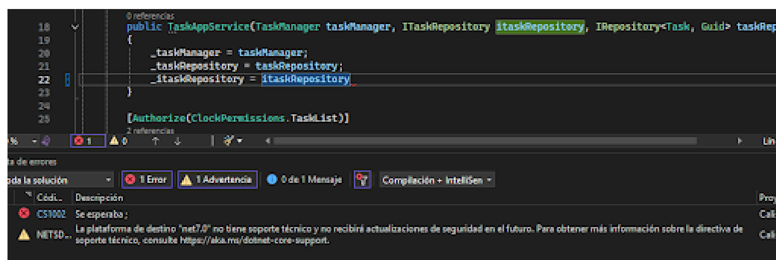


Figura 2.30: Ventana con errores Visual Studio

Por su parte, DrRacket maneja los errores de una forma distinta pero igualmente orientada a la asistencia del usuario. Cuando se produce un error en DrRacket, el entorno subraya el fragmento de código problemático, ya sea en la ventana de definitions o en la de interactions. Además, muestra mensajes de error claros y descriptivos, en ocasiones acompañados de hipervínculos que conducen directamente a la documentación correspondiente. Asimismo, incluye un ícono de "bug" que permite acceder a un depurador visual, el cual

muestra la pila de ejecución y resalta la línea exacta en la que se originó el error.

2.5.6. Colaboración y trabajo en grupo

En un contexto educativo, la colaboración en tiempo real brinda la posibilidad de que múltiples usuarios trabajen simultáneamente sobre un mismo proyecto, lo que facilita la supervisión docente, la resolución colaborativa de problemas y el desarrollo de habilidades de trabajo en equipo. Esta categoría se centra en identificar las herramientas y enfoques que distintos entornos de desarrollo ofrecen para habilitar el trabajo conjunto dentro de la misma plataforma.

Uno de los entornos que más se destaca en este aspecto es **Replit**, cuya arquitectura web permite la colaboración simultánea de hasta cincuenta usuarios en un mismo espacio de trabajo. La plataforma incorpora un sistema de *multiplayer mode* que habilita la edición conjunta de código, similar a lo que ocurre en procesadores de texto colaborativos como Google Docs. Además, se incluye un chat integrado para coordinar acciones, vista previa compartida de resultados en tiempo real y control de versiones automático, lo que lo convierte en una herramienta especialmente poderosa para dinámicas educativas y proyectos grupales.

El IDE **Cursor**, que como ya se dijo está basado en Visual Studio Code, ofrece colaboración indirecta a través de integración con sistemas de control de versiones como GitHub. Si bien no cuenta con edición simultánea en tiempo real al estilo de Replit, sí facilita el trabajo en equipo mediante herramientas de gestión de ramas, revisión de código asistida por inteligencia artificial y generación de mensajes de commit. Estas funcionalidades permiten optimizar flujos de trabajo en grupo y aportan un componente innovador al incorporar sugerencias automáticas en la colaboración.

En entornos como **Visual Studio Code** y **Eclipse**, la colaboración se apoya principalmente en integraciones externas. En el caso de VS Code, extensiones como *Live Share* posibilitan sesiones de edición compartida, depuración colaborativa y control de terminales entre usuarios. Eclipse, por su parte, incorpora integraciones con sistemas de control de versiones como Git y CVS, que aunque no brindan edición simultánea, permiten coordinar tra-

bajo en proyectos grupales de manera estructurada y segura.

En contraste, IDEs como **BlueJ**, **Thonny** y **DrRacket** priorizan la simplicidad y la experiencia individual del estudiante, careciendo de funcionalidades nativas de colaboración en tiempo real. No obstante, su enfoque pedagógico los convierte en herramientas igualmente relevantes, donde la supervisión y el trabajo grupal pueden apoyarse en entornos externos como repositorios compartidos o plataformas de gestión de clases.

2.6. Propuestas de nuevas funcionalidades para MateFun

A partir del resultado de la investigación realizada sobre diferentes entornos de desarrollo integrados (IDEs) y considerando la situación actual de MateFun, se presenta un conjunto de funcionalidades que se proponen como trabajo futuro para mejorar significativamente la experiencia de los usuarios finales. Estas propuestas se presentan siguiendo un esquema que contempla el estado actual de la plataforma, los problemas detectados y las mejoras sugeridas, indicando además una posible implementación en la arquitectura de MateFun. Las funcionalidades que fueron efectivamente implementadas en este proyecto se describen en los casos de uso del capítulo 3.

Manejo de errores en tiempo real

Situación actual: actualmente, los errores de sintaxis o semántica en MateFun se muestran únicamente luego de compilar la solución. Notar que en la imagen 2.31 la linea 8 tiene error de sintaxis, ya que le falta un paréntesis al final, pero el IDE no nos marca ningún error. Recien al cargar el programa, se indica el error de sintaxis, ver figura 2.32.

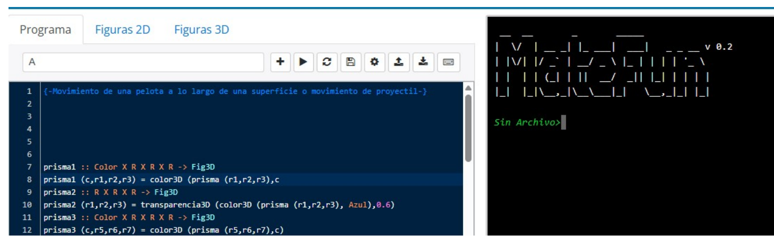


Figura 2.31: Manejo de errores en MateFun

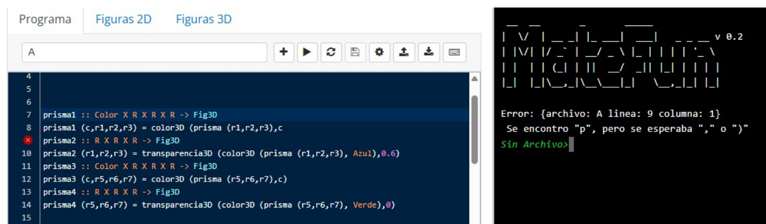


Figura 2.32: Manejo de errores en MateFun II

Problema: este enfoque genera demoras en el ciclo de retroalimentación y puede dificultar el aprendizaje, ya que el usuario no identifica el error hasta que ejecuta el programa completo.

