

Grado Universitario en Ingeniería Informática  
2024-2025

*Trabajo Fin de Grado*

# “Aplicación web para el diseño y ejecución de diagramas de flujo”

---

Daniel Pérez Fernández

Tutor

Álvaro Montero Montes

Leganés, 2025



Esta obra se encuentra sujeta a la licencia Creative Commons **Reconocimiento – No Comercial – Sin Obra Derivada**



## ABSTRACT

Learning to program presents significant challenges for beginners, particularly the need to simultaneously grasp algorithmic logic and the syntax of a textual programming language. This bachelor's thesis aims to reduce this initial difficulty through the design, development, and implementation of an interactive web application that allows students to create, execute, and debug algorithms using standard flowcharts. This approach enables users to focus on computational thinking and problem-solving logic before tackling the syntactic complexity of code.

To achieve this goal, the main development methods included designing an intuitive web interface for the visual creation and editing of flowcharts that represent fundamental algorithmic structures. An execution engine capable of interpreting these diagrams was implemented, allowing for the input of data, the display of output, and the step-by-step tracing of the execution flow directly on the diagram. Basic visual debugging capabilities were also incorporated.

The most important result is a functional web tool that not only allows users to interactively design and run their visual algorithms but also to automatically generate the equivalent source code in programming languages such as Python. This feature is crucial for demonstrating the correspondence between visual and textual representations, thereby easing the transition to traditional programming.

To further support self-learning, the application incorporates an exercises module with automatic evaluation, allowing users to validate their solutions against predefined test cases. Additionally, the project includes real-time collaboration features, seeking to enrich the learning experience and foster a more interactive and mutually supportive environment.

It is concluded that the developed application constitutes a pedagogical resource that addresses the limitations of similar desktop-based tools and offers an environment focused on algorithmic logic. By allowing users to experiment with executable flowcharts and visualize their translation into code, the tool helps to mitigate the initial barriers to learning programming and promotes the development of computational thinking in beginner students.

The repository containing the source code for the project can be found at:  
<https://github.com/MrPoll0/flowming>

**Keywords:** Flowchart, Programming, Learning, Web Application, Computational Thinking



## RESUMEN

El aprendizaje de la programación presenta desafíos significativos para los estudiantes principiantes, destacando la necesidad de asimilar simultáneamente la lógica algorítmica y la sintaxis de un lenguaje de programación textual. El presente Trabajo de Fin de Grado se centra en reducir esta dificultad inicial a través del diseño, desarrollo e implementación de una aplicación web interactiva que permita a los estudiantes crear, ejecutar y depurar algoritmos utilizando diagramas de flujo estándar. Esta aproximación busca que los usuarios se concentren en el pensamiento computacional y la lógica de resolución de problemas antes de enfrentarse a la complejidad sintáctica del código.

Para alcanzar este objetivo, los principales métodos de desarrollo incluyeron el diseño de una interfaz web intuitiva para la creación y edición visual de diagramas de flujo que representan estructuras algorítmicas fundamentales. Se implementó un motor de ejecución capaz de interpretar estos diagramas, permitiendo la introducción de datos de entrada, la visualización de datos de salida y el seguimiento paso a paso del flujo de ejecución sobre el propio diagrama. Asimismo, se incorporaron capacidades básicas de depuración visual.

El resultado más importante es una herramienta web funcional, que permite a los usuarios no solo diseñar y ejecutar interactivamente sus algoritmos visuales, sino también generar automáticamente el código fuente equivalente en lenguajes de programación tales como Python. Esta funcionalidad es crucial para mostrar la correspondencia entre la representación visual y la textual, facilitando así la transición a la programación tradicional.

Para reforzar el aprendizaje autónomo, la aplicación incluye un módulo de ejercicios con evaluación automática, que permite a los usuarios validar sus soluciones frente a casos de prueba predefinidos. Adicionalmente, el proyecto incluye funcionalidades de colaboración en tiempo real, buscando enriquecer la experiencia de aprendizaje y fomentar un entorno más interactivo y de apoyo mutuo.

Se concluye que la aplicación desarrollada constituye un recurso pedagógico que aborda las limitaciones de herramientas similares de escritorio y ofrece un entorno enfocado en la lógica algorítmica. Al permitir experimentar con diagramas de flujo ejecutables y visualizar su traducción a código, la herramienta contribuye a mitigar las barreras iniciales del aprendizaje de la programación y a fomentar el desarrollo del pensamiento computacional en estudiantes principiantes.

El repositorio con el código fuente del proyecto se encuentra en:  
<https://github.com/MrPoll0/flowming>

**Palabras Clave:** Diagrama de Flujo, Programación, Aprendizaje, Aplicación Web, Pensamiento Computacional



## AGRADECIMIENTOS

Me gustaría agradecer a todas las personas que, de una forma u otra, han sido un pilar fundamental durante esta etapa universitaria. En primer lugar, a mi familia, por su incondicional apoyo, su paciencia infinita y por estar siempre a mi lado en toda ocasión. A mis amigos, por haber estado siempre ahí, tanto en las buenas como en las malas, ya sea a 5.000, 500 o a meros kilómetros.

En el plano académico, quiero agradecer a todos los buenos profesores que han dejado una huella en mi formación por su magisterio y dedicación: a Ester Aurora Torrente Orihuela, por sus excelentes clases de Álgebra Lineal; a José Luis Mira Peidro, en Teoría de Autómatas y Lenguajes Formales; y a José Miguel Moreno López, por su clara e inspiradora pasión por la ciberseguridad. Mi agradecimiento también se extiende a la Universidad de Waterloo, por la valiosa oportunidad de aprender de Khuzaima Daudjee en *Distributed Systems*, Mehrdad Pirnia en *Introduction to Machine Learning* y Oliver Schneider en *Human-Computer Interaction*.

Finalmente, quiero también agradecer a mi tutor, Álvaro Montero Montes. Su guía, sus valiosos consejos y su orientación han sido claves para el desarrollo de este Trabajo de Fin de Grado. Gracias por la confianza y el apoyo brindado.



# ÍNDICE DE CONTENIDOS

<b>1</b>	<b>INTRODUCCIÓN .....</b>	<b>1</b>
1.1	Contexto.....	1
1.2	Problemas .....	1
1.3	Objetivos.....	3
1.4	Estructura del documento .....	4
<b>2</b>	<b>ESTADO DEL ARTE .....</b>	<b>7</b>
2.1	Primeros algoritmos.....	7
2.2	Orígenes de la representación visual de algoritmos .....	7
2.3	Pioneros de la interacción gráfica y nacimiento de los VPLs .....	8
2.4	Evolución a diagramas de flujo ejecutables educativos .....	13
2.5	Auge de la programación visual por bloques y transición a la web .....	18
2.6	Situación actual y oportunidades .....	21
<b>3</b>	<b>ANÁLISIS DEL PROYECTO .....</b>	<b>25</b>
3.1	Descripción y origen del problema.....	25
3.2	Requisitos de Usuario .....	25
3.3	Casos de Uso .....	26
3.4	Requisitos de Software .....	28
3.4.1	Requisitos Funcionales .....	28
3.4.2	Requisitos No Funcionales .....	32
3.5	Matrices de Trazabilidad .....	35
3.5.1	Trazabilidad de Requisitos de Usuario a Requisitos de Software .....	35
3.5.2	Trazabilidad de Requisitos de Software a Casos de Uso .....	35
<b>4</b>	<b>PLANIFICACIÓN Y PRESUPUESTO .....</b>	<b>37</b>
4.1	Estimación de esfuerzo .....	37
4.2	Planificación inicial .....	38
4.3	Presupuesto .....	39
4.3.1	Coste de personal (recursos humanos) .....	39
4.3.2	Otros costes .....	41
4.3.3	Presupuesto de venta .....	41
<b>5</b>	<b>DISEÑO DE LA SOLUCIÓN .....</b>	<b>43</b>

5.1	Mockups .....	43
5.1.1	Boceto inicial.....	43
5.1.2	Mockup detallado .....	44
5.2	Identidad visual: logo y nombre .....	50
5.3	Prototipo funcional y evolución del diseño web .....	52
5.3.1	Versión inicial del prototipo.....	52
5.3.2	Diseño final implementado.....	53
<b>6</b>	<b>ARQUITECTURA DEL SISTEMA .....</b>	<b>55</b>
6.1	Arquitectura general .....	55
6.2	Arquitectura del servidor .....	57
6.3	Gestión de código y despliegue continuo (CI/CD).....	57
6.3.1	Integración continua y pruebas automatizadas (CI) .....	57
6.3.2	Despliegue continuo (CD).....	58
<b>7</b>	<b>TECNOLOGÍAS EMPLEADAS Y DETALLES DE LA IMPLEMENTACIÓN.....</b>	<b>59</b>
7.1	Gestión del proyecto y control de versiones.....	59
7.2	Tecnologías empleadas.....	59
7.3	Estructura del proyecto.....	61
7.3.1	Estructura a nivel raíz .....	61
7.3.2	Estructura del directorio “src/” .....	61
7.4	Detalles clave de la implementación .....	63
7.4.1	Procesamiento de expresiones mediante AST .....	63
7.4.2	Sistema de tipado y gestión de variables .....	64
7.4.3	Implementación de listas .....	65
<b>8</b>	<b>EVALUACIÓN CON USUARIOS .....</b>	<b>67</b>
8.1	Objetivos de la evaluación.....	67
8.2	Diseño de las pruebas y recopilación de datos .....	67
8.3	Protocolo de evaluación.....	68
8.4	Resultados.....	68
8.4.1	Resultados cuantitativos .....	69
8.4.2	Resultados cualitativos .....	69
8.5	Discusión de datos .....	70
8.6	Conclusión de la evaluación con usuarios .....	71

<b>9</b>	<b>PLANIFICACIÓN FINAL .....</b>	<b>73</b>
<b>10</b>	<b>MARCO SOCIOECONÓMICO.....</b>	<b>75</b>
10.1	Presupuesto .....	75
10.2	Impacto socioeconómico .....	75
<b>11</b>	<b>MARCO REGULADOR.....</b>	<b>77</b>
11.1	Propiedad intelectual e industrial .....	77
11.2	Privacidad y protección de datos .....	77
11.3	Estándares técnicos.....	78
<b>12</b>	<b>CONCLUSIONES Y LÍNEAS FUTURAS .....</b>	<b>79</b>
12.1	Conclusiones generales.....	79
12.2	Conclusiones personales.....	80
12.3	Líneas futuras .....	81
<b>BIBLIOGRAFÍA .....</b>		<b>85</b>
A.	<b>ANEXO: DOCUMENTO DE REQUISITOS DE USUARIO .....</b>	<b>89</b>
B.	<b>ANEXO: MATRIZ DE TRAZABILIDAD DE REQUISITOS DE USUARIO A REQUISITOS DE SOFTWARE .....</b>	<b>103</b>
C.	<b>ANEXO: MATRIZ DE TRAZABILIDAD DE REQUISITOS DE SOFTWARE A CASOS DE USO .....</b>	<b>107</b>
D.	<b>ANEXO: DESGLOSE COMPLETO DE LA ESTIMACIÓN UCP .....</b>	<b>109</b>
D.1.	Valoración de los casos de uso .....	109
D.2.	Cálculos detallados .....	110
E.	<b>ANEXO: DOCUMENTO DE EVALUACIÓN ASÍNCRONA .....</b>	<b>111</b>
F.	<b>ANEXO: CUESTIONARIO DE LA EVALUACIÓN EN GOOGLE FORMS</b>	<b>116</b>
G.	<b>ANEXO: FORMULARIO DE CONSENTIMIENTO INFORMADO.....</b>	<b>120</b>
H.	<b>RESULTADOS DESGLOSADOS DEL CUESTIONARIO DE USABILIDAD (SUS).....</b>	<b>122</b>
I.	<b>ANEXO: DECLARACIÓN DE IA .....</b>	<b>125</b>



## ÍNDICE DE FIGURAS

Fig. 2.1. Sketchpad de Ivan Sutherland. [15], [16].....	8
Fig. 2.2. Símbolos del diagrama de flujo de GRAIL. [16] .....	9
Fig. 2.3. Demostración visual de la prueba de un caso base para una función factorial en Pygmalion. [16] .....	10
Fig. 2.4. Programa en Prograph para calcular la hipotenusa de un triángulo rectángulo. [49] .....	11
Fig. 2.5. Ejemplo de un programa en LabVIEW. [45] .....	11
Fig. 2.6. Programa en LogoBlocks para la evasión de paredes en el control de un coche LEGO. [1].....	12
Fig. 2.7. Animación básica de un patinador de hielo en Alice. [41] .....	13
Fig. 2.8. Algoritmo de búsqueda Dijkstra en DRAKON. [42].....	14
Fig. 2.9. Programa "Hello, world!" en Visual Logic. [39] .....	15
Fig. 2.10. Ejemplo de programa en Raptor. [47].....	16
Fig. 2.11. Ejemplo de programa en Flowgorithm. [48].....	17
Fig. 2.12. Ejemplo de programa en Scratch. [38].....	19
Fig. 2.13. Ejemplo de programa para convertir temperatura de Fahrenheit a Celsius en Blockly. [43].....	20
Fig. 2.14. Evolución cronológica de herramientas de programación visual.....	22
Fig. 3.1. Diagrama de casos de uso. ....	27
Fig. 4.1. Diagrama de Gantt del proyecto, incluyendo las fases de análisis, diseño, desarrollo, pruebas y la elaboración de la memoria. ....	39
Fig. 5.1. Boceto inicial ( <i>wireframe</i> ) de la distribución de la interfaz.....	43
Fig. 5.2. Mockup detallado de la interfaz de la aplicación.....	44
Fig. 5.3. Mockup: Área de trabajo principal. ....	46
Fig. 5.4. Mockup: Paleta de bloques. ....	47
Fig. 5.5. Mockup: Panel de comandos. ....	47
Fig. 5.6. Mockup: Panel de depuración, mostrando los apartados: variables y cadena de ejecución. ....	48
Fig. 5.7. Mockup: Panel de detalles (1), mostrando las pantallas para los bloques de declaración y entrada. ....	48
Fig. 5.8. Mockup: Panel de detalles (2), mostrando la pantalla para el bloque de asignación y la pantalla general de propiedades de bloques. ....	48

Fig. 5.9. Mockup: Panel de detalles (3), mostrando la pantalla para los ajustes de sistema.	49
Fig. 5.10. Mockup: Panel de código.....	49
Fig. 5.11. Mockup: Panel de ejercicios, mostrando el listado de ejercicios y los detalles de un ejercicio seleccionado.....	49
Fig. 5.12. Mockup: Panel de colaboración (1), mostrando la pantalla inicial y la pantalla de una sala desde la perspectiva del creador .....	50
Fig. 5.13. Mockup: Panel de colaboración (2), mostrando la pantalla para unirse a una sala y la pantalla de una sala desde la perspectiva de un participante.....	50
Fig. 5.14. Boceto inicial del logotipo. ....	51
Fig. 5.15. Logotipo final de la aplicación.....	51
Fig. 5.16. Favicon (icono) de la aplicación. ....	52
Fig. 5.17. Vista de la versión inicial del prototipo web.....	52
Fig. 5.18. Interfaz de la versión final del prototipo web. ....	53
Fig. 6.1. Diagrama de la arquitectura general del sistema.....	56
Fig. 8.1. Desglose del cálculo de la puntuación SUS por participante.....	69
Fig. 9.1. Diagrama de Gantt final que refleja la ejecución real del proyecto. ....	73

## **ÍNDICE DE TABLAS**

Tabla 2.1. Comparativa de herramientas de programación visual. ....	21
Tabla 3.1. Requisitos funcionales.....	28
Tabla 3.2. Requisitos no funcionales.....	32
Tabla 4.1. Roles y tarifas horarias estimadas [56–59].....	40
Tabla 4.2. Dedicación y coste por rol.....	40
Tabla 4.3. Desglose estimado para infraestructuras y equipos.....	41
Tabla 4.4. Desglose del presupuesto de venta .....	42