Propuesta: implementar un sistema de detección de errores en tiempo real que subraye en rojo el código problemático a medida que se escribe, similar al funcionamiento de entornos como Visual Studio o Thonny.

Posible implementación: mejora en el *front-end*, con un motor de análisis léxico y sintáctico ligero que ejecute en segundo plano al momento de edición, generando advertencias visuales inmediatas.

Resaltado y edición múltiple de palabras

Situación actual: MateFun no permite identificar de forma visual todas las ocurrencias de una palabra en un archivo, ni modificarlas en conjunto.

Problema: la refactorización manual de variables o funciones puede ser propensa a errores y consumir tiempo.

Propuesta: al seleccionar una palabra, todas sus apariciones se resalten automáticamente, con la opción de modificarlas en simultáneo.

Posible implementación: funcionalidad en el *front-end*, vinculada al editor de texto, que aproveche algoritmos de búsqueda incremental y operaciones de edición múltiple.

Mejoras en el autocompletado

Situación actual: el sistema de autocompletado en MateFun es básico y poco atractivo visualmente. Ver figura 2.33.

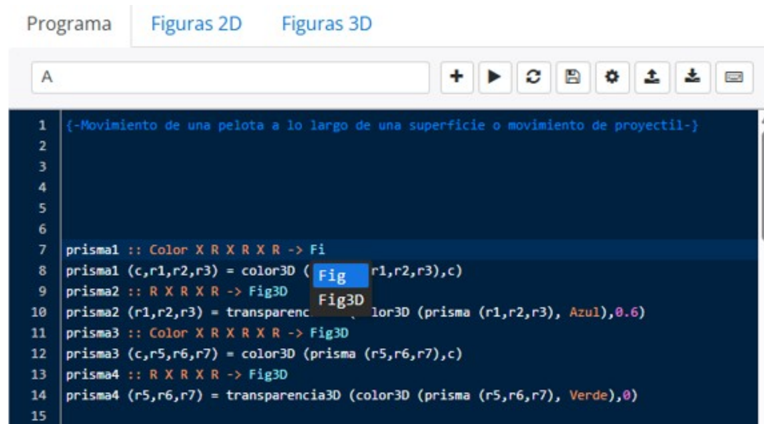


Figura 2.33: Autocompletado en MateFun

Problema: la experiencia de usuario se ve limitada y no fomenta el uso eficiente de las funciones disponibles. Visualmente no es atractivo.

Propuesta: enriquecer el autocompletado con un diseño visual más claro, incorporar información contextual (parámetros esperados, descripciones) y facilitar la navegación en las sugerencias.

Posible implementación: requiere ajustes en el *front-end*, con posible conexión al *back-end* para obtener información de contexto semántico de funciones definidas.

Corrección automática de sintaxis

Situación actual: actualmente, la corrección de errores de formato (alineación de corchetes, indentación) debe hacerse manualmente.

Problema: esto puede llevar a confusiones en los usuarios y generar código que no es claro ni legible.

Propuesta: incorporar un sistema de formateo automático que ajuste indentaciones, alinee corchetes y mantenga un estilo de código homogéneo.

Posible implementación: motor de formateo en el *front-end*, que se ejecute al guardar el archivo o mediante un comando específico.

Panel de errores y advertencias

Situación actual: la información sobre errores aparece únicamente al compilar, y no está centralizada en un panel, sino que se muestra en el editor de texto con la línea marcada donde se genera el error.

Problema: dificulta tener una visión global de los problemas presentes en el archivo.

Propuesta: incluir una sección dedicada para listar errores y advertencias detectados en el editor, incluso antes de ejecutar la compilación completa.

Posible implementación: componente de *front-end* que consolide advertencias generadas por el analizador sintáctico en tiempo real.

Visualización de variables en figuras

Situación actual: actualmente, los valores de las variables no se reflejan dinámicamente en las figuras generadas o en las funciones implementadas.

Problema: esto dificulta la comprensión de la relación entre el código y su representación visual.

Propuesta: incorporar un sistema que muestre en tiempo real cómo van cambiando las variables dentro de la figura generada o la función ejecutada, brindando retroalimentación inmediata. Esto es similar a lo que se mostró anterior en *Thonny*.

Posible implementación: combinación de mejoras en el *front-end* (figuras dinámicas) con un soporte en el *back-end* que proporcione valores actualizados de las variables en ejecución.

Valor en intersecciones en gráficos 3D

Situación actual: en las figuras 3D de MateFun, actualmente no se ofrece información contextual al interactuar con la visualización. Por ejemplo, obtener el valor de una intersección en esa figura.

Problema: la interpretación de las gráficas tridimensionales puede resultar compleja para los estudiantes, dificultando la identificación precisa de valores en puntos específicos.

Propuesta: agregar una funcionalidad que muestre el valor numérico en la intersección del gráfico cuando el usuario pase el cursor o seleccione un punto de interés.

Posible implementación: desarrollo en el *front-end* que capture eventos de interacción (hover o clic) y despliegue dinámicamente los valores en pantalla, aprovechando la lógica gráfica ya existente en MateFun.

Ejecución parcial de código

Situación actual: MateFun requiere compilar y ejecutar el archivo completo para ver resultados.

Problema: esta restricción limita la experimentación y puede ser no ser eficiente en programas que son extensos.

Propuesta: permitir cargar y ejecutar únicamente las secciones de código seleccionadas, sin necesidad de compilar todo el archivo.

Posible implementación: modificación en el *back-end* para admitir ejecución parcial, con soporte en el *front-end* para seleccionar el bloque a ejecutar.

Scroll horizontal

Situación actual: el editor incorpora un desplazamiento vertical pero no uno que sea de forma horizontal.

Problema: esto dificulta la lectura sobre todo de el comienzo de las líneas.

Propuesta: incorporar la posibilidad de hacer scroll horizontal en el editor de texto, mejorando la legibilidad en casos específicos.

Posible implementación: cambio exclusivamente en el *front-end*, mediante configuración del componente de edición de texto.

Conclusión de las propuestas

En síntesis, las funcionalidades propuestas buscan reducir las limitaciones actuales de MateFun y alinearlo con las mejores prácticas observadas en otros entornos de desarrollo analizados. La incorporación de herramientas como el manejo de errores en tiempo real, la edición múltiple de ocurrencias, un autocompletado enriquecido y un sistema de corrección automática de

sintaxis apuntan a optimizar la experiencia de edición y a mejorar la claridad del código.

Asimismo, la integración de un panel de errores y advertencias, junto con la visualización dinámica de variables dentro de las figuras, contribuiría a fortalecer la retroalimentación inmediata para los estudiantes o quienes hagan uso de la herramienta, generando un uso menos propenso a la frustración. Por otra parte, funcionalidades como la ejecución parcial de programas y la posibilidad de scroll horizontal representan mejoras en la flexibilidad y usabilidad del entorno.

De este modo, estas propuestas no solo modernizan la interfaz y la experiencia de usuario, sino que también contribuyen a adaptar al IDE de MateFun a las necesidades actuales, facilitando tanto la enseñanza como la exploración autónoma por parte de los estudiantes.

Capítulo 3

Análisis y Diseño de la solución

.

3.1. Descripción general del sistema

Como ya se detalló anteriormente, MateFun es un entorno de desarrollo orientado a la enseñanza de la programación funcional en el ámbito educativo. El sistema integra un editor de código, un intérprete del lenguaje MateFun y herramientas de visualización gráfica en dos y tres dimensiones, permitiendo a los usuarios escribir, ejecutar y analizar funciones matemáticas dentro de un mismo entorno.

Este proyecto se desarrolla sobre una versión preexistente del sistema, cuya arquitectura y funcionamiento general ya se encontraban definidos. En este contexto, el trabajo realizado se centró en la extensión de ciertas funcionalidades del entorno, sin modificar los componentes fundamentales del sistema ni su arquitectura base.

3.2. Requerimientos del sistema

El sistema permite a los usuarios crear y editar archivos que contienen definiciones de funciones, ejecutar código de manera interactiva y visualizar los resultados mediante gráficos y figuras. La interacción con el entorno se realiza a través de una interfaz web, diseñada para facilitar el aprendizaje y la exploración de conceptos matemáticos y de programación funcional.

En el marco de este proyecto, se mantuvieron los requerimientos generales del sistema existente, incorporando nuevas funcionalidades orientadas a mejorar la experiencia de uso y ampliar las capacidades del entorno. En particular, se buscó mejorar la interacción con las visualizaciones gráficas en dos dimensiones, permitir una mayor flexibilidad en la organización del trabajo y facilitar la navegación y reutilización de código mediante el uso de funciones y librerías.

3.3. Casos de uso

En el marco de este proyecto se definieron un conjunto de requerimientos funcionales y no funcionales, orientados a extender las capacidades del entorno MateFun sin modificar su arquitectura base. Dichos requerimientos se enfocan principalmente en mejorar la interacción con gráficos en dos dimensiones, facilitar la navegación y reutilización de código, y optimizar la organización de la interfaz de usuario.

Asimismo, se identificaron y modelaron los casos de uso correspondientes a las nuevas funcionalidades incorporadas, describiendo la interacción entre el usuario y el sistema para cada una de ellas.

A continuación, se presenta un diagrama general y una breve descripción global de cada uno de los casos de uso definidos:

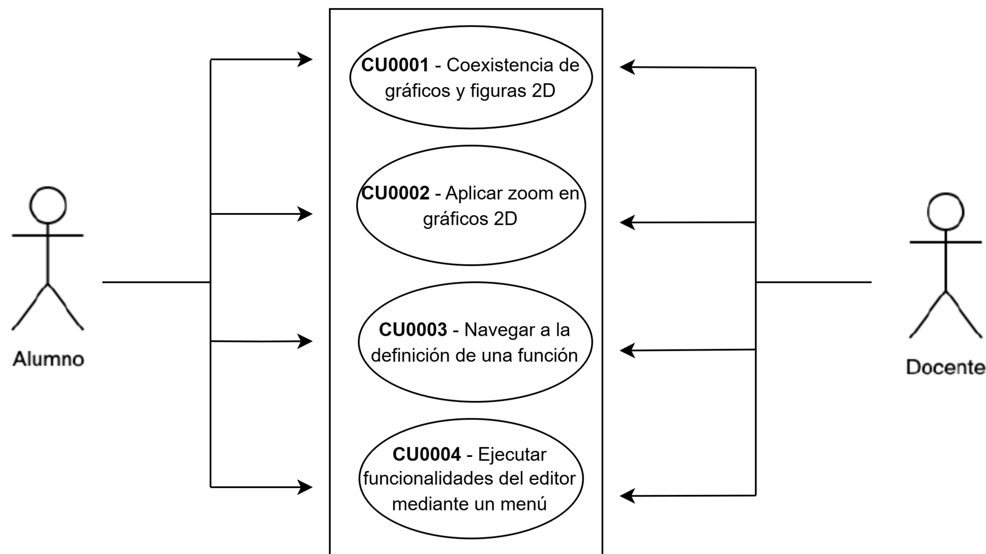


Figura 3.1: Diagrama general de casos de uso

- **CU001 – Coexistencia de gráficos y figuras 2D:** Permite al usuario visualizar simultáneamente múltiples funciones y figuras en un mismo panel de gráficos en dos dimensiones, manteniendo el orden de solicitud y posibilitando la superposición de elementos.

- **CU002 – Aplicar zoom en gráficos 2D:** habilita al usuario a modificar el nivel de zoom de los gráficos en dos dimensiones, permitiendo un zoom general o independiente por eje, sujeto a restricciones según el tipo de elementos visualizados.