## LISTA DE ABREVIATURAS

<b>ACM</b>	Association for Computing Machinery
<b>ANECA</b>	Agencia Nacional de Evaluación de la Calidad y Acreditación
<b>ANSI</b>	Instituto Nacional Estadounidense de Estándares ( <i>American National Standards Institute</i> )
<b>API</b>	Interfaz de Programación de Aplicaciones ( <i>Application Programming Interface</i> )
<b>AST</b>	Árbol de Sintaxis Abstracta ( <i>Abstract Syntax Tree</i> )
<b>CC</b>	Creative Commons
<b>CD</b>	Despliegue Continuo ( <i>Continuous Deployment</i> )
<b>CI</b>	Integración Continua ( <i>Continuous Integration</i> )
<b>CI/CD</b>	Integración y Despliegue Continuo
<b>CLA</b>	Acuerdo de Licencia de Colaborador ( <i>Contributor License Agreement</i> )
<b>CRDT</b>	Estructura de Datos Replicada sin Conflictos ( <i>Conflict-free Replicated Data Type</i> )
<b>CSS</b>	Hojas de Estilo en Cascada ( <i>Cascading Style Sheets</i> )
<b>CU</b>	Caso de Uso
<b>DRAKON</b>	Lenguaje Algorítmico Ruso Amigable que Proporciona Claridad (acrónimo ruso)
<b>ECTS</b>	Sistema Europeo de Transferencia y Acumulación de Créditos ( <i>European Credit Transfer and Accumulation System</i> )
<b>EF</b>	Factor Ambiental ( <i>Environmental Factor</i> )
<b>EUR-ACE</b>	<i>European Network for Accreditation of Engineering Education</i>
<b>FR</b>	Requisito Funcional ( <i>Functional Requirement</i> )
<b>GDPR</b>	Reglamento General de Protección de Datos ( <i>General Data Protection Regulation</i> )
<b>GRAIL</b>	<i>GRApical Input Language</i>
<b>GUI</b>	Interfaz Gráfica de Usuario ( <i>Graphical User Interface</i> )
<b>HTML</b>	Lenguaje de Marcado de Hipertexto ( <i>HyperText Markup Language</i> )
<b>IA</b>	Inteligencia Artificial
<b>IBM</b>	International Business Machines
<b>ISO</b>	Organización Internacional de Normalización ( <i>International Organization for Standardization</i> )
<b>LabVIEW</b>	<i>Laboratory Virtual Instrument Engineering Workbench</i>

<b>LXC</b>	Contenedores de Linux ( <i>Linux Containers</i> )
<b>MIT</b>	Instituto de Tecnología de Massachusetts ( <i>Massachusetts Institute of Technology</i> )
<b>NFR</b>	Requisito No Funcional ( <i>Non-Functional Requirement</i> )
<b>ODS</b>	Objetivo de Desarrollo Sostenible
<b>OT</b>	Transformación Operacional ( <i>Operational Transformation</i> )
<b>P2P</b>	Red entre pares ( <i>Peer-to-Peer</i> )
<b>PBE</b>	Programación por Ejemplo ( <i>Programming by Example</i> )
<b>PHM</b>	Factor de Productividad ( <i>Person Hours Multiplier</i> )
<b>PM2</b>	Process Manager 2
<b>QA</b>	Garantía de Calidad ( <i>Quality Assurance</i> )
<b>RAPTOR</b>	<i>Rapid Algorithmic Prototyping Tool for Ordered Reasoning</i>
<b>RRHH</b>	<i>Recursos Humanos</i>
<b>SPA</b>	Aplicación de Página Única ( <i>Single Page Application</i> )
<b>SUS</b>	Escala de Usabilidad del Sistema ( <i>System Usability Scale</i> )
<b>TCF</b>	Factor de Complejidad Técnica ( <i>Technical Complexity Factor</i> )
<b>TFG</b>	Trabajo de Fin de Grado
<b>UCP</b>	Puntos de Caso de Uso ( <i>Use Case Points</i> )
<b>UCR</b>	Requisito de Restricción del Usuario ( <i>User Constraint Requirement</i> )
<b>UI</b>	Interfaz de Usuario ( <i>User Interface</i> )
<b>UR</b>	Requisito de Capacidad del Usuario ( <i>User Requirement</i> )
<b>USMA</b>	Academia Militar de los Estados Unidos ( <i>U.S. Military Academy</i> )
<b>UX</b>	Experiencia de Usuario ( <i>User Experience</i> )
<b>VI</b>	Instrumento Virtual ( <i>Virtual Instrument</i> )
<b>VPL</b>	Lenguaje de Programación Visual ( <i>Visual Programming Language</i> )
<b>VPS</b>	Servidor Virtual Privado ( <i>Virtual Private Server</i> )
<b>W3C</b>	World Wide Web Consortium
<b>WCAG</b>	Pautas de Accesibilidad para el Contenido Web ( <i>Web Content Accessibility Guidelines</i> )
<b>WebRTC</b>	Comunicación en Tiempo Real para la Web ( <i>Web Real-Time Communication</i> )
<b>YAML</b>	YAML no es un lenguaje de marcado ( <i>YAML Ain't Markup Language</i> )



# 1 INTRODUCCIÓN

## 1.1 Contexto

La programación se ha consolidado como una competencia esencial en el panorama tecnológico actual, de tal manera que su importancia y utilidad transcenden el ámbito informático. Su relevancia se destaca por la introducción paulatina en etapas educativas previas a la universidad, como la educación secundaria [23], reconociéndola como una herramienta clave para la innovación y la resolución de problemas en una sociedad cada vez más digitalizada.

En el marco universitario, la programación es una asignatura de formación básica que, si bien no es obligatoria que se incluya en los planes de estudios de las ingenierías por propia exigencia de la Agencia Nacional de Evaluación de la Calidad y Acreditación (ANECA), la mayoría de estos la implementan. En cualquier caso, algunos prestigiosos sellos de acreditaciones de educación superior en ingenierías como el EUR-ACE (*European Network for Accreditation of Engineering Education*) destacan que los estudiantes deben adquirir competencias en el uso de software para el contexto de la ingeniería. Por ello, al inicio del primer curso de cualquier grado de este ámbito, el estudiante se debe enfrentar a la adquisición de los conocimientos y habilidades que programar implica. Esto requiere que el estudiante desarrolle nuevas maneras de pensar, basadas fundamentalmente en el pensamiento computacional.

## 1.2 Problemas

Aprender a programar presenta desafíos considerables. Una dificultad clave para los estudiantes que se inician en la programación radica en la necesidad de asimilar simultáneamente dos conceptos complejos: la lógica algorítmica y la sintaxis específica de un lenguaje de programación [16]. La lógica algorítmica consiste en la capacidad de descomponer un problema, diseñar una secuencia de pasos (algoritmo) para resolverlo, y comprender las estructuras de control fundamentales (secuencia, selección, iteración) y el manejo de datos. Por otro lado, la sintaxis de un lenguaje de programación define las reglas gramaticales y el vocabulario específico que un lenguaje de programación particular (como Python, Java, C, etc.) emplea para expresar esa lógica de manera que el ordenador pueda entenderlo y ejecutarlo.

La sintaxis, con sus reglas a menudo poco intuitivas para un principiante, puede convertirse en un obstáculo significativo que desvía la atención del estudiante del objetivo fundamental: aprender a pensar computacionalmente. La frustración causada por errores sintácticos recurrentes es una causa frecuente de desmotivación y abandono temprano en los cursos introductorios [11].

Como solución a este problema, han surgido los Lenguajes de Programación Visual (VPL – *Visual Programming Language*). Estos lenguajes permiten a los usuarios construir programas utilizando elementos gráficos (como bloques, iconos o diagramas de

flujo) en vez de código textual. El objetivo de los VPL es reducir la carga cognitiva asociada a la sintaxis, permitiendo a las personas poco experimentadas centrarse en los aspectos conceptuales y lógicos de la programación [25].

Existen diversos paradigmas dentro del espectro de los VPL, como la programación visual basada en bloques (popularizada por herramientas como Scratch o Blockly) o enfoques basados en flujos de datos. Otro ejemplo fundamental, especialmente arraigado en la enseñanza inicial de los algoritmos y el diseño del software, es el uso de diagramas de flujo [25]. Estos diagramas ofrecen una representación gráfica estándar y ampliamente comprendida de la secuencia de operaciones, las estructuras de control y el flujo general de un algoritmo, sirviendo como una excelente forma de conceptualizar la solución computacional antes de abordar la complejidad sintáctica de un lenguaje textual.

Si bien existen numerosas herramientas que permiten dibujar diagramas de flujo<sup>1</sup>, una limitación significativa para el aprendizaje es que muchas de estas herramientas dan como resultado un diagrama estático. Es decir, permiten representar la lógica, pero no permiten al estudiante validar su corrección o comprender el comportamiento del algoritmo plasmado en el diagrama. Sin embargo, para un aprendizaje efectivo, es crucial poder observar cómo el algoritmo procesa datos, sigue decisiones programadas y produce resultados. La capacidad de ejecutar el diagrama, introducir entradas, seguir el flujo paso a paso y ver las salidas correspondientes proporciona una retroalimentación inmediata que facilita la depuración de errores lógicos y refuerza la comprensión de los conceptos fundamentales.

Por otro lado, aunque los VPL basados en bloques son muy populares, a veces se plantean dudas sobre su efectividad en la transferencia de conocimiento a la programación textual tradicional [35] que los estudiantes de ingeniería encontrarán posteriormente. Los diagramas de flujo, al ser una representación más cercana a la estructura secuencial y de control de código, podrían mejorar esta transición. Además, las herramientas que sí permiten la ejecución de diagramas de flujo suelen ser aplicaciones de escritorio [47, 48], lo que limita su accesibilidad al requerir instalaciones específicas y depender del sistema operativo.

Por lo tanto, la motivación central de este proyecto es abordar estas limitaciones y oportunidades: desarrollar una aplicación web que integre de forma fluida tanto el diseño intuitivo de diagramas de flujo estándar como su ejecución y depuración interactiva. La elección de una plataforma web busca maximizar la accesibilidad, mientras que el enfoque en diagramas de flujo ejecutables busca combinar la claridad visual con una estructura conceptualmente cercana a la programación textual. Esta conexión se verá reforzada por la capacidad de la herramienta de generar automáticamente el código fuente correspondiente en conocidos lenguajes de programación (como Python), facilitando así el aprendizaje inicial y la posterior transición.

---

<sup>1</sup> <https://draw.io>, <https://lucidchart.com>, <https://miro.com>, <https://canva.com>

Adicionalmente, la integración de aspectos colaborativos en entornos de aprendizaje basados en programación visual ha demostrado ser beneficiosa. Estudios [31] evidencian que la combinación de una herramienta de programación visual combinada con una plataforma que facilita la comunicación y colaboración en tiempo real mejora significativamente el desarrollo del pensamiento computacional, las habilidades de resolución de problemas y el rendimiento general de los estudiantes al crear un entorno de aprendizaje más interactivo y de apoyo mutuo.

En resumen, se busca crear una herramienta que aproveche las ventajas pedagógicas de la representación visual mediante diagramas de flujo donde los estudiantes puedan centrarse en la lógica algorítmica, experimentar sus diseños, recibir *feedback* inmediato y visualizar su equivalencia en código textual, mitigando así las dificultades iniciales asociadas al aprendizaje de la programación, superando algunas de las barreras prácticas de las soluciones existentes y sentando las bases para un entorno de aprendizaje potencialmente más completo, colaborativo e interactivo.

### 1.3 Objetivos

El presente Trabajo de Fin de Grado tiene como objetivo último reducir la dificultad del proceso de aprendizaje de la programación. Específicamente, se busca **facilitar el desarrollo del pensamiento computacional por medio de una herramienta que permita a los estudiantes crear y ejecutar algoritmos usando diagramas de flujo**. Esto permitirá a los estudiantes centrarse en la lógica algorítmica y la resolución de problemas antes de lidiar con las reglas estrictas del código. Asimismo, se aspira a suavizar la transición a la programación textual, reduciendo la barrera impuesta por la sintaxis. Para alcanzar este fin, se diseñará, desarrollará e implementará una aplicación web interactiva que permita a estudiantes principiantes crear, ejecutar y depurar diagramas de flujo y generar el código fuente asociado.

Para alcanzar el objetivo de este Trabajo de Fin de Grado, se establecen los siguientes objetivos específicos del proyecto:

- Diseñar una interfaz web intuitiva que facilite la creación y edición de diagramas de flujo mediante operaciones visuales que representen las estructuras algorítmicas fundamentales (secuencia, asignación, entrada/salida, selección, iteración).
- Implementar un motor de ejecución capaz de interpretar los diagramas de flujo creados por el usuario, simulando el comportamiento de un programa real.
- Desarrollar funcionalidades interactivas para la ejecución, incluyendo:
  - La capacidad de introducir datos de entrada durante la ejecución.
  - La visualización clara de los datos de salida generados por el diagrama.
  - La visualización paso a paso del flujo de ejecución sobre el propio diagrama.

- Incorporar capacidades básicas de depuración visual, permitiendo al estudiante identificar y comprender errores lógicos en su diagrama durante la ejecución.
- Implementar una funcionalidad de generación automática de código fuente (seleccionable por el usuario) equivalente al diagrama de flujo diseñado, mostrando la correspondencia entre los elementos visuales y las estructuras sintácticas textuales.
- Asegurar la accesibilidad y portabilidad de la herramienta mediante su desarrollo como aplicación web estándar, accesible desde navegadores modernos sin necesidad de instalación local.

Adicionalmente, se plantean los siguientes objetivos secundarios, cuya consecución dependerá de la viabilidad técnica y el tiempo disponible durante el desarrollo del proyecto:

- Implementar funcionalidades de colaboración en tiempo real que permitan a varios usuarios visualizar y editar el mismo diagrama de forma concurrente y participar en videoconferencias.
- Diseñar e implementar un módulo de “Banco de problemas” que ofrezca a los usuarios una colección de ejercicios de programación para resolver utilizando la herramienta, con la opción de validar su corrección respecto a la solución existente, para que así los usuarios puedan probar sus conocimientos de una forma sencilla.

Por otra parte, este proyecto persigue los siguientes objetivos académicos:

- Desarrollar un proyecto de software completo, abarcando desde la concepción y análisis de requisitos hasta el diseño, implementación y pruebas.
- Mejorar las habilidades de resolución de problemas, gestión del tiempo y redacción de documentación técnica.
- Aplicar y consolidar los conocimientos adquiridos durante el grado en áreas como ingeniería de software, desarrollo web, diseño de interfaces y algoritmos.

## 1.4 Estructura del documento

A lo largo de este capítulo se ha presentado el contexto en el que se enmarca este Trabajo de Fin de Grado, así como los problemas identificados y el objetivo que se persigue. El resto del documento se estructura de la siguiente manera.

- **Estado del Arte:** Se realiza un recorrido por la evolución de las herramientas de programación visual, desde los primeros algoritmos hasta las soluciones

contemporáneas, analizando sus contribuciones y limitaciones para justificar la oportunidad del presente proyecto.

- **Análisis del Proyecto:** Se detallarán los requisitos de usuario y de software, los casos de uso que definen la interacción con el sistema y las matrices de trazabilidad que garantizan la coherencia del desarrollo.
- **Planificación y Presupuesto:** Se presentará la estimación del esfuerzo mediante la metodología de Puntos de Casos de Uso, la planificación temporal del proyecto en un diagrama de Gantt y el presupuesto teórico detallado.
- **Diseño de la Solución:** Se expondrá el proceso de diseño de la interfaz y la experiencia de usuario, desde los bocetos iniciales y la identidad visual hasta el prototipo funcional implementado.
- **Arquitectura del Sistema:** Se describirá la arquitectura de alto nivel del sistema, incluyendo los componentes de *frontend* y *backend*, la arquitectura de colaboración *Peer-to-Peer* y el flujo de integración y despliegue continuo (CI/CD).
- **Tecnologías y Detalles de la Implementación:** Se detallarán las tecnologías seleccionadas para el desarrollo y las decisiones clave de implementación de los componentes más complejos, como el procesador de expresiones mediante AST y el motor de ejecución.
- **Evaluación con Usuarios:** Se presentará la metodología seguida para las pruebas de usabilidad, los instrumentos de recopilación de datos utilizados y los resultados obtenidos de la evaluación con usuarios externos.
- **Planificación Final:** Se comparará la planificación inicial con la ejecución real del proyecto, analizando las desviaciones y las causas de las mismas.
- **Marco Socioeconómico:** Se analizará el impacto económico y social del proyecto, incluyendo el presupuesto y su alineación con los Objetivos de Desarrollo Sostenible (ODS).
- **Marco Regulador:** Se detallarán las normativas, licencias (propiedad intelectual, software de terceros) y estándares técnicos (ISO, WCAG) aplicables al desarrollo y despliegue de la aplicación.
- **Conclusiones y Líneas Futuras:** Se resumirán las conclusiones generales y personales del trabajo, se evaluará el grado de cumplimiento de los objetivos y se propondrán posibles mejoras y trabajos futuros.
- **Bibliografía:** Se listarán todas las referencias bibliográficas citadas a lo largo del documento con el estilo ACM.
- **Anexos:** Se incluirán documentos complementarios relevantes.



## 2 ESTADO DEL ARTE

El objetivo de este capítulo es identificar las herramientas y enfoques utilizados para facilitar el aprendizaje de la programación, con un énfasis particular en las representaciones visuales y, específicamente, los diagramas de flujo. Este análisis cronológico explorará la evolución de las soluciones propuestas para mitigar esta dificultad, desde los orígenes de la representación algorítmica hasta las herramientas contemporáneas, identificando sus contribuciones, limitaciones y oportunidades que justifican el desarrollo de la solución propuesta en este Trabajo de Fin de Grado.

### 2.1 Primeros algoritmos

Si bien la idea de secuencias de pasos para resolver problemas es antigua, la conceptualización de algoritmos para máquinas marca un punto de inflexión. En este ámbito, destaca el trabajo de Augusta Ada King, condesa de Lovelace, en el siglo XIX [14, 22]. En 1843, al traducir el artículo de Luigi Menabrea sobre la Máquina Analítica de Charles Babbage, Ada Lovelace no solo facilitó su acceso al público inglés, sino que añadió un extenso conjunto de notas que triplicaban en longitud al texto original. Dentro de estas, la “Nota G” es especialmente célebre, ya que contiene lo que muchos consideran el primer algoritmo complejo diseñado explícitamente para ser ejecutado por una máquina: un método detallado para que la Máquina Analítica calculara los números de Bernoulli. Su contribución fue fundamental, demostrando una profunda comprensión de las capacidades de la máquina, más allá de la mera capacidad de cálculo [14, 22].

Ada Lovelace describió este algoritmo mediante una combinación de texto explicativo riguroso y el uso de diagramas o tablas que detallaban la secuencia de operaciones, las variables implicadas, los registros de la máquina y los resultados intermedios y finales [22]. Su programa para los números Bernoulli incorporaba conceptos algorítmicos avanzados para la época, como bucles (repetición de secuencias de instrucciones) y bifurcaciones (toma de decisiones), fundamentales en la computación moderna. Más allá del cálculo numérico, Ada Lovelace tuvo la visión de que la Máquina Analítica podría operar sobre otros tipos de símbolos, como notas musicales, anticipando la versatilidad de los ordenadores actuales [14, 22].

### 2.2 Orígenes de la representación visual de algoritmos

La representación visual de la lógica de los programas precede a la popularización de los ordenadores personales. Fue por primera vez en la década de 1940 que Herman Goldstine y John von Neumann adaptaron y extendieron el concepto de diagrama de flujo al ámbito de la programación [10]. La razón principal fue la necesidad de gestionar la complejidad inherente al flujo de control dinámico de los programas (saltos, bucles y automodificación del código), que iba más allá de la simple traducción de fórmulas matemáticas. Estos diagramas de flujo fueron concebidos como una herramienta conceptual esencial para visualizar cómo interactuaban las diversas partes de un programa

antes de su codificación detallada, alegando la justificación bajo su propia experiencia en la codificación de problemas reales. Utilizaban grafos etiquetados para representar el flujo dinámico de control, mostrando la secuencia en la que la computadora ejecutaría las órdenes y cómo estas afectarían a los valores de las variables almacenadas [10].

Durante las décadas de 1950 y 1960, los diagramas de flujo se consolidaron como una herramienta estándar y fundamental para el diseño y desarrollo de algoritmos antes de su codificación manual. Empresas como IBM jugaron un papel en su estandarización, promoviendo un conjunto de símbolos para representar operaciones, decisiones y el flujo de control [7]. En la década de 1960, el Instituto Nacional Estadounidense de Estándares (ANSI) estableció el estándar para los diagramas de flujo y sus símbolos [36]. Este estándar fue precursor del actual estándar, ISO 5807, publicado por la Organización Internacional de Normalización (ISO) por primera vez en 1985 [37].

### 2.3 Pioneros de la interacción gráfica y nacimiento de los VPLs

El camino desde los diagramas de flujo estáticos hacia entornos de programación más interactivos y visuales fue pavimentado por visionarios que exploraron nuevas formas de interacción persona-ordenador y representación del software. Un hito fundamental en la década de 1960 fue Sketchpad (1963), desarrollado por Ivan Sutherland [34]. Sketchpad fue revolucionario dado que demostró por primera vez la viabilidad y el poder de la interacción gráfica directa con un ordenador mediante un lápiz óptico. Tal y como se puede apreciar en la Figura 2.1, donde el propio Sutherland interactúa con el sistema, los usuarios podían dibujar y manipular figuras geométricas directamente en la pantalla y el sistema entendía restricciones y relaciones entre ellas.

Este proyecto sentó las bases para todas las futuras interfaces gráficas de usuario (GUI – *Graphical User Interface*) y mostró cómo los elementos gráficos podían ser una forma de comunicación bidireccional con la máquina, influyendo indirectamente en el pensamiento sobre cómo se podrían “construir” programas visualmente.

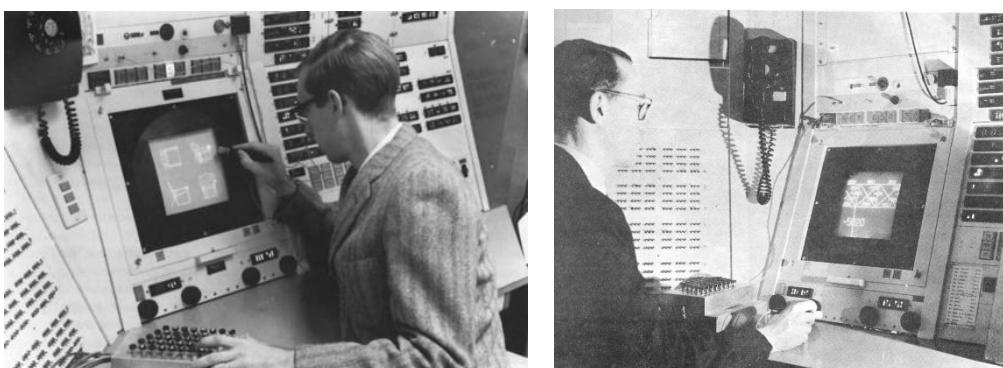


Fig. 2.1. Sketchpad de Ivan Sutherland. [15], [16]

A partir de las décadas de 1970, la investigación y desarrollo en este campo continuaron, dando lugar a la emergencia de los primeros Lenguajes de Programación

Visual (VPLs). El objetivo primordial de los VPLs es reducir la carga cognitiva asociada a la sintaxis textual, permitiendo a los usuarios, especialmente a las personas poco experimentadas, centrarse en los aspectos conceptuales y lógicos de la programación [25]. Durante este periodo, se exploraron diversos paradigmas visuales:

### **GRAIL (1969)**

Desarrollado por RAND Corporation, GRAIL (*GRApical Input Language*) fue uno de los primeros sistemas que utilizaba una Tablet RAND y un lápiz óptico para permitir la creación de diagramas de flujo que luego podían ser ejecutados [8]. Aunque experimental y limitado por la tecnología de la época, GRAIL es significativo ya que conectaba directamente la idea de los diagramas de flujo con la interacción gráfica y la ejecución. Representa uno de los primeros intentos de hacer que los diagramas de flujo fueran más que una simple herramienta de diseño estático. Para lograrlo, el sistema definía su propio léxico visual con un conjunto de símbolos específicos para representar procesos, decisiones e incluso operaciones de sincronización, como se ilustra en la Figura 2.2.

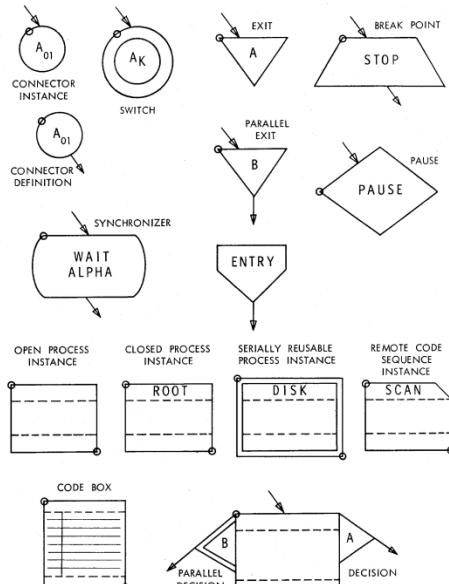


Fig. 2.2. Símbolos del diagrama de flujo de GRAIL. [16]

### **Pygmalion (1975)**

Creado por David Canfield Smith como parte de su tesis doctoral en la Universidad de Stanford, Pygmalion introdujo el concepto de “programación por ejemplo” (PBE – *Programming by Example*) [33]. El programador demostraba las operaciones deseadas directamente sobre representaciones visuales de los datos (iconos) y el sistema infería el programa a partir de estas acciones. La Figura 2.3, por ejemplo, ilustra este enfoque

mostrando la representación visual del caso base para una función factorial, donde la lógica se construye y prueba de forma interactiva. Aunque su enfoque no era estrictamente el de diagramas de flujo, su énfasis en la representación visual y la interacción directa fue significativo para el campo de los VPLs.

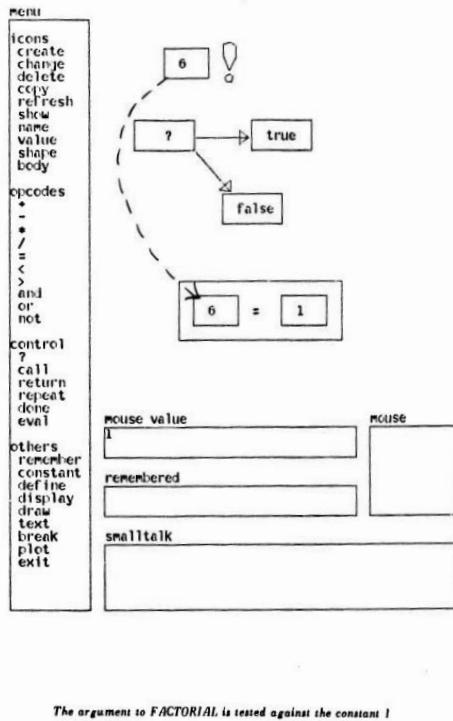


Fig. 2.3. Demostración visual de la prueba de un caso base para una función factorial en Pygmalion. [16]

### **Prograph (1983)**

Concebido en 1983 por Tomasz Pietrzykowski y Philip T. Cox, Prograph fue uno de los primeros Lenguajes de Programación Visual que alcanzó cierto uso comercial, especialmente en la plataforma Macintosh [15]. Prograph se diseñó como un lenguaje funcional y orientado al flujo de datos (*dataflow*), donde los programas (*pictographs*) se construían gráficamente con redes de “cajas” (que representaban operaciones con entradas en la parte superior y salidas en la inferior) y “cables” que simbolizaban el flujo de información entre ellas [21]. La Figura 2.4 ilustra este modelo con un programa que calcula la hipotenusa de un triángulo rectángulo, donde se observa cómo los valores de entrada fluyen a través de una secuencia de operaciones hasta producir el resultado final.

Una característica distintiva de Prograph era la ausencia total de variables en el sentido tradicional, yendo más allá de la simple no declaración, lo que junto a su naturaleza visual buscaba reducir la carga cognitiva asociada a la programación textual. A pesar de que su desarrollo comercial principal cesó a mediados de la década de 1990, Prograph demostró la viabilidad de un VPL completo para aplicaciones del mundo real.

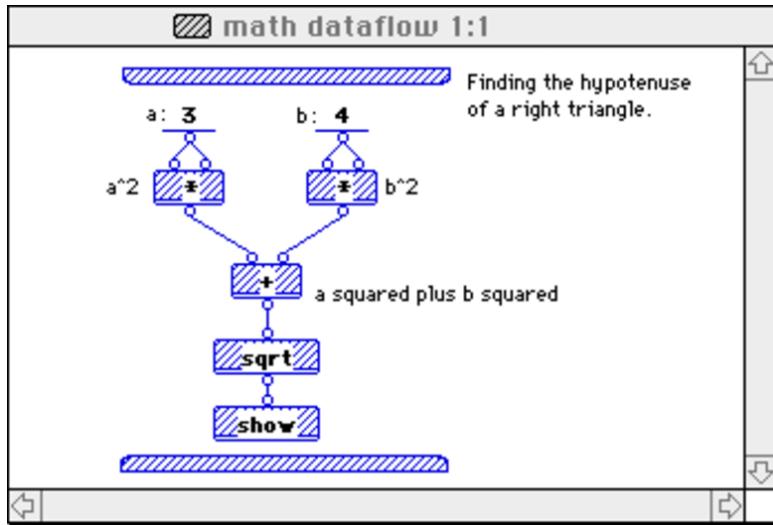


Fig. 2.4. Programa en Graph para calcular la hipotenusa de un triángulo rectángulo. [49]

### **LabVIEW (1986)**

Desarrollado por National Instruments y lanzado en octubre de 1986, LabVIEW (*Laboratory Virtual Instrument Engineering Workbench*) es un lenguaje de programación visual y un entorno de desarrollo diseñado con el objetivo de ayudar a ingenieros y científicos a automatizar sistemas de prueba y medida, sin necesidad de ser programadores expertos [17, 50]. Se basa en el paradigma de flujo de datos (*dataflow*), donde los programas, llamados “Instrumentos Virtuales” (*VI – Virtual Instrument*), se construyen conectando iconos gráficos (que actúan como nodos que representan funciones u operaciones) con “cables” que simbolizan el flujo de datos entre ellos.

Una innovación clave de LabVIEW fue la integración de estructuras de control de flujo como elementos gráficos directamente en el diagrama de flujo de datos. Esto permite que los diagramas no solo representan la lógica, sino que sean directamente ejecutables, ofreciendo retroalimentación inmediata sobre el comportamiento del programa. La Figura 2.5 muestra un ejemplo de un Instrumento Virtual, donde se aprecia la integración de estructuras de control (el gran recuadro) dentro del propio flujo de datos.

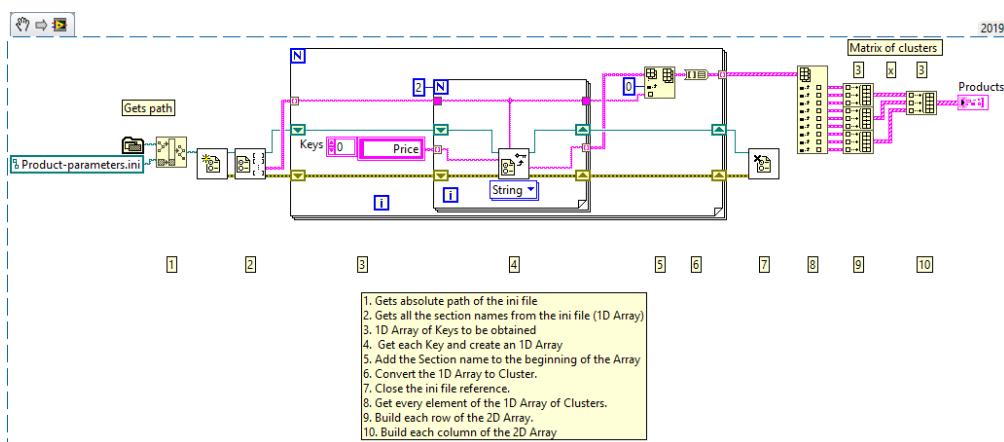


Fig. 2.5. Ejemplo de un programa en LabVIEW. [45]

## *LogoBlocks (1996)*

Desarrollado por Andrew Begel en el MIT Media Laboratory, bajo la supervisión de Mitchel Resnick, LogoBlocks es un lenguaje de programación gráfica diseñado específicamente para el “Programmable Brick”, un pequeño ordenador que se acopla a las creaciones de LEGO para controlar motores y leer sensores [1]. Se concibió como una variante gráfica del lenguaje BrickLogo (una extensión del lenguaje de programación Logo, conocido por su enfoque construcciónista del aprendizaje y el uso de la “tortuga” programable [26]). En lugar de código textual, los usuarios, principalmente niños, construyen programas arrastrando y conectando bloques de colores y formas distintas que representan comandos (acciones, sensores, procedimientos, variables).