- **CU003 – Navegar a la definición de una función:** permite al usuario acceder rápidamente a la definición de una función utilizada en el código, ya sea mediante la visualización de información contextual o mediante la navegación directa al archivo donde se encuentra definida.

- **CU004 – Ejecutar acciones del editor mediante el menú:** posibilita la ejecución de acciones del editor a través de un menú estructurado, facilitando el acceso a las funcionalidades disponibles y mejorando la usabilidad del entorno.

Dado el nivel de detalle requerido para la correcta especificación de estos casos de uso, su descripción completa se presenta en un documento anexo titulado “Documento de requerimientos y casos de uso”.

3.4. Arquitectura del sistema

La arquitectura general del sistema se mantuvo sin modificaciones estructurales durante el desarrollo del proyecto. Tal como se describe en el documento anexo de Arquitectura entregado, y como se puede apreciar en la imagen 3.2 el sistema se organiza en una arquitectura en capas que separa la presentación, la lógica de negocio, la persistencia y el intérprete del lenguaje.

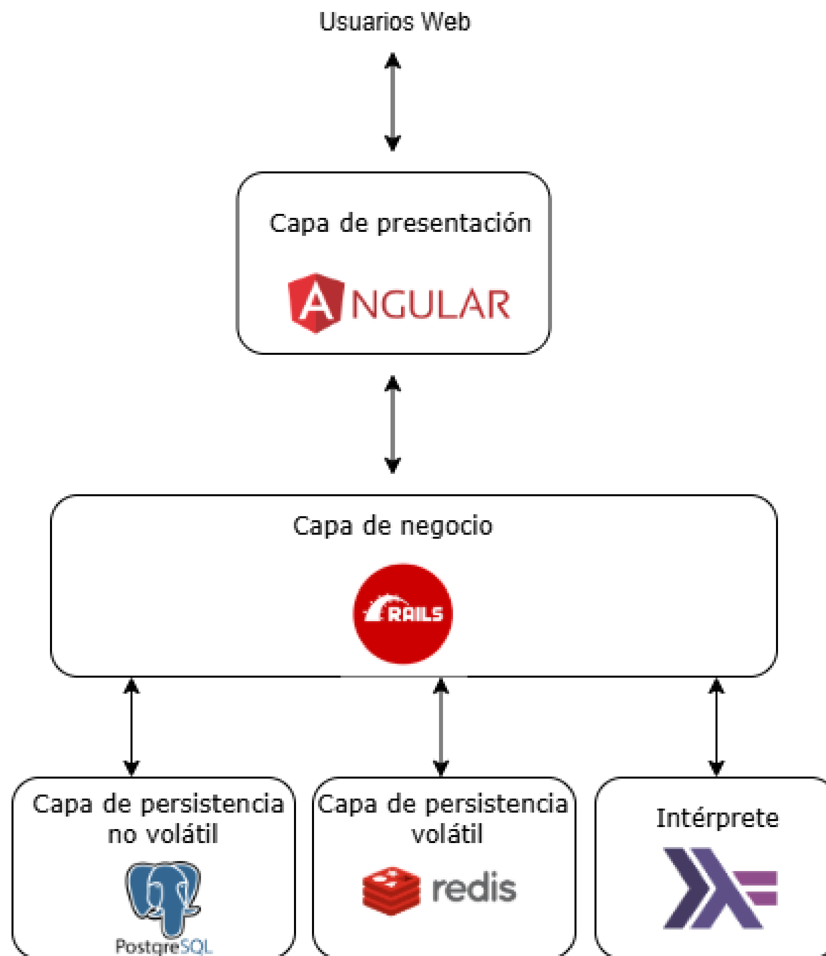


Figura 3.2: Diagrama de capas de MateFun

Las extensiones desarrolladas se integran principalmente en la capa de presentación, respetando los mecanismos de comunicación existentes con el backend y el intérprete. Esta decisión permitió preservar la estabilidad del sistema y asegurar la compatibilidad con desarrollos previos.

3.5. Diseño de la solución

El diseño adoptado se basó en la extensión incremental del sistema existente. Dado que el objetivo del proyecto no fue rediseñar MateFun, sino

ampliar sus capacidades, las nuevas funcionalidades fueron diseñadas para integrarse de forma natural con los componentes ya existentes.

Las mejoras relacionadas con la visualización gráfica se implementaron en los componentes gráficos actuales, permitiendo la coexistencia de gráficos y figuras en dos dimensiones dentro de un gráfico, así como un manejo más flexible del zoom en los ejes. Por otro lado, las funcionalidades vinculadas a la navegación entre definiciones de funciones y al soporte de librerías se diseñaron como extensiones del editor de código, con el objetivo de facilitar la comprensión y reutilización de funciones en programas más complejos.

Asimismo, se realizaron ajustes en la interfaz y en el menú del entorno, con el fin de mejorar la accesibilidad y organización de las funcionalidades disponibles, manteniendo la coherencia con el diseño general del sistema.

Capítulo 4

Implementación y Pruebas

4.1. Metodología de trabajo

La organización del trabajo se basó en una combinación de colaboración conjunta y división de tareas. Las actividades más complejas o de mayor alcance fueron desarrolladas en equipo, mientras que aquellas más acotadas se abordaron de forma individual, quedando posteriormente sujetas a una revisión por parte del otro integrante. Ambos miembros del equipo asumimos roles tanto de desarrollo como de pruebas, lo que permitió que cada funcionalidad implementada fuera testeada por ambos, fortaleciendo la calidad del producto.

Se realizaron reuniones periódicas de planificación en las que analizamos en conjunto las funcionalidades a desarrollar y definimos la distribución del trabajo. Para el seguimiento y control del avance se utilizó una hoja compartida en Google Docs, que permitió mantener un registro actualizado de las tareas completadas y pendientes. No se consideró necesario incorporar herramientas más avanzadas como Trello o ClickUp, dado que la comunicación fluida dentro del equipo resultó suficiente para asegurar una buena coordinación.

La comunicación fue un aspecto central del proceso. Se empleó WhatsApp para las interacciones cotidianas y Discord para las reuniones sincrónicas, aprovechando su versatilidad y facilidad de uso. Dado que la infraestructura del proyecto ya estaba definida, se utilizó GitLab como repositorio para el código fuente, lo que facilitó el control de versiones y el seguimiento del progreso. Asimismo, la documentación del proyecto se gestionó mediante Google

Drive y Overleaf, herramientas que permitieron una colaboración eficiente y acceso simultáneo a los materiales del trabajo.

4.2. Entorno de desarrollo

El proyecto se desarrolló sobre una aplicación preexistente, construida a lo largo de varios proyectos previos, cuya arquitectura y tecnologías principales ya se encontraban definidas, pero no suficientemente documentadas. El trabajo consistió en elaborar una documentación rigurosa de todos los componentes de MateFun y la incorporación y modificación de funcionalidades dentro de dicho sistema, respetando la estructura y el diseño previamente establecidos y solucionando problemas heredados como los que se detallan en la sección 4.5.

La aplicación existente se basa en una arquitectura cliente-servidor. El frontend está desarrollado en Angular, mientras que el backend utiliza Ruby on Rails. En el marco de este proyecto, se trabajó principalmente sobre el frontend, realizando extensiones y ajustes según los requerimientos planteados.

La persistencia de los datos se gestiona mediante una base de datos PostgreSQL, utilizada para el almacenamiento permanente de la información del sistema. Asimismo, se emplea Redis como base de datos no persistente, destinada al manejo de datos temporales y a la optimización de determinadas operaciones.

El desarrollo se llevó a cabo en entornos locales bajo el sistema operativo Windows, utilizando Visual Studio Code como entorno de desarrollo integrado (IDE). Ambos trabajamos con configuraciones equivalentes de herramientas y tecnologías, lo que permitió mantener consistencia en la implementación y facilitar la integración del trabajo realizado.

El control de versiones y la gestión del código fuente se realizaron mediante GitLab, herramienta provista como parte de la infraestructura del proyecto. No se utilizaron contenedores ni entornos de virtualización, apoyándose el desarrollo en la infraestructura previamente establecida.

4.3. Implementación de casos de uso

4.3.1. CU001 - Coexistencia de gráficos y figuras 2D

La coexistencia de gráficos de funciones y figuras en el panel bidimensional se implementó mediante un esquema de comunicación basado en eventos entre la terminal integrada, el intérprete y el componente de visualización. Desde el punto de vista del usuario, la solicitud de graficación se realiza a través de la terminal: las funciones se visualizan utilizando el comando `?grafica`, mientras que las figuras se generan invocando directamente las funciones o constructores correspondientes. En ambos casos, la evaluación del comando es delegada al intérprete.

La terminal integrada actúa como intermediaria, enviando el texto ingresado al intérprete y procesando las respuestas obtenidas. Cuando la evaluación produce un resultado gráfico bidimensional, el intérprete retorna un mensaje estructurado que identifica el tipo de salida generada. A nivel de implementación, se distinguen mensajes asociados a gráficos de funciones y mensajes asociados a figuras geométricas.

Ante la recepción de estos mensajes, el componente `Graph2DComponent` actualiza su estado interno, el cual mantiene una estructura acumulada que representa todos los elementos a mostrar en un mismo gráfico. Cada nuevo resultado se normaliza al formato requerido por la librería de graficación y se incorpora a dicha estructura, diferenciando funciones y figuras mediante un identificador de tipo.

La coexistencia se logra porque, ante cada actualización, el componente vuelve a renderizar el gráfico completo utilizando la lista acumulada de elementos, de modo que todos se visualizan simultáneamente dentro del mismo sistema de ejes. El orden de solicitud se preserva al incorporar los nuevos elementos al final de la estructura, lo que define tanto el orden cronológico como la superposición visual cuando corresponde.

Para el flujo alternativo de limpieza, el panel incluye un control que permite reiniciar el estado del componente de graficación, eliminando todos los elementos acumulados y dejando el panel vacío. En caso de errores de ejecución o referencias a elementos no definidos, el intérprete retorna mensajes de error a la terminal sin generar eventos gráficos, por lo que el panel conserva la visualización previa.

4.3.2. CU002 - Aplicar zoom en gráficos 2D

La funcionalidad de zoom en el panel de gráficos bidimensionales se implementó dentro del componente de visualización `Graph2DComponent`, el cual administra el estado del gráfico y su renderizado utilizando la librería `function-plot`. El sistema ofrece dos modalidades de zoom: un zoom general, que actúa de forma uniforme sobre ambos ejes, y un zoom independiente por eje, disponible únicamente cuando el gráfico contiene funciones.

En el flujo principal, correspondiente al zoom general, el usuario interactúa con los controles de acercar y alejar del panel. Estas acciones invocan métodos del componente que delegan directamente la operación de zoom en la instancia actual del gráfico, aplicando el escalado de manera uniforme sobre ambos ejes y actualizando la visualización en forma inmediata.

Como flujo alternativo, el sistema permite aplicar zoom independiente sobre el eje X o el eje Y. Para ello, el panel incorpora un selector que modifica el estado interno del componente (`settings.zoomType`). Cuando se selecciona un eje, las operaciones de zoom ajustan únicamente el dominio correspondiente mediante una rutina específica, manteniendo fijo el otro eje y forzando el redibujado del gráfico para reflejar el nuevo nivel de ampliación.

La disponibilidad del zoom independiente se controla automáticamente en función del contenido del panel. El componente mantiene una marca interna que indica la presencia de figuras geométricas; si se detecta al menos una figura, el sistema deshabilita la selección de eje, fuerza el modo de zoom general y muestra un indicador visual de advertencia. De este modo, se garantiza un comportamiento consistente, permitiendo siempre el zoom general y restringiendo el zoom por eje a los escenarios en los que resulta conceptualmente adecuado.

4.3.3. CU003 - Navegar a la definición de una función

La navegación a la definición de una función se implementó integrando el editor de código basado en `CodeMirror` con un mecanismo de resolución de símbolos que opera sobre el archivo actual y los archivos incluidos mediante las directivas `include/incluir`. El objetivo de esta funcionalidad es permitir al usuario inspeccionar rápidamente la definición de una función y, cuando corresponde, navegar al archivo donde dicha definición se encuentra.