Esta metáfora de “piezas de puzzle” que se encajan para construir la lógica, claramente visible en el programa para la evasión de paredes de la Figura 2.6, fue fundamental para su objetivo de ofrecer una alternativa más intuitiva y accesible al lenguaje textual BrickLogo, reduciendo la barrera de entrada a la programación para niños pequeños y permitiéndoles dar vida a sus construcciones de LEGO de manera lúdica. Este enfoque influiría en el desarrollo posterior de entornos de programación basados en bloques.

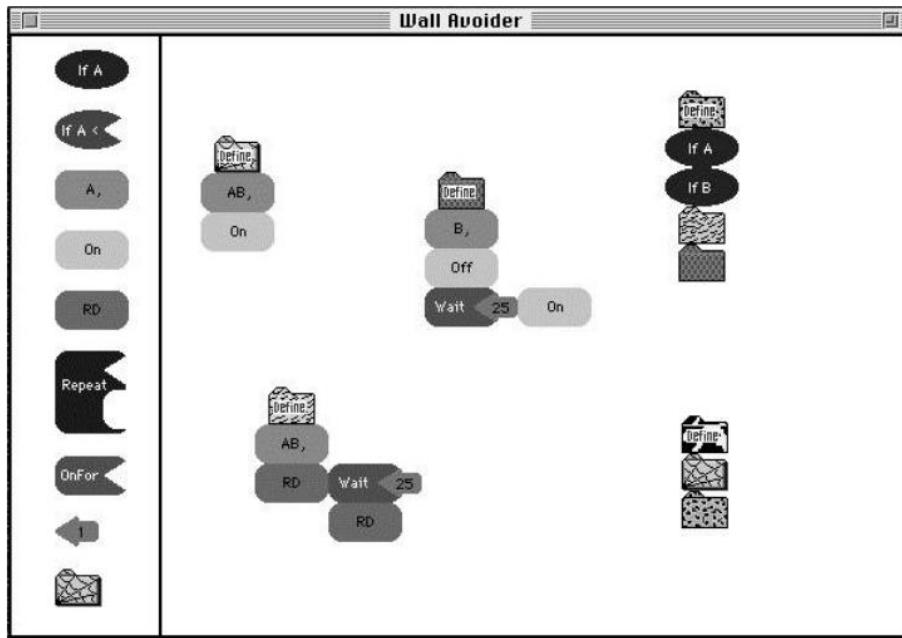


Fig. 2.6. Programa en LogoBlocks para la evasión de paredes en el control de un coche LEGO. [1]

## *Alice (1999)*

Desarrollado por el Stage 3 Research Group en la Universidad de Carnegie Mellon, bajo la dirección de Randy Pausch, Alice es un entorno de programación 3D interactivo [6, 12, 51]. Su objetivo fundamental era hacer accesible la creación de animaciones 3D, mundos interactivos y narrativas digitales a principiantes, especialmente a aquellos sin experiencia previa en la programación. Evolucionando hacia interfaces cada vez más

visuales, Alice utiliza una interfaz gráfica donde los usuarios manipulan objetos 3D y construyen programas arrastrando y soltando bloques de instrucciones que representan acciones, estructuras de control y eventos [12]. Estos bloques se encajan para formar scripts que controlan el comportamiento de los objetos en el mundo virtual. La Figura 2.7 captura la esencia de esta propuesta, mostrando cómo la interfaz integra la representación 3D de la escena (un patinador de hielo) con el editor de *scripts* basado en bloques, permitiendo al usuario ver la conexión directa entre el código visual y el comportamiento del programa.

El enfoque de Alice en la visualización del comportamiento del programa y en la abstracción de la sintaxis compleja comparte el espíritu de los VPLs que buscan facilitar la comprensión de los conceptos de programación. Su influencia ha sido notable en la educación informática, demostrando el potencial de los entornos 3D y la programación basada en objetos visuales para la enseñanza introductoria.

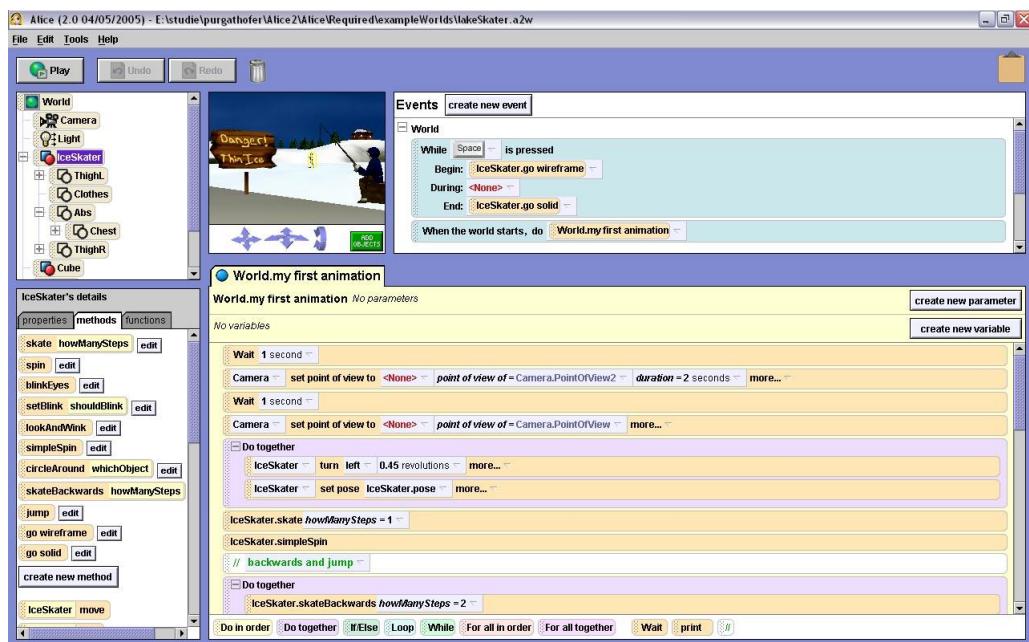


Fig. 2.7. Animación básica de un patinador de hielo en Alice. [41]

## 2.4 Evolución a diagramas de flujo ejecutables educativos

Si bien desarrollos tempranos como GRAIL [8] ya habían explorado la ejecución de diagramas de flujo, su impacto fue limitado por la tecnología de la época y su carácter experimental. Paralelamente a la exploración de diversos paradigmas de VPLs, también hubo esfuerzos por formalizar y mejorar la robustez de los propios diagramas de flujo, incluso en contextos no estrictamente educativos al principio:

## DRAKON (1995)

Un ejemplo notable de este enfoque es el lenguaje DRAKON (acrónimo ruso de “Lenguaje Algorítmico Ruso Amigable que Proporciona Claridad”), desarrollado en la Unión Soviética para el programa espacial Buran durante la década de 1980 [27, 52]. DRAKON es un lenguaje algorítmico visual diseñado con el objetivo primordial de crear programas que fueran fáciles de entender, verificar y mantener, especialmente para sistemas críticos. Se basa en un conjunto estricto de reglas para la construcción de diagramas de flujo (conocidos como “DRAKON-charts”) que buscan eliminar ambigüedades y estandarizar la representación visual del control de flujo.

Aunque su origen es industrial, los principios de DRAKON sobre claridad algorítmica y su capacidad para ser traducido a código por herramientas como DRAKON Editor [53] han generado interés en su aplicación educativa como una forma de enseñar diseño algorítmico. Su énfasis en una representación visual rigurosa lo distingue como un formalismo que va más allá del simple dibujo de diagramas. La Figura 2.8 representa un ejemplo de este formalismo con la implementación del algoritmo de búsqueda de Dijkstra.

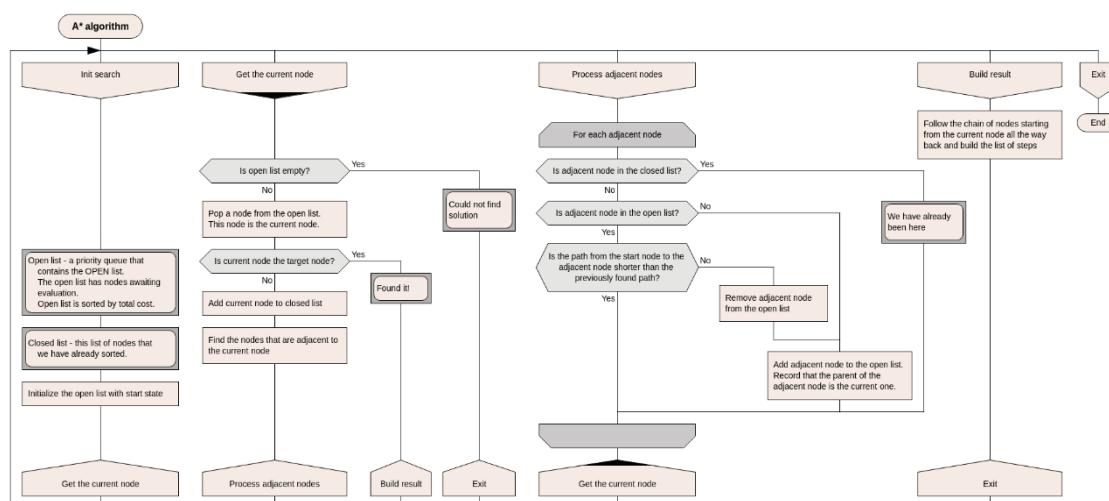


Fig. 2.8. Algoritmo de búsqueda Dijkstra en DRAKON. [42]

Sin embargo, el verdadero impulso para el uso de diagramas de flujo ejecutables como herramienta pedagógica surgió más tarde con el desarrollo de software destinado a entornos educativos y ordenadores personales. Como se argumentó en la [sección 1.2](#), la capacidad de no solo diseñar, sino también ejecutar interactivamente un diagrama, introducir datos y observar el flujo paso a paso proporciona una retroalimentación inmediata que es crucial para que los estudiantes puedan validar la corrección de su lógica y comprender su comportamiento dinámico. Esta necesidad de un enfoque más práctico y menos abstracto para el aprendizaje de la algoritmia fundamental motivó la creación de las siguientes herramientas:

### **Visual Logic (2005)**

Desarrollada por PGS Systems, LLC, Visual Logic<sup>©</sup> fue una aplicación de escritorio diseñada como una herramienta gráfica interactiva que permite a los estudiantes, especialmente a los principiantes, construir diagramas de flujo utilizando símbolos estándar para representar estructuras algorítmicas [13]. La característica distintiva y fundamental residía en que estos diagramas no eran meramente representaciones estáticas, sino que eran ejecutables. Los estudiantes podían ejecutar sus diagramas de flujo directamente dentro de la herramienta, introducir datos de entrada cuando se les solicitaba, observar visualmente cómo el flujo de control se movía paso a paso a través de los diferentes símbolos del diagrama y ver los datos de salida generados.

Esta capacidad de ejecución interactiva proporcionaba una retroalimentación inmediata, crucial para que los estudiantes validaran la corrección de su lógica, depurasesen errores conceptuales y comprendiesen el comportamiento dinámico de sus algoritmos. Visual Logic conseguía una sintaxis mínima al permitir a los estudiantes seleccionar sentencias de un menú desplegable e introducir únicamente constantes, nombres de variables o expresiones, cubriendo las estructuras fundamentales como entrada/salida, asignación, condiciones, bucles y procedimientos [13]. Este enfoque permitía a los usuarios centrarse en la lógica algorítmica subyacente sin la sobrecarga cognitiva impuesta por las reglas sintácticas de un lenguaje de programación textual, facilitando así la transición posterior a dichos lenguajes [13]. La Figura 2.9 muestra un ejemplo de un programa simple en el entorno de Visual Logic como claro testimonio de esta filosofía pedagógica.

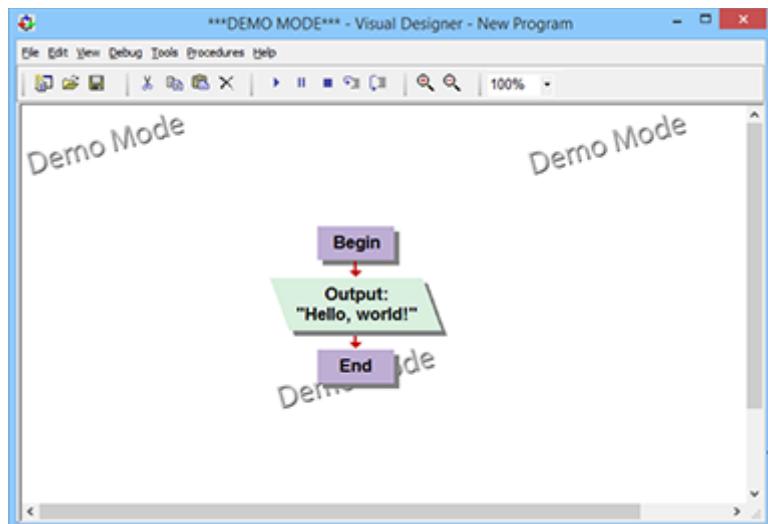


Fig. 2.9. Programa "Hello, world!" en Visual Logic. [39]

### **RAPTOR (2005)**

Desarrollado en la Academia de la Fuerza Aérea de los Estados Unidos, RAPTOR (*Rapid Algorithmic Prototyping Tool for Ordered Reasoning*) es un entorno de programación visual diseñado específicamente para enseñar la resolución algorítmica de

problemas a estudiantes en formación [4, 9, 47]. Surgió como respuesta a dos observaciones clave: la primera, que los estudiantes suelen dedicar más tiempo a lidiar con la sintaxis de un lenguaje de programación que con la lógica del problema en sí; y la segunda, que la naturaleza textual de los entornos de programación tradicionales a menudo no se alinea con el estilo de aprendizaje visual de la mayoría de estudiantes [4].

En RAPTOR, los algoritmos se construyen mediante un conjunto limitado de símbolos de diagrama de flujo. Como se evidencia en la Figura 2.10, estos diagramas son directamente ejecutables dentro del entorno, permitiendo la visualización paso a paso del flujo de control (nótese el símbolo de decisión resaltado en verde) y el seguimiento del estado de las variables (panel lateral). La sintaxis requerida es mínima y se valida al instante, previniendo errores comunes y permitiendo a los estudiantes centrarse en la lógica [4].

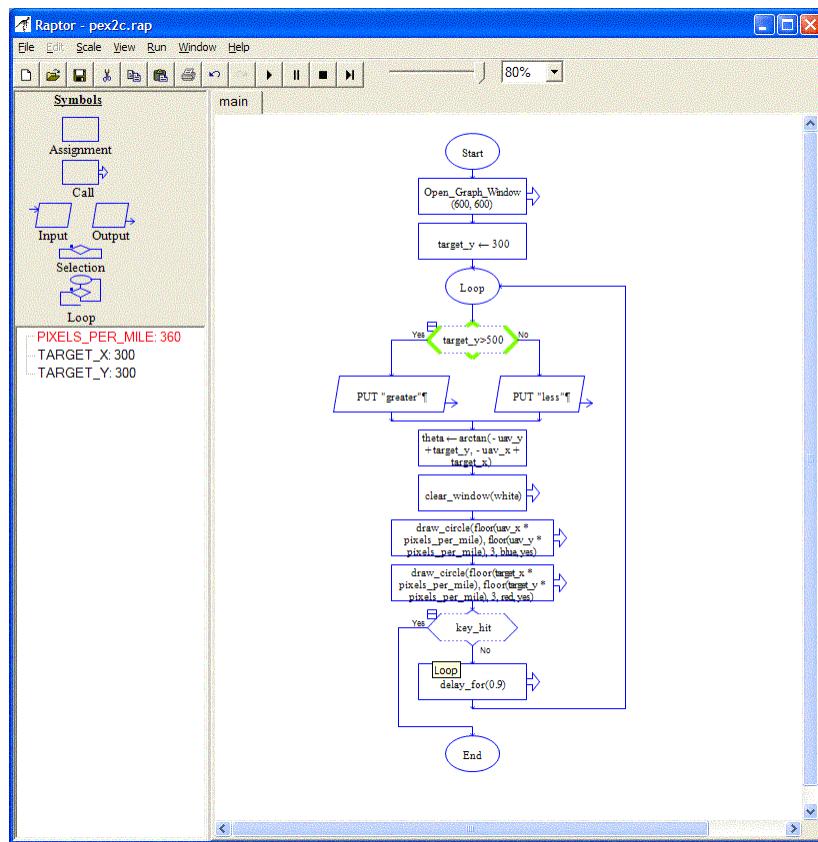


Fig. 2.10. Ejemplo de programa en Raptor. [47]

Estudios demostraron que los estudiantes preferían RAPTOR, el cual les ayudaba a mejorar su comprensión algorítmica y facilitaba la transición a lenguajes textuales, como se evidenció en su adopción y adaptación en la Academia Militar de EE. UU. (USMA) para preceder la enseñanza de Java [9]. De esta forma, RAPTOR se consolidó como una herramienta pedagógica influyente para introducir los fundamentos de la algoritmia mediante diagramas de flujo ejecutables.