En el flujo principal, correspondiente a la visualización de información mediante *hover*, el componente principal del entorno registra eventos de mo-

vimiento del cursor sobre el editor. Cuando el cursor se posiciona sobre un identificador válido, el sistema aplica un retardo breve para evitar búsquedas innecesarias y luego intenta resolver la definición de la función. La resolución se realiza primero sobre el archivo actual y, si no se encuentra una definición local, se extiende a los archivos incluidos, los cuales se identifican analizando las cláusulas de inclusión presentes en el programa. En caso de ser necesario, el contenido de los archivos incluidos se carga dinámicamente. Una vez localizada la definición, se extrae su firma y, si existe, el comentario asociado, y se muestra esta información en un componente emergente cercano al cursor.

Como flujo alternativo, el sistema permite la navegación directa a la definición mediante la acción de *Ctrl+Click*. Ante esta interacción, el editor identifica el identificador seleccionado y reutiliza el mismo mecanismo de resolución de símbolos. Si la definición se encuentra en el archivo actual, el editor desplaza el cursor hasta la línea correspondiente. En caso contrario, el sistema abre el archivo donde se encuentra definida la función en una nueva pestaña (o activa la existente), posicionando el cursor sobre la definición para hacer explícito el salto realizado.

Finalmente, si el identificador no puede resolverse ni en el archivo actual ni en los archivos incluidos, el sistema no muestra información adicional ni realiza ninguna navegación. Este comportamiento asegura que la funcionalidad se ofrezca únicamente cuando la definición de la función es accesible a partir de las dependencias efectivamente incluidas en el programa.

4.3.4. CU004 - Ejecutar acciones del editor mediante el menú

La ejecución de acciones del editor mediante el menú se implementó integrando un conjunto de controles de interfaz gráfica asociados al editor de código, desarrollado sobre el componente CodeMirror. El objetivo de esta funcionalidad es permitir al usuario aplicar operaciones habituales sobre el contenido del editor de forma estructurada y accesible, sin necesidad de recurrir exclusivamente a atajos de teclado.

Desde el punto de vista del usuario, las acciones se encuentran organizadas en un menú del editor, el cual presenta las opciones disponibles agrupadas según su funcionalidad. Cada opción del menú incluye una referencia visual a la acción que representa y, cuando corresponde, el atajo de teclado asociado, facilitando su identificación y uso.

A nivel de implementación, cada opción del menú se vincula a un manejador de eventos definido en el componente del editor. Cuando el usuario selecciona una acción, el sistema intercepta el evento y delega su procesamiento a un método específico encargado de ejecutar la operación solicitada sobre el estado actual del editor. Estas operaciones incluyen acciones de edición y manipulación del contenido, las cuales actúan directamente sobre la instancia activa de CodeMirror.

Antes de ejecutar una acción, el sistema valida el contexto actual del editor para determinar si la operación es aplicable. Esta validación se realiza utilizando el estado interno del editor, lo que permite habilitar o deshabilitar dinámicamente las opciones del menú según corresponda. De este modo, se evita la ejecución de acciones inválidas en determinados contextos, como aquellas que requieren una selección previa o un contenido existente.

En el flujo principal, cuando la acción seleccionada es válida, el sistema la ejecuta inmediatamente, actualizando el contenido del editor y reflejando los cambios en la interfaz de usuario de forma consistente. El editor mantiene su estado sincronizado, garantizando que las modificaciones realizadas a través del menú se integren correctamente con el resto de las funcionalidades disponibles.

Como flujo alternativo, si el usuario selecciona una acción que no es aplicable al estado actual del editor, el sistema detecta esta situación y no ejecuta la operación solicitada, manteniendo el contenido y el estado del editor sin modificaciones. Este comportamiento asegura un uso robusto de la funcionalidad y evita efectos no deseados.

4.4. Pruebas realizadas

Las pruebas realizadas en el marco de este proyecto tuvieron como objetivo verificar el correcto funcionamiento de las nuevas funcionalidades incorporadas al entorno MateFun, así como su adecuada integración con el sistema preexistente. Dado que MateFun es una aplicación previamente desarrollada, el enfoque de las pruebas estuvo centrado principalmente en la validación funcional y en la verificación de que las extensiones implementadas no afectan negativamente el comportamiento original del sistema.

Las pruebas fueron realizadas de forma manual, siguiendo los casos de uso definidos en el documento de requerimientos y casos de uso, ejecutando distintos escenarios representativos del uso esperado de la aplicación.

Debido a quetodo el desarrollo del proyecto estuvo enfocado en el Frontend, se hicieron muchas pruebas sobre la aplicación cliente MateFun.

4.4.1. Estrategia de pruebas

La estrategia de pruebas adoptada se basó en pruebas funcionales y de integración a nivel de sistema. Para cada caso de uso se verificó que:

- El sistema responda de acuerdo con el comportamiento esperado.
- Las nuevas funcionalidades convivan correctamente con las funcionalidades existentes de MateFun.

4.4.2. Pruebas funcionales

- **Coexistencia de gráficos y figuras 2D:** Se verificó que el sistema permite la visualización simultánea de múltiples gráficos y figuras en el panel de gráficos 2D. Para ello, se cargaron programas que definen distintas funciones y figuras, solicitando su visualización de manera secuencial.

  El resultado observado fue que todos los elementos solicitados se muestran correctamente en un mismo gráfico, respetando el orden de invocación y sin eliminar los elementos previamente dibujados. Asimismo, se comprobó el correcto funcionamiento de la opción de limpieza del panel, la cual elimina todos los elementos visualizados.

- **Aplicación de zoom en gráficos 2D** Se realizaron pruebas sobre gráficos que contienen únicamente funciones, así como sobre gráficos que incluyen figuras. En el primer caso, se verificó que el sistema permite aplicar zoom independiente sobre cada eje, actualizando correctamente la visualización. En el segundo caso, se comprobó que la opción de zoom independiente se encuentra deshabilitada y que el sistema informa dicha restricción al usuario mediante la interfaz.

- **Navegación a la definición de una función** Se verificó la funcionalidad de navegación entre archivos mediante la interacción con funciones definidas en archivos externos. Al posicionar el cursor sobre el nombre de una función, el sistema muestra correctamente un componente

emergente con la definición y el comentario asociado. Asimismo, al realizar la acción de control+click, se comprobó que el sistema abre el archivo correspondiente en una nueva pestaña, posicionándose sobre la definición de la función seleccionada.

- **Ejecución de acciones del editor mediante el menú** Se validó que las acciones del editor accesibles desde el menú (como ejecutar, guardar o descargar archivos) funcionan correctamente y producen el mismo efecto que sus atajos de teclado correspondientes. Esto mejora la accesibilidad del entorno sin afectar la funcionalidad existente.

4.5. Problemas encontrados durante el desarrollo

Para el despliegue de la aplicación frontend no se presentaron inconvenientes. Sin embargo, al intentar levantar el backend de forma local en un sistema operativo Windows comenzaron a surgir dificultades.

En primer lugar, se procedió a instalar Ruby on Rails, PostgreSQL y Redis. Posteriormente, se ejecutaron los comandos correspondientes para la instalación de las dependencias mediante bundle install. El siguiente paso consistía en la creación de la base de datos en PostgreSQL, pero en esta instancia surgió el problema principal: Ruby no lograba establecer conexión con la base de datos.

Se revisó el archivo database.yml para corroborar que el usuario de la base estuviera configurado correctamente, pero el error persistía. Ante esta dificultad, se contactó al estudiante que trabajó en el proyecto previo y quien fue el que incorporó la nueva tecnología al proyecto, quien nos sugirió utilizar WSL (Windows Subsystem for Linux). Su recomendación se basaba en las limitaciones de compatibilidad que presenta Ruby al ejecutarse directamente en Windows.

WSL es una funcionalidad de Windows que permite ejecutar un entorno Linux completo dentro del propio sistema, sin necesidad de recurrir a máquinas virtuales ni a un arranque dual. Siguiendo esta sugerencia, instalamos las dependencias necesarias en WSL y procedimos a levantar la aplicación en Linux en lugar de Windows.

Además de este inconveniente principal, durante el trabajo con el frontend en Angular se identificaron fallos en la versión anterior del sistema relacio-

nados con la carga de archivos y la generación de gráficos. En particular:

- No se estaba pudiendo cargar correctamente los archivos desde la interfaz, lo que impedía que ciertos componentes funcionaran.
- Las gráficas no se renderizaban adecuadamente, lo que implicaba que funciones esenciales de visualización no se ejecutaban como se esperaba.

Para resolver estas dificultades se realizaron ajustes en los componentes de frontend afectados.

4.6. Dedicación

La figura 4.1 muestra la distribución relativa del esfuerzo del proyecto a lo largo del tiempo, normalizada por mes y clasificada por áreas de trabajo. Para cada mes, el total del esfuerzo realizado se considera como el 100 %, lo que permite visualizar cómo se fue modificando el foco del proyecto durante su desarrollo.

En los primeros meses se observa una mayor dedicación al estado del arte y al análisis, correspondientes a la etapa inicial de investigación y comprensión del problema. Posteriormente, el esfuerzo se desplaza hacia las actividades de diseño, que permiten definir las soluciones a implementar. En la fase intermedia del proyecto, la implementación concentra la mayor parte del trabajo, reflejando el desarrollo efectivo de las funcionalidades propuestas.

Hacia los últimos meses, la dedicación se orienta principalmente a las pruebas de la aplicación y a la documentación, actividades necesarias para validar el funcionamiento del sistema y consolidar los resultados obtenidos. La configuración de la estación de trabajo se concentra en los meses iniciales del proyecto, dado que constituye una actividad de soporte requerida para el desarrollo posterior. Cabe destacar que los valores presentados no representan horas de trabajo, sino una medida relativa del énfasis puesto en cada área durante cada mes del proyecto.

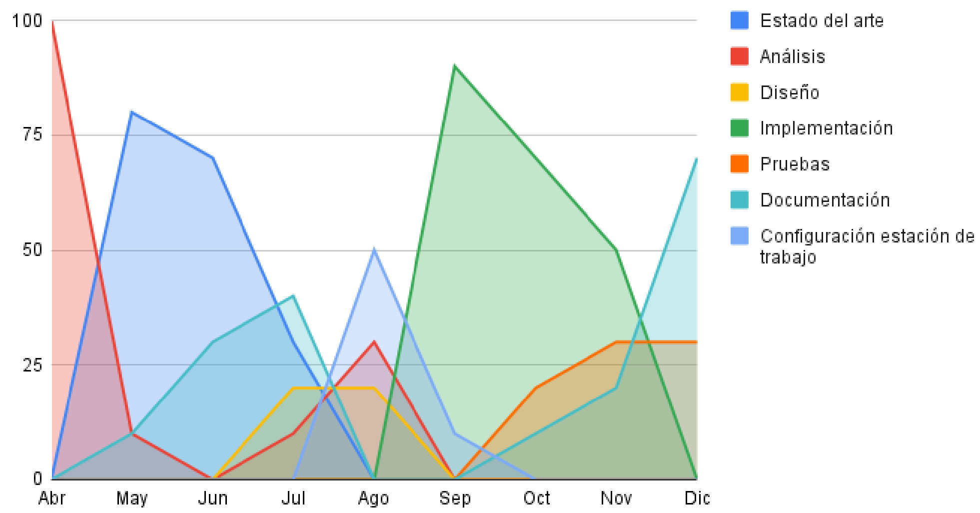


Figura 4.1: Distribución relativa del esfuerzo

Capítulo 5

Conclusiones y Trabajo Futuro

5.1. Conclusiones

En el presente Proyecto de Grado, como ya se mencionó a lo largo de este informe, el trabajo se centró en mejorar aspectos concretos del entorno existente de MateFun, sin modificar su arquitectura base, priorizando la experiencia de usuario y la coherencia con los objetivos didácticos del sistema.