## Flowgorithm (2014)

Desarrollado inicialmente por Devin Cook en la Universidad Estatal de Sacramento en 2014, Flowgorithm es una aplicación de escritorio gratuita diseñada para permitir a los estudiantes crear programas utilizando diagramas de flujo estándar [18, 48]. Su objetivo principal es ayudar a los principiantes a concentrarse en la lógica algorítmica sin la carga de la sintaxis de un lenguaje de programación específico. Al igual que RAPTOR y Visual Logic, los diagramas de flujo en Flowgorithm son ejecutables, permitiendo a los usuarios introducir datos, observar el flujo de control paso a paso y ver los resultados, lo que proporciona una retroalimentación inmediata esencial para el aprendizaje. La Figura 2.11, que muestra la implementación de un juego de “adivinar el número”, es un claro ejemplo de la filosofía de la herramienta.

Una de las características más destacadas de Flowgorithm es su robusta capacidad para generar automáticamente código fuente a partir del diagrama de flujo en una gran variedad de lenguajes de programación populares, incluyendo C#, C++, Java, Python, JavaScript, entre otros [18]. Esta funcionalidad actúa como un puente directo hacia la programación textual, permitiendo a los estudiantes ver la correspondencia entre los elementos visuales del diagrama y las construcciones sintácticas del código.

A diferencia de RAPTOR o Visual Logic, que pueden inferir tipos de datos dinámicamente, Flowgorithm requiere la declaración explícita de variables (Entero, Real, Texto, Booleano) antes de su uso, lo cual, si bien introduce un paso adicional, familiariza a los estudiantes con una práctica común en muchos lenguajes tipados estáticamente y puede mejorar la claridad en la generación de código [18].

Estudios recientes continúan validando su eficiencia pedagógica, demostrando que su uso, particularmente cuando se integra con plataformas que facilitan la colaboración como Discord, puede mejorar significativamente el desarrollo del pensamiento computacional en los estudiantes al fomentar un aprendizaje más interactivo y de apoyo mutuo [31].

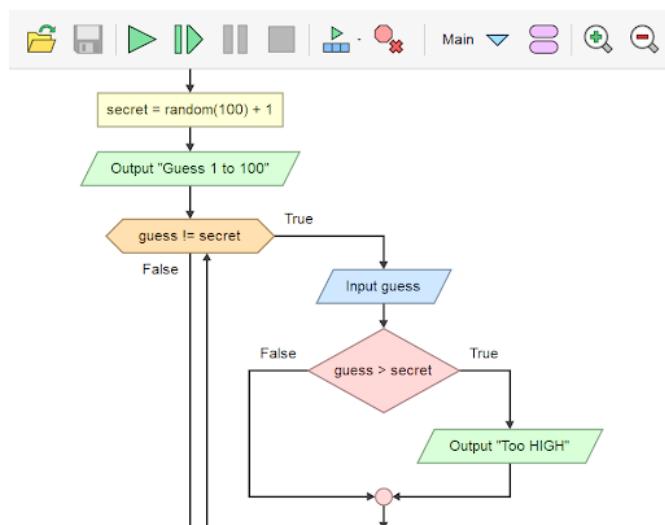


Fig. 2.11. Ejemplo de programa en Flowgorithm. [48]

Estas herramientas de diagramas de flujo ejecutables de escritorio demostraron colectivamente el valor pedagógico de permitir a los estudiantes diseñar, ejecutar y depurar algoritmos visualmente. Facilitaron que los principiantes se centraran en la lógica algorítmica fundamental antes de enfrentarse a las complejidades sintácticas. Sin embargo, una característica común y limitación significativa de estas soluciones es su naturaleza como aplicaciones de escritorio. Esto conlleva la necesidad de instalación, dependencia del sistema operativo y una menor accesibilidad en comparación con las soluciones basadas en web.

## 2.5 Auge de la programación visual por bloques y transición a la web

Paralelamente a la consolidación de las herramientas de diagramas de flujo ejecutables en entornos de escritorio, otro paradigma de Lenguaje de Programación Visual (VPL) comenzó a ganar una gran relevancia, transformando la enseñanza de la programación especialmente en etapas tempranas y cursos introductorios: la programación visual basada en bloques. Este enfoque, que tiene raíces en desarrollos como LogoBlocks [1] y fue popularizado por entornos como Alice [6, 12] con su manipulación de objetos 3D mediante bloques de instrucciones, alcanzó una difusión masiva con la llegada de herramientas específicamente diseñadas para este paradigma y, crucialmente, su exitosa adopción de la plataforma web.

### **Scratch (2007)**

Desarrollado por el Lifelong Kindergarten Group en el MIT Media Lab y lanzado en 2007, Scratch se ha convertido en un referente mundial de los VPLs educativos, diseñado principalmente para jóvenes [30, 54]. Su paradigma se basa en la programación visual basada en bloques: los usuarios crean historias interactivas, juegos y animaciones encajando bloques gráficos de colores que representan comandos, estructuras de control y operadores, en lugar de escribir código textual [20]. La Figura 2.12 ilustra la interfaz característica de Scratch, donde se integran la paleta de bloques, el área de *scripts* y el escenario de ejecución en una única pantalla, reflejando su filosofía de retroalimentación inmediata. Esta aproximación reduce significativamente la carga cognitiva asociada a la sintaxis, permitiendo a los usuarios centrarse en la lógica algorítmica.

La filosofía de Scratch se articula en torno a tres ejes: hacer la programación más “tinkerable”, “meaningful” y social [30]. La “tinkerability” se manifiesta en una interfaz interactiva donde los bloques y scripts pueden ejecutarse instantáneamente, sin modos separados de edición y ejecución, y con un diseño “failsoft” que minimiza los errores sintácticos y la frustración [20]. La creación de proyectos de carácter personal se facilita mediante la importación de multimedia y herramientas de edición integradas.

Un componente distintivo y crucial de Scratch es su comunidad online, que permite a los usuarios compartir sus creaciones, explorar los proyectos de otros, ofrecer y recibir

comentarios, y “remixar” (modificar y adaptar proyectos existentes) [20]. Esta plataforma web democratizó el acceso a una herramienta de programación potente, facilitando la colaboración y superando las barreras de instalación típicas de las aplicaciones de escritorio, y fomentó un ecosistema de aprendizaje colaborativo a gran escala.

Scratch ha demostrado ser una herramienta efectiva para el desarrollo del pensamiento computacional, abarcando conceptos fundamentales (como secuencias, bucles o condicionales), prácticas de diseño (iteración, depuración, modularización) y la formación de nuevas perspectivas sobre la tecnología y la creación [2]. El éxito masivo de Scratch en hacer la programación accesible y atractiva, junto con su modelo basado en web, ha influido considerablemente en el diseño de herramientas educativas posteriores, destacando el valor de la retroalimentación inmediata y la facilidad de experimentación.

No obstante, a pesar de la efectividad de Scratch para la introducción a la programación, un área de debate y estudio continuo es la transición desde entornos basados en bloques hacia la programación textual tradicional, que es la que los estudiantes suelen encontrar en niveles educativos superiores como la ingeniería. Si bien los bloques eliminan la barrera sintáctica inicial, investigaciones sugieren que la ventaja obtenida con los bloques no siempre se traduce automáticamente en un mejor rendimiento o una transición más fluida a lenguajes textuales como Java [35]. Esto plantea la pregunta de si otros paradigmas visuales, como los diagramas de flujo, podrían ofrecer un puente más natural debido a su representación estructural más cercana al código textual, al mismo tiempo que conservan los beneficios de la reducción de la carga sintáctica.

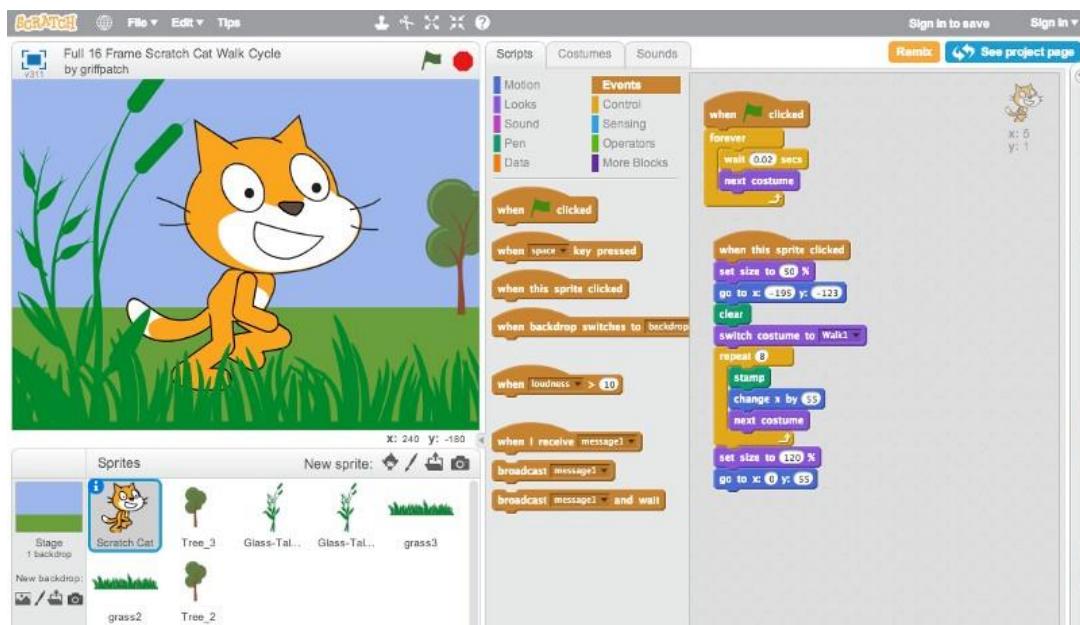


Fig. 2.12. Ejemplo de programa en Scratch. [38]

## Blockly (2012)

Siguiendo la estela del éxito de los lenguajes basados en bloques, Google lanzó Blockly en 2012. A diferencia de Scratch, que es una aplicación completa con un lenguaje de bloques definido, Blockly es una biblioteca de JavaScript de código abierto que permite a los desarrolladores añadir editores de programación visual basados en bloques a sus propias aplicaciones web y móviles [28, 55].

Esto significa que Blockly no es un lenguaje en sí mismo, sino un *framework* que proporciona la “gramática” visual (la interfaz de usuario del editor de bloques, cómo encajan, etc.), pero requiere que los desarrolladores definan el “vocabulario” específico de bloques: su apariencia, qué acciones representan y, crucialmente, el código textual que estos bloques generan (en lenguajes como JavaScript, Python, Lua, PHP, Dart o lenguajes personalizados) [28]. La Figura 2.13, por ejemplo, muestra un conjunto de bloques que implementan una función de conversión de temperatura, actuando como el equivalente visual de una función en un lenguaje textual. Los desarrolladores también deben integrar Blockly con alguna forma de salida o entorno de ejecución para que los programas creados con los bloques tengan efecto.

La gran flexibilidad de Blockly ha permitido su adopción en una diversidad de aplicaciones, que van desde la programación de personajes animados en pantalla, la creación de guiones para historias, el control de robots, e incluso la generación de documentos legales [28]. Al igual que otros VPLs basados en bloques, su objetivo principal es reducir la complejidad sintáctica inherente a los lenguajes textuales, permitiendo a los usuarios, especialmente a principiantes, concentrarse en la lógica algorítmica y la resolución de problemas. Su naturaleza como biblioteca la convierte en una herramienta poderosa para crear experiencias de programación visual altamente personalizadas, donde los desarrolladores deben considerar el estilo de bloques y qué APIs son adecuadas para su público objetivo para que la herramienta sea efectiva [28].

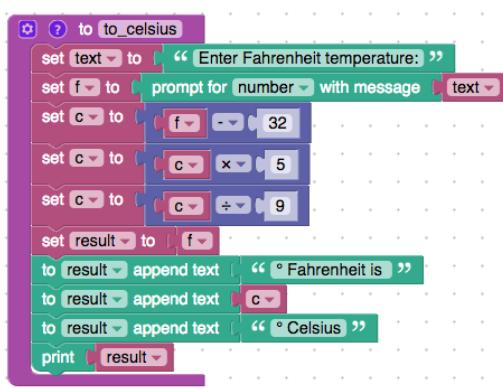


Fig. 2.13. Ejemplo de programa para convertir temperatura de Fahrenheit a Celsius en Blockly. [43]

El auge de Scratch y Blockly consolidó el dominio de los VPLs basados en bloques en el ámbito educativo, especialmente por su facilidad de uso y su exitosa implementación

web, ofreciendo una accesibilidad y alcance sin precedentes. Esta marcada transición hacia la web por parte de los VPLs basados en bloques contrastó con la situación de las herramientas de diagramas de flujo ejecutables (como RAPTOR o Flowgorithm) que, a pesar de su valor pedagógico, permanecieron en gran medida como aplicaciones de escritorio.

## 2.6 Situación actual y oportunidades

El análisis de la evolución de las herramientas para la enseñanza de la programación mediante representaciones visuales revela un panorama diverso y en constante desarrollo. Por un lado, los diagramas de flujo ejecutables, representados por herramientas consolidadas como RAPTOR [4, 47] y Flowgorithm [18, 48], continúan demostrando su valor pedagógico en la educación superior. Su capacidad de visualizar de forma clara el flujo de control, el estado de las variables y permitir una ejecución paso a paso sin la carga sintáctica inicial los mantiene como una opción relevante para que los estudiantes se centren en la lógica algorítmica fundamental [11, 16]. La funcionalidad de generación de código, especialmente robusta en Flowgorithm, también es altamente valorada como un mecanismo para facilitar la transición a la programación textual, mostrando la correspondencia entre los elementos visuales y las construcciones sintácticas [18].

Por otra parte, los VPLs basados en bloques, con Scratch [30, 54] y la biblioteca Blockly [28, 55] a la cabeza, han revolucionado la introducción a la programación en etapas más tempranas y en contextos más amplios, gracias en gran medida a su exitosa transición y consolidación en plataformas web. Estas herramientas han logrado una accesibilidad y un alcance masivos, eliminando las barreras de instalación y permitiendo el acceso desde prácticamente cualquier dispositivo con un navegador. La Tabla 2.1 muestra una comparativa de todas las herramientas analizadas, y la Figura 2.14 muestra una representación visual de su evolución cronológica.

TABLA 2.1. COMPARATIVA DE HERRAMIENTAS DE PROGRAMACIÓN VISUAL.

Herramienta (Año)	Paradigma	Plataforma	Ejecutable	Generación de código	Colaboración	¿Requiere instalación?
GRAIL (1969)	Diagrama de flujo	Sistema experimental	Sí	Sí (a máquina)	No	Sí
Pygmalion (1975)	Progr. por ejemplo	Sistema experimental	Sí	No	No	Sí
Prograph (1983)	Flujo de datos	Escritorio	Sí	No	No	Sí
LabVIEW (1986)	Flujo de datos	Escritorio	Sí	No	No	Sí
LogoBlocks (1996)	Prog. por bloques	Escritorio	Sí	No	No	Sí

Alice (1999)	Prog. por bloques (3D)	Escritorio	Sí	No	No	Sí
DRAKON (1995)	Diagrama de flujo	Escritorio	Sí	Sí (con herramientas)	No	Sí
Visual Logic (2005)	Diagrama de flujo	Escritorio	Sí	No	No	Sí
RAPTOR (2005)	Diagrama de flujo	Escritorio	Sí	No	No	Sí
Flowgorithm (2014)	Diagrama de flujo	Escritorio	Sí	Sí	No	Sí
Scratch (2007)	Prog. por bloques (2D)	Web	Sí	No	Sí (comunidad)	No
Blockly (2012)	Prog. por bloques	Web	Sí (con implementación)	Sí	No	No
<b>TFG (2025)</b>	<b>Diagrama de flujo</b>	<b>Web</b>	<b>Sí</b>	<b>Sí</b>	<b>Sí (Obj. Secundario)</b>	<b>No</b>

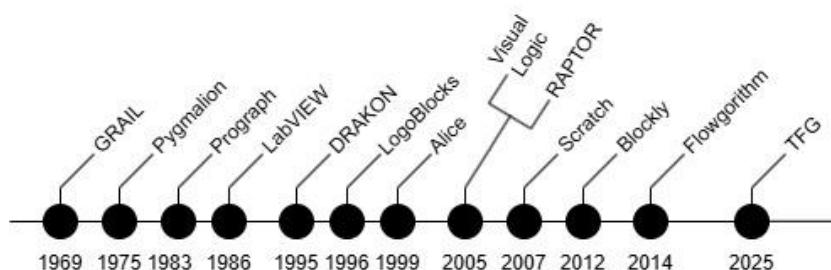


Fig. 2.14. Evolución cronológica de herramientas de programación visual.