Entre los principales aportes del proyecto se encuentran, por un lado, la investigación exhaustiva realizada sobre distintos entornos de desarrollo integrados (IDEs), que permitió identificar funcionalidades relevantes y orientar las decisiones de diseño. Por otro lado, se incorporaron mejoras en la visualización gráfica en dos dimensiones, permitiendo la coexistencia de funciones y figuras en un mismo gráfico, así como un manejo más flexible del zoom, incluyendo la posibilidad de realizar zoom independiente por eje en aquellos casos en los que resulta conceptualmente adecuado. Estas extensiones amplían las posibilidades de exploración visual y contribuyen a una mejor comprensión de los conceptos matemáticos representados.

Asimismo, se incorporaron mejoras en la navegabilidad y organización del entorno de desarrollo, facilitando la reutilización de funciones definidas en distintos archivos y el acceso directo a sus definiciones. Estas funcionalidades contribuyen a una mayor fluidez en el trabajo con programas más complejos y acercan la experiencia de uso de MateFun a la de otros entornos de desarrollo ampliamente utilizados.

El proyecto también permitió consolidar una experiencia de trabajo sobre un sistema preexistente, enfrentando desafíos asociados a la comprensión de

una base de código heredada, a la integración de nuevas funcionalidades y a la resolución de problemas técnicos derivados del entorno de desarrollo. En este sentido, se logró cumplir con los objetivos planteados inicialmente, aportando mejoras concretas y funcionales que incrementan el valor educativo y la usabilidad del entorno MateFun.

5.2. Trabajo futuro

Si bien el proyecto cumplió con los objetivos propuestos, durante su desarrollo se identificaron diversas líneas de trabajo que podrían abordarse en el futuro para continuar evolucionando el entorno MateFun.

En primer lugar, una posible extensión consiste en mejorar el manejo de errores en tiempo real dentro del editor de código, incorporando detección temprana de errores sintácticos y semánticos, así como un panel centralizado de advertencias y errores. Estas funcionalidades permitirían reducir el tiempo de retroalimentación y facilitarían el aprendizaje, especialmente en usuarios con poca experiencia en programación.

Otra línea de trabajo relevante es la ampliación del sistema de autocompletado y ayuda contextual, incorporando información más detallada sobre las funciones disponibles, sus parámetros y su comportamiento.

En el ámbito de la visualización gráfica, podrían explorarse extensiones orientadas a una mayor interactividad, como la visualización dinámica de variables durante la ejecución, la obtención de valores específicos al interactuar con gráficos tridimensionales o la ejecución parcial de código para facilitar la experimentación.

Finalmente, una línea de trabajo futura de especial interés consiste en evaluar el impacto real de las funcionalidades incorporadas en contextos educativos, mediante pruebas con docentes y estudiantes. Este tipo de evaluación permitiría validar empíricamente las mejoras realizadas y orientar futuras decisiones de diseño en función del uso efectivo del entorno en el aula.

De este modo, el trabajo realizado sienta una base para futuras extensiones de MateFun y para la incorporación de nuevas funcionalidades que continúen fortaleciendo su uso en la enseñanza.

Referencias

Alex Hhh. (2018). *Interactive overlays with the racket plot package.* https://alex-hhh.github.io/2018/02/interactive-overlays-with-the-racket-plot-package.html. (Accessed: 2025-06-24)

BlueJ Team, King's College London. (2025). *Bluej official website and documentation.* https://www.bluej.org/. (Accessed: 2025-05-26)

Cursor AI Team. (2025). *Cursor ide – documentation and features.* https://cursor.sh/docs. (Accessed: 2025-06-16)

Eclipse Foundation. (2025a). *Eclipse ide documentation.* https://www.eclipse.org/documentation/. (Accessed: 2025-05-15)

Eclipse Foundation. (2025b). *Eclipse - Team Development and Collaboration Tools.* https://www.eclipse.org/ide/. (Accessed: 2025-07-26)

Microsoft. (2025). *Visual studio code documentation.* https://code.visualstudio.com/docs. (Accessed: 2025-05-15)

Microsoft Visual Studio Code Team. (2017). *Introducing Visual Studio Live Share.* https://code.visualstudio.com/blogs/2017/11/15/live-share. (Accessed: 2025-08-03)

Plotly Technologies Inc. (2025). *Interactive graphing with plotly.* https://dash.plotly.com/interactive-graphing. (Accessed: 2025-07-03)

Racket Team. (2025). *Racket documentation – plot and drracket.* https://docs.racket-lang.org/plot/intro.html. (Accessed: 2025-05-23)

Replit Docs. (2025). *Multiplayer Mode in Replit.* https://docs.replit.com/replit-app/collaborate. (Accessed: 2025-08-01)

Replit Inc. (2025). *Replit documentation – getting started.* https://docs.replit.com/getting-started/intro-replit. (Accessed: 2025-06-05)

Thonny Project. (2025). *Thonny python ide – official documentation.* https://thonny.org/. (Accessed: 2025-05-24)

Anexos

El presente informe se complementa con dos documentos anexos, en los cuales se desarrollan en mayor detalle aspectos específicos del proyecto que, por su nivel de profundidad, no se incluyen en el cuerpo principal del documento. A continuación, se describen brevemente los contenidos de cada uno de los documentos anexos.

.1. Documento de requerimientos y casos de uso

Este documento anexo presenta la definición detallada de los requerimientos funcionales y no funcionales del sistema, así como la especificación completa de los casos de uso implementados en el marco del proyecto. Para cada caso de uso se describen los actores involucrados, las precondiciones, las postcondiciones, los flujos principales y los flujos alternativos, junto con los diagramas correspondientes.

.2. Documento de arquitectura

Este documento anexo describe la arquitectura del sistema y las decisiones de diseño adoptadas durante el desarrollo del proyecto. En él se detallan los componentes principales del sistema, su organización interna y los mecanismos de integración utilizados para incorporar las nuevas funcionalidades respetando la arquitectura existente.