Sin embargo, esta divergencia en la adopción de plataformas, los diagramas de flujo ejecutables predominantemente en escritorio y los bloques mayoritariamente en web, junto con las consideraciones pedagógicas específicas, configura la situación actual y define una clara oportunidad para el desarrollo de nuevas herramientas:

- **Brecha de accesibilidad para diagramas de flujo ejecutables:** La principal limitación de herramientas potentes y probadas como RAPTOR y Flowgorithm sigue siendo su naturaleza como aplicaciones de escritorio. Esto restringe su uso a entornos donde la instalación es factible, excluyendo o dificultando su adopción en contextos donde se prefiere o requiere acceso a través de un navegador (por ejemplo, en dispositivos personales de los estudiantes sin permisos de administrador).

- **Potencial de los diagramas de flujo para la transición a texto en ingeniería:** Como se ha discutido en relación con los VPLs de bloques [35], existe un debate sobre la efectividad de la transferencia de conocimiento desde los bloques a la programación textual que los estudiantes de ingeniería inevitablemente encontrarán. Los diagramas de flujo, al ofrecer una representación visual que mapea de forma más directa las estructuras secuenciales, condicionales e iterativas del código procedural, podrían constituir un puente conceptual más natural y efectivo para este público específico, facilitando la comprensión de la lógica subyacente antes de abordar la complejidad sintáctica de lenguajes como Python, Java o C.
- **Demanda de herramientas web modernas:** La tendencia general en el software educativo es hacia soluciones basadas en web debido a su facilidad de distribución, actualización y acceso multiplataforma. Una aplicación web que permita el diseño, ejecución y depuración de diagramas de flujo, así como la generación del código correspondiente, con una interfaz de usuario intuitiva y moderna llenaría un vacío existente, combinando la robustez pedagógica de los diagramas de flujo ejecutables con las ventajas de la tecnología web actual.
- **Generación de código como facilitador del aprendizaje:** La capacidad de generar automáticamente el código fuente correspondiente al diagrama de flujo en lenguajes de programación relevantes (como Python) es una característica crucial. No solo refuerza la comprensión de la equivalencia entre la representación visual y textual, sino que también proporciona a los estudiantes un punto de partida tangible para sus primeras incursiones en la programación basada en texto.
- **Potencial para funcionalidades enriquecidas por la web:** Una plataforma web abre la puerta a funcionalidades que son más difíciles de implementar o menos comunes en aplicaciones de escritorio tradicionales. Esto incluye la colaboración en tiempo real, que según estudios [31] puede mejorar significativamente el desarrollo del pensamiento computacional y las habilidades de resolución de problemas al fomentar entornos de aprendizaje más interactivos y de apoyo mutuo. Asimismo, la integración de módulos como un “Banco de problemas” con validación automática se ve facilitada por una arquitectura web y puede proveer a los estudiantes una herramienta valiosa para la práctica autónoma y la autoevaluación de sus conocimientos.

En este contexto, el presente Trabajo de Fin de Grado se posiciona para abordar estas limitaciones y capitalizar estas oportunidades. El desarrollo de una aplicación web para el diseño y ejecución de diagramas de flujo busca ofrecer una solución que combina la claridad pedagógica de los diagramas de flujo con la accesibilidad y modernidad de una plataforma web, integrando funcionalidades clave como la ejecución interactiva, la depuración visual y la generación de código. Este enfoque se alinea directamente con el objetivo de reducir la dificultad del proceso de aprendizaje de la programación para estudiantes poco experimentados, especialmente en el ámbito universitario y de ingeniería.



### **3 ANÁLISIS DEL PROYECTO**

Este capítulo se adentra en el análisis detallado del proyecto, comenzando por un análisis de la problemática que se busca abordar y el origen de esta necesidad. Posteriormente, se definirán los requisitos de usuario, tanto de capacidad como de restricción, que guiarán el desarrollo. A continuación, se especificarán los casos de uso que describen la interacción del usuario con el sistema, seguidos de la derivación de los requisitos de software necesarios para implementar dichas funcionalidades. Finalmente, se presentarán las matrices de trazabilidad para asegurar la coherencia y cobertura entre los requisitos de usuario y los requisitos de software, además de entre los requisitos de software y los casos de uso.

#### **3.1 Descripción y origen del problema**

El principal problema que este proyecto busca abordar es la pronunciada curva de aprendizaje que enfrentan los estudiantes al iniciarse en programación. Esta dificultad se acentúa por la necesidad de comprender simultáneamente la lógica algorítmica y la sintaxis de un lenguaje de programación. La presente iniciativa surge de una propuesta del tutor, orientada a desarrollar una herramienta que facilite este proceso, con especial énfasis en aquellos estudiantes de grados de ingeniería (como ingeniería mecánica, entre otras) donde la programación es una asignatura fundamental pero no el eje central de su formación.

#### **3.2 Requisitos de Usuario**

Los requisitos de usuario son fundamentales en el desarrollo de cualquier proyecto de software, ya que establecen las necesidades y expectativas que el sistema debe satisfacer desde la perspectiva del usuario. Sirven como un “contrato” entre los desarrolladores y los usuarios (o sus representantes), definiendo qué deberá hacer el sistema y bajo qué condiciones. Su correcta identificación y especificación son cruciales para asegurar que el producto final sea útil, usable y cumpla con los objetivos para los cuales fue concebido.

Los requisitos de usuario se clasifican en dos categorías principales:

- Requisitos de capacidad: Describen las funcionalidades que el sistema debe ofrecer al usuario, es decir, las tareas o acciones que el usuario podrá realizar con la aplicación.
- Requisitos de restricción: Especifican las limitaciones o condiciones bajo las cuales el sistema debe operar. Estas pueden incluir restricciones de rendimiento, de interfaz, de entorno operativo, de seguridad, o cualquier otra condición que limite las opciones de diseño o implementación.

La lista detallada de los requisitos de usuario identificados para este proyecto se encuentra en el [Anexo A](#). Este documento sirve como un contrato entre el estudiante y el tutor y como referencia principal para la definición del alcance funcional y las características que deberá implementar el sistema.

### 3.3 Casos de Uso

Los casos de uso describen la interacción entre los actores y el sistema para alcanzar objetivos específicos. Son una herramienta esencial para comprender y visualizar cómo se utilizará la aplicación y ayudan a asegurar que todas las funcionalidades necesarias estén contempladas.

A continuación, se enumeran y describen los casos de uso (CU) identificados para el sistema. La Figura 3.1 muestra su representación visual.

- (CU01) **Exportar diagrama de flujo:** Permite al estudiante guardar el diagrama de flujo actual en un formato específico para su uso externo o posterior importación. Este caso de uso puede considerarse una extensión adicional del caso de uso CU03, ya que se realiza sobre un diagrama existente.
- (CU02) **Importar diagrama de flujo:** Permite al estudiante cargar un diagrama de flujo previamente guardado en un formato compatible con la aplicación.
- (CU03) **Diseñar diagrama de flujo:** Es el caso de uso central que permite al estudiante crear y modificar diagramas de flujo utilizando las herramientas visuales proporcionadas por la aplicación. Implica añadir, conectar y configurar los diferentes nodos (inicio/fin, proceso, decisión, entrada/salida).
- (CU04) **Ejecutar diagrama de flujo:** Permite al estudiante poner en marcha la ejecución del diagrama de flujo diseñado. Este caso de uso incluye necesariamente el caso de uso CU03 previo para poder operar. Durante la ejecución, el sistema interpretará el diagrama, permitiendo el seguimiento del flujo y la interacción con el mismo.
- (CU05) **Depurar diagrama de flujo:** Ofrece al estudiante herramientas para analizar el comportamiento de su diagrama paso a paso, identificar errores lógicos y observar el valor de las variables en diferentes puntos de la ejecución. Este caso de uso incluye la funcionalidad del caso de uso CU04 previo, ya que la ejecución es un paso obligatorio para realizar la depuración.
- (CU06) **Traducir diagrama de flujo a código:** Permite al estudiante generar el código fuente equivalente al diagrama de flujo diseñado en un lenguaje de programación seleccionado (por ejemplo, Python). Este caso de uso incluye la funcionalidad del caso de uso CU03 debido a la necesidad de la existencia de un diagrama de flujo para poder traducirlo a código.

- (CU07) **Colaborar en línea:** Habilita a múltiples estudiantes para trabajar de forma concurrente en el mismo diagrama de flujo, visualizando cambios en tiempo real y potencialmente comunicándose a través de la plataforma.
- (CU08) **Consultar banco de ejercicios:** Permite al estudiante acceder a una colección de enunciados de ejercicios de programación propuestos, que puede intentar resolver utilizando la herramienta.
- (CU09) **Resolver ejercicio de banco de ejercicios:** Permite al estudiante utilizar las funcionalidades de la aplicación para validar su solución propuesta a través de un diagrama de flujo en correspondencia con la solución al ejercicio seleccionado. Este caso de uso incluye el caso de uso CU03, ya que es necesario construir un diagrama previamente, y extiende el caso de uso CU08 previo, ya que primero se consulta y selecciona un ejercicio.

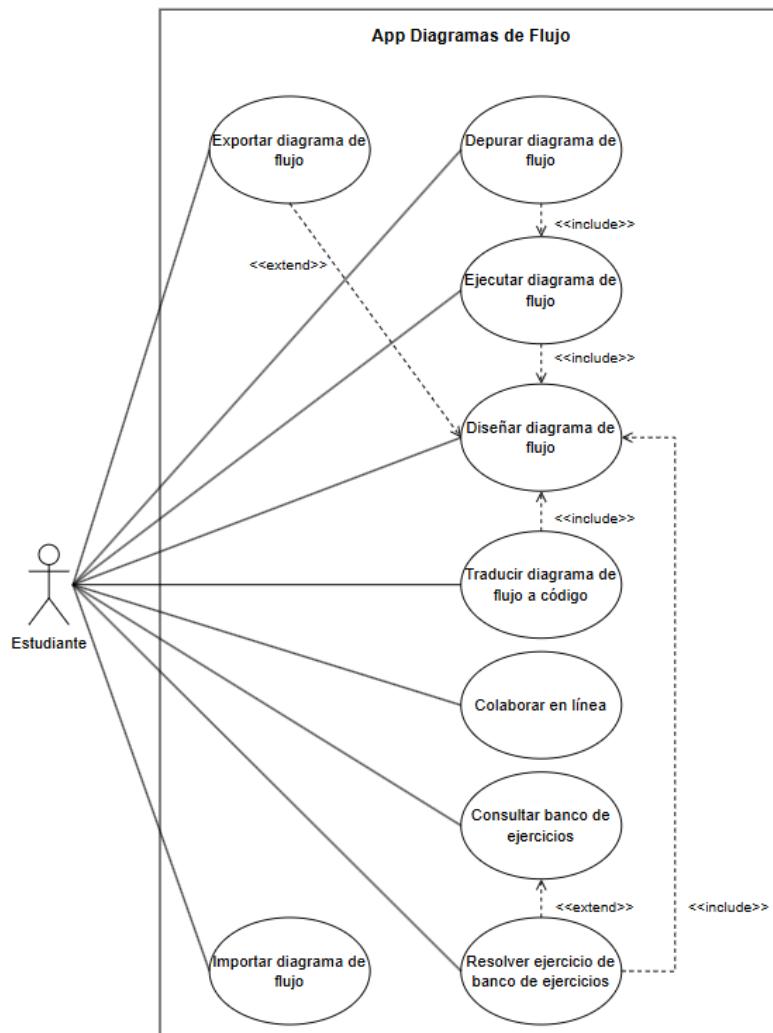


Fig. 3.1. Diagrama de casos de uso.

### 3.4 Requisitos de Software

Los requisitos de software traducen las necesidades del usuario, definidas en la [sección 3.2](#), en especificaciones técnicas concretas que guiarán el diseño y la implementación del sistema. Se dividen en dos categorías: requisitos funcionales, que describen las funcionalidades que el sistema debe proporcionar, y requisitos no funcionales, que definen las cualidades y restricciones del sistema, como rendimiento, usabilidad, fiabilidad o seguridad. Cada requisito de software se deriva de uno o varios requisitos de usuario para asegurar la trazabilidad. La prioridad de los requisitos se define en base a la escala de priorización “MoSCoW” definida en el [Anexo A](#).

#### 3.4.1 Requisitos Funcionales

Los requisitos funcionales (FR – *Functional Requirement*) detallan las acciones específicas que la aplicación debe ser capaz de ejecutar. La Tabla 3.1 a continuación detalla cada uno de ellos.

TABLA 3.1. REQUISITOS FUNCIONALES

ID	DESCRIPCIÓN	MÓDULO	PRIORIDAD	FUENTE
FR-01	El sistema proporcionará un lienzo gráfico interactivo donde se puedan renderizar y manipular los elementos del diagrama de flujo.	Diseño	M	UR-01
FR-02	El sistema ofrecerá una paleta de herramientas que permita al usuario seleccionar y arrastrar al lienzo los siguientes tipos de nodos: Inicio, Fin, Proceso (Asignación), Decisión (Condicional), Entrada, Salida.	Diseño	M	UR-02
FR-03	El sistema permitirá la creación de conectores (aristas) entre los nodos del diagrama. La creación se iniciará al arrastrar desde un punto de anclaje de un nodo de origen a un punto de anclaje de un nodo de destino. El sistema aplicará las siguientes reglas de validación de conexión para mantener la integridad lógica del diagrama de flujo:	Diseño	M	UR-03

## VALIDACIÓN DE SALIDAS

- Inicio, Proceso,  
Entrada/Salida: Solo  
una conexión.
- Fin: Ninguna conexión.
- Decisión: Máximo dos  
conexiones.
- Excepto el nodo de  
Decisión, los nodos no  
podrán realizar  
conexiones a sí  
mismos.

## VALIDACIÓN DE ENTRADAS

- Inicio: Ninguna  
conexión.
- Fin, Proceso, Decisión,  
Entrada/Salida:  
Múltiples conexiones.

FR-04	Al seleccionar un nodo que requiera lógica interna (Proceso, Decisión, Entrada/Salida), el sistema deberá presentar un panel de propiedades. Este panel contendrá una interfaz para la construcción de expresiones estructuradas con variables existentes, valores literales (numéricos, de texto o lógicos), operadores o funciones integradas (como las de conversión de tipo) que definirán el contenido del nodo.	Diseño	M	UR-04
FR-05	El sistema permitirá la selección de uno o varios nodos y/o conectores y ofrecerá una función para eliminarlos del lienzo.	Diseño	M	UR-05
FR-06	El sistema soportará la funcionalidad de arrastrar y soltar ( <i>drag-and-drop</i> ) para todos los nodos dentro del lienzo, actualizando su posición. Los conectores asociados a un nodo movido	Diseño	M	UR-06

	deberán redibujarse automáticamente para mantener la conexión.			
FR-07	El sistema implementará una función de “Exportar” que serialice la estructura del diagrama actual (nodos, posiciones, conexiones y contenido) a un formato de archivo estándar.	Diseño	M	UR-07
FR-08	El sistema implementará una función de “Importar” que permita al usuario seleccionar un archivo local, el cual será leído y analizado para reconstruir el diagrama de flujo en el lienzo a partir de los datos del archivo.	Diseño	M	UR-07
FR-09	El sistema proveerá controles (ejecutar, pausar, reanudar y detener) para gestionar el ciclo de vida del motor de ejecución.	Ejecución	M	UR-08
FR-10	Cuando el flujo de ejecución alcance un nodo de “Entrada”, el sistema pausará la ejecución y mostrará una ventana emergente con un campo para que el usuario introduzca el valor solicitado. Este valor será almacenado en la variable correspondiente.	Ejecución	M	UR-09
FR-11	Cuando el flujo de ejecución alcance un nodo de “Salida”, el sistema mostrará el valor de la expresión en un panel de salida visible para el usuario.	Ejecución	M	UR-10
FR-12	El sistema gestionará el estado de todas las variables del diagrama. Durante la ejecución, se mostrará una vista que liste cada variable y su valor actual, actualizándose después de cada paso de ejecución.	Depuración	S	UR-11
FR-13	El sistema resaltará visualmente el nodo y el conector que se están	Ejecución	M	UR-12

	ejecutando en cada paso del flujo de ejecución.			
FR-14	Durante y tras finalizar la ejecución, el sistema ofrecerá un historial de cambios por variable. Al seleccionar una variable, se mostrará una vista cronológica de todos los valores que ha tomado durante la ejecución.	Depuración	S	UR-13
FR-15	El sistema registrará la secuencia de nodos visitados durante la ejecución. Al finalizar, esta secuencia (la cadena de ejecución) podrá ser mostrada al usuario en un panel de historial.	Depuración	S	UR-14
FR-16	El sistema proporcionará un traductor que recorra la estructura del diagrama de flujo y genere una cadena de texto que represente el código fuente equivalente en un lenguaje de programación seleccionable.	Código	M	UR-15
FR-17	El sistema dispondrá de un módulo que presente al usuario una lista de enunciados de ejercicios, obtenidos de una fuente de datos interna.	Ejercicios	C	UR-16
FR-18	Al seleccionar un ejercicio, el sistema tendrá una función para ejecutar el diagrama de flujo contra un conjunto de casos de prueba predefinidos para ese ejercicio y comparar las salidas obtenidas con las salidas esperadas de la solución.	Ejercicios	C	UR-17
FR-19	El sistema soportará la edición colaborativa de diagramas de flujo. Cualquier acción de creación, eliminación o modificación de nodos y conexiones realizada por un usuario deberá ser propagada y visible para todos los demás	Colaboración	C	UR-18

	participantes de la misma sesión.			
FR-20	El sistema integrará una funcionalidad de comunicación por audio y vídeo, permitiendo la transmisión entre los participantes de una sesión de colaboración.	Colaboración	C	UR-19

### 3.4.2 Requisitos No Funcionales

Los requisitos no funcionales (NFR – *Non-Functional Requirement*) definen las cualidades y restricciones operativas del sistema, asegurando que la aplicación no solo sea funcional, sino también usable, fiable y eficiente. La Tabla 3.2 a continuación detalla cada uno de ellos.

TABLA 3.2. REQUISITOS NO FUNCIONALES

ID	DESCRIPCIÓN	MÓDULO	PRIORIDAD	FUENTE
NFR-01	La interfaz de usuario deberá seguir los principios de diseño consistentes y predecibles. Los elementos de la interfaz con funciones similares deberán tener una apariencia y comportamiento unificados en toda la aplicación.	Usabilidad	M	UCR-01
NFR-02	La interfaz de usuario deberá ser eficiente. Las operaciones comunes deberán poder completarse de una forma directa y con un número reducido de pasos.	Usabilidad	M	UCR-01
NFR-03	La latencia de las interacciones en el lienzo debe ser inferior a 100ms para proporcionar una sensación de respuesta inmediata.	Rendimiento	M	UCR-01
NFR-04	En el modo de colaboración (FR-19), la propagación de cambios entre los clientes deberá tener una latencia	Rendimiento	M	UCR-01, UR-18

	inferior a 500ms en condiciones de red óptimas para asegurar una experiencia de edición en tiempo real.			
NFR-05	Los símbolos gráficos para los nodos del diagrama de flujo deberán adherirse a la representación visual estándar ISO 5807, utilizando formas y colores distintivos para ser fácilmente identificables.	Usabilidad	M	UCR-02
NFR-06	El resaltado del flujo de ejecución (FR-13) deberá ser visualmente fluido y perceptible, sin parpadeos o saltos que dificulten el seguimiento por parte del usuario.	Rendimiento	M	UCR-03
NFR-07	Las ventanas modales para la entrada de datos (FR-10) y los paneles de salida (FR-11) deberán presentar la información de forma clara, utilizando un lenguaje directo y un formato limpio para evitar la sobrecarga cognitiva.	Usabilidad	M	UCR-04
NFR-08	Todos los mensajes de error del sistema deben ser informativos, redactados en lenguaje claro y comprensible. Deben indicar la causa probable del error y, si es posible, resaltar el nodo del diagrama donde se originó el problema.	Usabilidad	M	UCR-05
NFR-09	La aplicación deberá implementarse utilizando un diseño adaptable (responsive). La interfaz deberá reorganizarse y ser completamente funcional en dispositivos con pantallas más pequeñas.	Compatibilidad	S	UCR-06

NFR-10	El motor de ejecución deberá interpretar la lógica del diagrama de flujo de forma determinista y correcta, asegurando que la semántica de cada nodo y el flujo de control se ejecuten según los estándares de los algoritmos.	Fiabilidad	M	UCR-07
NFR-11	El generador de código (FR-16) deberá producir código que sea sintácticamente válido para los lenguajes de destino y que sea semánticamente equivalente a la lógica del diagrama de flujo. El código generado deberá poder ejecutarse sin modificaciones en un intérprete estándar del lenguaje de destino correspondiente.	Fiabilidad	M	UCR-08, UCR-09
NFR-12	El analizador sintáctico de expresiones y el motor de ejecución deberán soportar los siguientes tipos de datos primitivos: cadena de texto (string), número entero (integer), número de punto flotante (float), lógico (boolean) y lista (array).	Alcance	M	UCR-10
NFR-13	La aplicación deberá ser compatible con las dos últimas versiones estables de los principales navegadores web: Google Chrome, Mozilla Firefox, Microsoft Edge, Brave y Safari.	Compatibilidad	M	UCR-11
NFR-14	La aplicación debe ser una aplicación web del lado de cliente (SPA – Single Page Application) que no requiera la instalación de ningún software o extensión de navegador por parte del usuario.	Despliegue	M	UCR-11

### **3.5 Matrices de Trazabilidad**

Las matrices de trazabilidad son herramientas esenciales en la ingeniería de software que permiten visualizar las relaciones entre diferentes artefactos del desarrollo. Su propósito es garantizar que todos los requisitos de usuario hayan sido considerados y traducidos a requisitos de software específicos, y que cada requisito de software contribuye a la implementación de uno o más casos de uso. Esto asegura la completitud del sistema y facilita la validación del producto final.

A continuación, se presentan dos matrices: la primera relaciona los requisitos de usuario con los requisitos de software, y la segunda relaciona los requisitos de software con los casos de uso del sistema.

#### **3.5.1 Trazabilidad de Requisitos de Usuario a Requisitos de Software**

Véase el [Anexo B](#) con la matriz que muestra la correspondencia entre los requisitos de usuario (de capacidad, UR – *User Requirement*, y de restricción, UCR – *User Constraint Requirement*) y los requisitos de software (funcionales, FR – *Functional Requirement*, y no funcionales, NFR – *Non-Functional Requirement*). Una “X” en la intersección indica que el requisito de software de la columna satisface, total o parcialmente, el requisito de usuario de la fila.

Como se puede observar en la matriz, cada requisito de usuario (tanto UR como UCR) tiene correspondencia con al menos un requisito de software. La presencia de al menos una “X” en cada fila confirma que la totalidad de necesidades y restricciones extraídas del cliente (en este caso, representadas por el tutor) han sido recogidas y traducidas a especificaciones técnicas para el equipo de desarrollo. Esto garantiza la completitud del análisis y asegura que no se han omitido aspectos funcionales o de calidad solicitados, estableciendo una base sólida para el diseño del sistema.

#### **3.5.2 Trazabilidad de Requisitos de Software a Casos de Uso**

Véase el [Anexo C](#) con la matriz que muestra qué requisitos de software (FR y NFR) son necesarios para implementar cada caso de uso (CU). Una “X” indica que el requisito de la fila es relevante para la realización del caso de uso de la columna.

El análisis de la matriz revela dos hechos clave. Primero, que cada requisito de software está asociado a, como mínimo, un caso de uso. Esto demuestra que no se han definido requisitos técnicos superfluos: cada especificación tiene un propósito directo en la implementación de una o más funcionalidades observables por el usuario. Segundo, que cada caso de uso está soportado por un conjunto de requisitos de software, lo que valida que las interacciones descritas tienen el respaldo técnico necesario para su realización. Esta doble verificación asegura la cohesión y eficiencia del diseño propuesto.



## 4 PLANIFICACIÓN Y PRESUPUESTO

El éxito de cualquier proyecto de desarrollo de software se basa en una cuidadosa planificación temporal y una estimación de los recursos necesarios. Este capítulo detalla la estrategia de planificación adoptada para el desarrollo de este proyecto, incluyendo una estimación formal del esfuerzo mediante la metodología de Puntos de Casos de Uso (UCP – *Use Case Points*), la metodología de desarrollo planteada, la planificación inicial de tareas y fases mediante un diagrama de Gantt y una estimación del presupuesto teórico. El objetivo es establecer una hoja de ruta clara que permita gestionar el tiempo de forma eficiente, anticipar posibles desafíos y asegurar la consecución de los objetivos dentro del plazo establecido.

### 4.1 Estimación de esfuerzo

Se ha empleado la metodología por Puntos de Casos de Uso (UCP – *Use Case Points*) para obtener una estimación formal del esfuerzo requerido para el desarrollo del proyecto. Esta técnica permite cuantificar el tamaño funcional del software basándose en la complejidad de sus casos de uso y los actores involucrados. Posteriormente, se ajusta esta medida con factores que evalúan la complejidad técnica del sistema y las características del entorno de desarrollo, para finalmente derivar una estimación del esfuerzo en horas persona.

La elección de UCP se justifica por su capacidad para proporcionar una estimación temprana basada en los casos de uso, lo que ayuda a comprender la magnitud del proyecto desde sus fases iniciales. En la [sección 3.3](#), se han identificado 9 casos de uso principales, considerando un actor principal (el estudiante). Tras evaluar los factores de complejidad técnica (resultando en un TCF de 0,89) y los factores ambientales (obteniendo un EF de 0,695), se calcularon **51,34 Puntos de Casos de Uso Ajustados (UCP)**.

Aplicando un Factor de Productividad (PHM – *Person Hours Multiplier*) de 7 horas por UCP, que se considera adecuado para el contexto de un proyecto académico individual donde se eliminan sobrecargas de equipo (comunicación, coordinación, etc.) y se enfoca en la funcionalidad esencial, se estima un esfuerzo total para el proyecto de aproximadamente **359 horas**. Esta cifra sirve como base para la planificación detallada de tareas y la asignación de tiempos en el diagrama de Gantt. El desglose completo de la estimación UCP, incluyendo la valoración de tanto los casos de uso como de cada factor y los cálculos intermedios, se encuentra en el [Anexo D](#).

Nótese que el número total de horas estimado excede ligeramente las 300 horas que se derivan de la carga de 12 créditos ECTS para la asignatura del Trabajo de Fin de Grado (25 horas/ECTS). Esta discrepancia responde a una visión inicial optimista sobre el alcance funcional abordable y reconoce, a su vez, la inclusión implícita de un componente de formación y aprendizaje de nuevas tecnologías que, si bien enriquecen el desarrollo del proyecto y las competencias del estudiante, pueden extender el tiempo neto invertido más allá de la pura ejecución de tareas con conocimiento previo consolidado. Se considera

que este esfuerzo adicional es valioso para la consecución de un proyecto completo y de calidad, aunque también sugiere la posibilidad de que algunas funcionalidades propuestas inicialmente, especialmente aquellas secundarias o más complejas, puedan requerir una simplificación o no alcanzar su completitud total dentro del cronograma establecido, priorizándose siempre la entrega de un núcleo funcional robusto y bien probado.

## 4.2 Planificación inicial

El ciclo de desarrollo de software empleado en este proyecto se basa en el modelo de desarrollo en espiral. Esta metodología se ha seleccionado debido a su naturaleza iterativa e incremental, que se ajusta eficazmente a las características y la complejidad inherente al desarrollo de una aplicación web interactiva como la de este proyecto que pueda necesitar ajustes finos respecto a la retroalimentación recibida en las constantes evaluaciones. El modelo en espiral combina la naturaleza sistemática de los modelos secuenciales con la flexibilidad de los prototipos, permitiendo abordar el proyecto en ciclos sucesivos que incluyen fases de planificación, ingeniería (diseño y desarrollo) y evaluación por parte del cliente (en este caso, autoevaluación y revisión con el tutor).

Esta aproximación permite identificar y mitigar riesgos de forma temprana, refinar los requisitos a medida que se obtiene una mejor comprensión del problema y de la solución, e ir construyendo la aplicación de forma progresiva. El modelo en espiral facilita la incorporación de mejoras de forma controlada a lo largo del desarrollo, en lugar de intentar definir todos los aspectos de antemano.

La planificación detallada de las tareas, su duración estimada y las dependencias se han plasmado en un diagrama de Gantt, el cual se presenta a continuación (Figura 4.1). Este diagrama se ha construido tomando como base el esfuerzo estimado total obtenido de la estimación por UCP, distribuyendo dicho esfuerzo entre las diferentes fases y tareas del proyecto. Se ha considerado una dedicación media de 3 horas diarias y un cronograma que se extiende desde el 27 de enero de 2025 hasta la fecha de entrega prevista el 14 de junio de 2025 (tomando unos días extra para no apresurar al límite de la entrega).

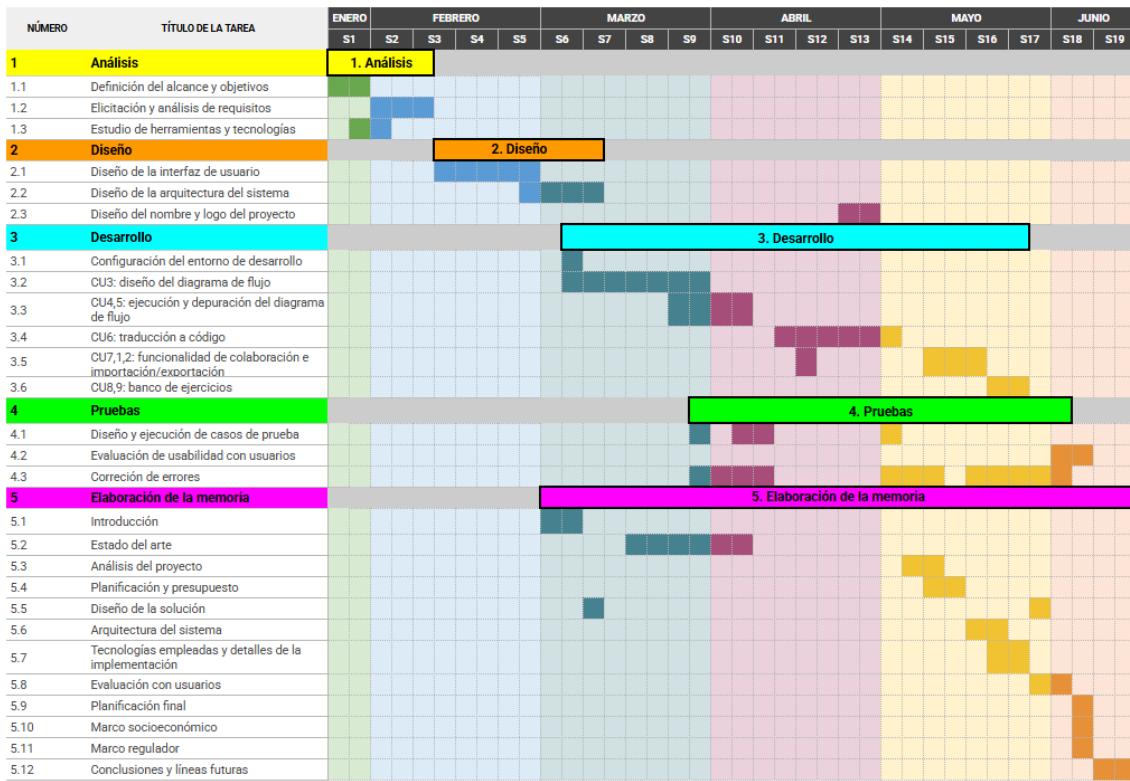


Fig. 4.1. Diagrama de Gantt del proyecto, incluyendo las fases de análisis, diseño, desarrollo, pruebas y la elaboración de la memoria.

### 4.3 Presupuesto

Si bien este proyecto se desarrolla en un contexto académico utilizando principalmente recursos personales y software gratuito, es útil realizar una estimación del presupuesto teórico que implicaría un proyecto de estas características en un entorno profesional. Esta estimación se basa en el esfuerzo total calculado mediante la metodología UCP y considera los costes asociados a los recursos humanos asumiendo que el estudiante desempeña múltiples roles con tarifas horarias medias del sector en España.

#### 4.3.1 Coste de personal (recursos humanos)

El componente principal de este presupuesto teórico lo constituyen los costes del personal. Para su cálculo, se ha distribuido el esfuerzo total entre los diferentes roles profesionales que el estudiante asumiría a lo largo del ciclo de vida del proyecto. Las tarifas horarias asignadas a cada rol, presentes en la Tabla 4.1, se basan en salarios medios para perfiles en el sector tecnológico de España [56–59].

TABLA 4.1. ROLES Y TARIFAS HORARIAS ESTIMADAS [56–59].

CATEGORÍA	€/H	DESCRIPCIÓN
<b>Jefe de Proyecto</b>	20,67 €	Responsable de la gestión del proyecto, planificación y supervisión general.
<b>Diseñador UX/UI</b>	16,59 €	Encargado del diseño de la interfaz de usuario, experiencia del usuario y mockups.
<b>Ingeniero de Software</b>	19,71 €	Responsable del diseño de la arquitectura y del desarrollo de frontend y backend.
<b>Ingeniero de Pruebas (QA)</b>	19,23 €	Encargado del diseño y ejecución de casos de prueba y evaluación de usabilidad.

La asignación de horas y el coste derivado para cada perfil se detallan en la Tabla 4.2, reflejando la dedicación porcentual estimada para cada rol dentro del esfuerzo total del proyecto.

TABLA 4.2. DEDICACIÓN Y COSTE POR ROL.

CATEGORÍA	FASE PRINCIPAL	DEDICACIÓN	HORAS ASIGNADAS	€/H	COSTE (€)
<b>Jefe de Proyecto</b>	Análisis, Gestión (Extra)	20,00%	72h	20,67 €	1.484,33 €
<b>Diseñador UX/UI</b>	Diseño	20,00%	72h	16,59 €	1.190,91 €
<b>Ingeniero de Software</b>	Diseño (Arq.), Desarrollo	45,00%	162h	19,71 €	3.184,40 €
<b>Ingeniero de Pruebas (QA)</b>	Pruebas	15,00%	54h	19,23 €	1.035,58 €
<b>Total</b>		<b>100%</b>	<b>359h</b>		<b>6.895,22 €</b>

### 4.3.2 Otros costes

Además del coste personal, se consideran los siguientes costes para completar la estimación presupuestaria:

- Infraestructuras y equipos: Se ha realizado una estimación teórica de los costes asociados al uso de la infraestructura y equipamientos necesarios. Para el ordenador personal, se ha estimado un coste de 800€ con una vida útil de 4 años, que se usará durante 6 meses de desarrollo. El servicio de un servidor virtual privado básico tendrá también una duración de 6 meses. El desglose se presenta en la Tabla 4.3.

TABLA 4.3. DESGLOSE ESTIMADO PARA INFRAESTRUCTURAS Y EQUIPOS.

Componente	Coste/tiempo	Coste total
<b>Amortización ordenador personal</b>		100,00 €
<b>Hosting VPS básico</b>	7,00€/mes	42,00 €
<b>Total</b>		<b>142,00 €</b>

- Software y licencias: Para el desarrollo de este proyecto se utilizará principalmente software de código abierto, por lo que no se estiman costes directos en esta categoría.
- Consumibles: No se prevén costes significativos de consumibles para este tipo de proyecto de desarrollo de software.
- Otros costes indirectos: Se asigna una partida de 100€ para cubrir de forma estimada los costes derivados del consumo de electricidad y el servicio de conexión a internet necesarios durante el periodo de desarrollo.

### 4.3.3 Presupuesto de venta

A partir de los costes directos e indirectos detallados en las secciones anteriores, se procede a calcular el precio de venta final del proyecto. Este cálculo transforma la estimación de costes en un presupuesto profesional que una empresa presentaría a un cliente, con costes empresariales obligatorios y los márgenes comerciales correspondientes. El proceso se estructura en dos fases:

1. **Cálculo del coste total de ejecución:** En primer lugar, se consolidan todos los costes operativos. A los costes de personal, infraestructuras y otros costes indirectos se les añade la cotización a la Seguridad Social a cargo de la empresa. Siguiendo las prácticas del sector [44], se ha estimado un 32% sobre

el total de Recursos Humanos. La suma de todos estos conceptos da como resultado el coste total de ejecución, que es la cifra que la empresa debe cubrir para no incurrir en pérdidas.

2. **Cálculo del precio de venta:** Para determinar el precio final, se aplican dos márgenes estratégicos que se definen como un porcentaje del precio de venta:
  - a. **Margen de riesgo (5%):** Un fondo para cubrir imprevistos, desviaciones en la planificación o problemas técnicos no contemplados.
  - b. **Margen de beneficio (15%):** Es la rentabilidad que la empresa obtiene por la ejecución del proyecto.

Dado que estos márgenes (20% en total) se calculan sobre el precio final, el coste total de ejecución representa el 80% restante de dicho precio. Por lo tanto, se aplica la siguiente fórmula:

$$\text{Precio de Venta} = \text{Coste de Ejecución} / (1 - (\text{Margen de Beneficio} + \text{Margen de Riesgo}))$$

La Tabla 4.4 desglosa cada uno de estos componentes y presenta el precio de venta final del proyecto antes de impuestos.

TABLA 4.4. DESGLOSE DEL PRESUPUESTO DE VENTA

Descripción	Total
<b>1. Equipo (RRHH)</b>	6.895,22 €
<b>2. Cotización Seguridad Social (32%)</b>	2.206,47 €
<b>3. Infraestructuras y Equipos</b>	142,00 €
<b>4. Software / Licencias</b>	- €
<b>5. Consumibles</b>	- €
<b>6. Otros costes (ej. costes indirectos)</b>	100,00 €
<b>COSTE DE EJECUCIÓN</b>	
	<b>9.343,69 €</b>
<b>7. % Margen</b>	15,00%
<b>8. % Riesgo</b>	5,00%
<b>PRECIO DE VENTA (SIN IMPUESTOS)</b>	
	<b>11.679,61 €</b>

## 5 DISEÑO DE LA SOLUCIÓN

El diseño de la interfaz de usuario (UI) y la experiencia de usuario (UX) es un pilar fundamental de este proyecto. Dado que la aplicación está destinada a estudiantes principiantes, una interfaz clara, intuitiva y eficiente es crucial para cumplir los objetivos pedagógicos y evitar la frustración del usuario. Este capítulo presenta el proceso de diseño visual de la aplicación, desde los bocetos iniciales hasta los mockups detallados que definen la disposición y funcionalidad de cada componente.

### 5.1 Mockups

Los *mockups* o maquetas son representaciones visuales de la interfaz de la aplicación que sirven como guía para el desarrollo. El diseño final es el resultado de un proceso iterativo que comenzó con un boceto conceptual básico y evolucionó hacia un diseño detallado y funcional.

#### 5.1.1 Boceto inicial

El punto de partida del diseño fue un boceto de bajo nivel o *wireframe* (Figura 5.1). Este primer esquema se centró exclusivamente en la distribución estructural de los componentes principales de la interfaz, sin entrar en detalles visuales. Su objetivo era definir las zonas funcionales clave y su relación espacial.

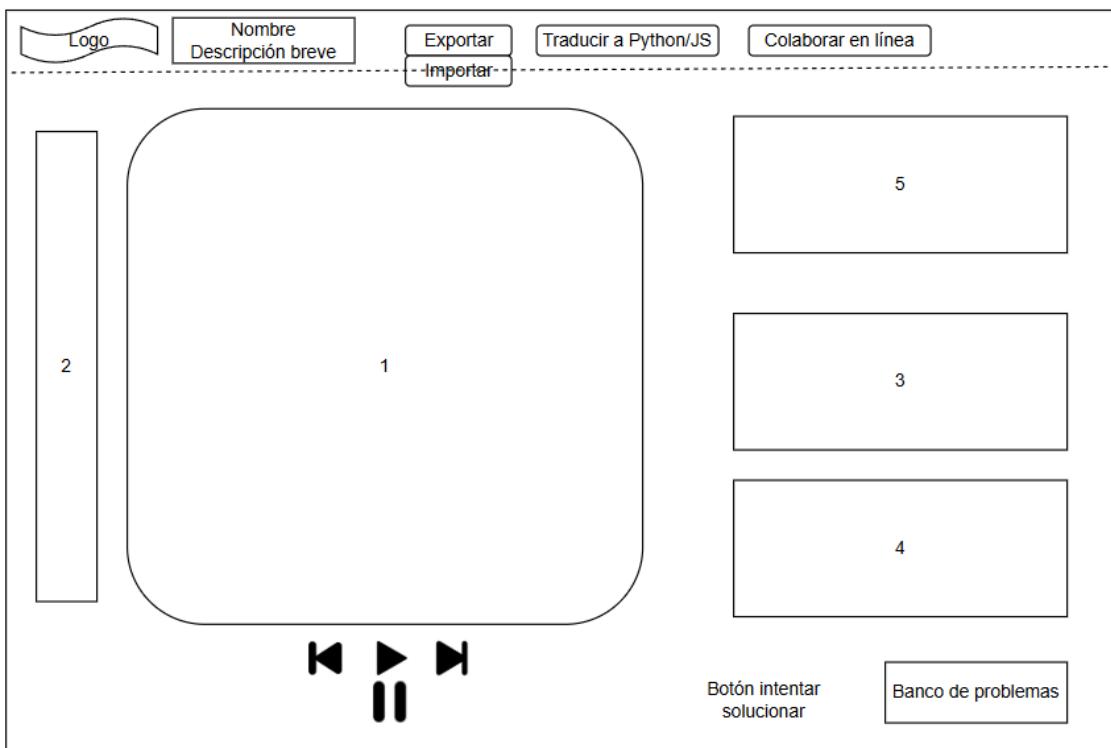


Fig. 5.1. Boceto inicial (*wireframe*) de la distribución de la interfaz.

En este boceto se identificaron las siguientes áreas funcionales:

1. Área de trabajo principal (lienzo): El espacio central para la creación del diagrama.
2. Paleta de bloques: Una barra lateral para los bloques del diagrama.
3. Panel de depuración: Para inspeccionar el estado del programa.
4. Panel de comandos: Para mostrar resultados y solicitar entradas.
5. Panel de detalles: Un área contextual para editar los detalles del elemento seleccionado.

Este esquema inicial sentó las bases para la disposición general, priorizando un gran lienzo central y agrupando las herramientas de apoyo en los laterales y la parte inferior para no interferir con el área de trabajo principal.

### 5.1.2 Mockup detallado

Partiendo del boceto inicial, se desarrolló un mockup de alta fidelidad que añade detalles visuales y una organización más refinada. Este diseño evolucionado representa la visión final de la interfaz de la aplicación, como se muestra en la Figura 5.2.

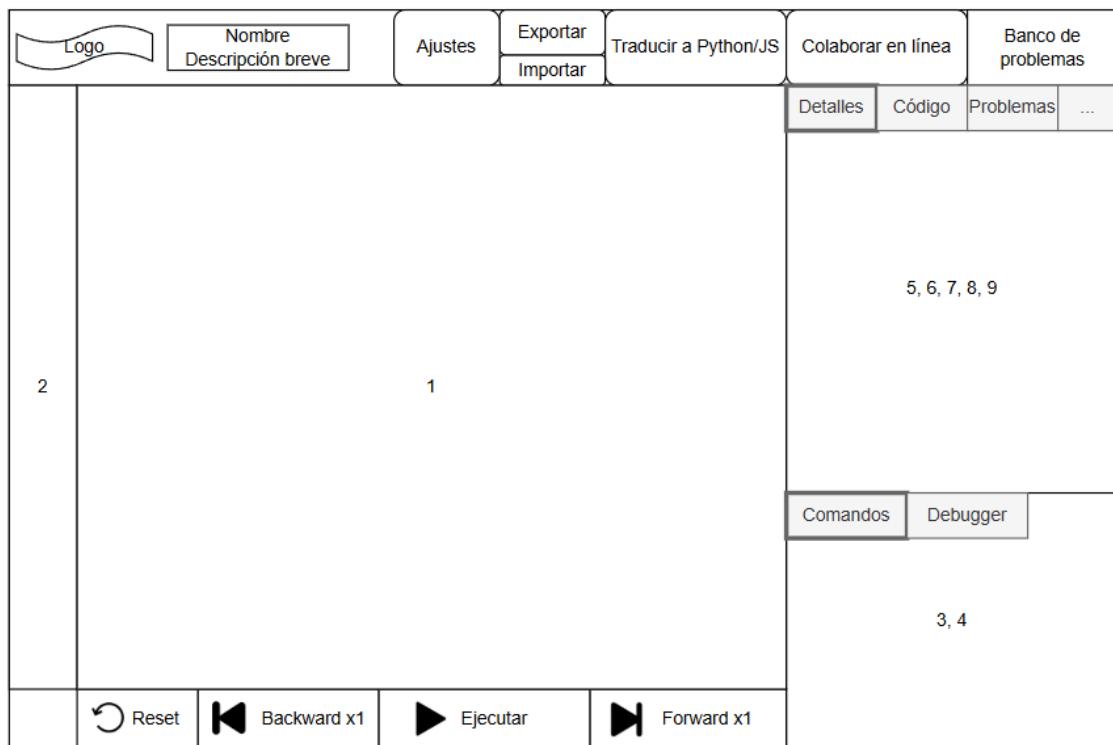


Fig. 5.2. Mockup detallado de la interfaz de la aplicación.

La interfaz final se compone de las siguientes áreas:

- **Área de trabajo principal (1):** Es el área de trabajo central donde los usuarios diseñan, visualizan y ejecutan sus diagramas de flujo. Está diseñado para ser un espacio infinito y navegable, permitiendo la creación de algoritmos de cualquier tamaño y complejidad.
- **Paleta de bloques (2):** Esta barra de herramientas vertical en el lateral izquierdo proporciona acceso rápido a todos los elementos construibles de un diagrama de flujo. El usuario puede arrastrar y soltar (*drag-and-drop*) estos nodos directamente sobre el lienzo. Incluye los nodos estándar y elementos más específicos como la declaración de variables o la creación de funciones específicas.
- Panel lateral derecho inferior:
  - **Pestaña de Comandos (3):** Muestra los mensajes de salida y permite la entrada de datos cuando el programa lo solicita.
  - **Pestaña de Depuración (4):** Permite al usuario inspeccionar el estado del programa durante la ejecución. Incluye una subpestana para ver el valor actual de todas las variables y otra para visualizar la cadena de ejecución.
- Panel lateral derecho superior:
  - **Panel de Detalles (5):** Un área contextual para editar los detalles del elemento seleccionado.
    - Bloque de declaración: Definición de los nombres y tipos de las variables.
    - Bloque de entrada: Selección de variable y opcionalmente texto de solicitud.
    - Bloque de asignación: Selección de variable y construcción de una expresión con variables existentes, operadores y literales a través de una funcionalidad de *drag-and-drop*.
    - Cualquier bloque: Detalles generales del bloque, como su posición.
    - Ningún bloque seleccionado: Ajustes del sistema, como el idioma.
  - **Panel de Código (6):** Muestra el código correspondiente traducido a partir del diagrama de flujo actual, permitiendo la selección del lenguaje y otras funcionalidades como descargar o copiar el código.
  - **Panel de Ejercicios (7):** Muestra un banco de ejercicios con sus características. Al seleccionar uno, se muestra una descripción más detallada y se da la opción de probar el diagrama actual contra casos de

prueba para su validación, además de la opción de visualizar una solución de referencia.

- **Panel de Colaboración (8):** Permite al usuario crear o unirse a salas de colaboración. En dicha sala, se mostrará el listado de integrantes con sus colores asignados y opcionalmente transmisión de vídeo.
- **Controles de ejecución:** Una barra de control en la parte inferior de la pantalla, similar a los controles de un reproductor multimedia. Permite al usuario ejecutar, pausar, detener y avanzar o retroceder paso a paso el flujo del programa, facilitando una depuración visual e interactiva.
- **Barra de menú superior:** Contiene las acciones globales de la aplicación, como la gestión de archivos (Importar/Exportar), acceso a los ajustes generales y botones para acceso rápido a las funcionalidades principales.

A continuación, se muestran los mockups detallados para cada uno de los componentes descritos (Figuras 5.3-5.13).

### **ÁREA DE TRABAJO PRINCIPAL**

Nótese que en esta versión los símbolos no siguen el estándar ISO 5807.

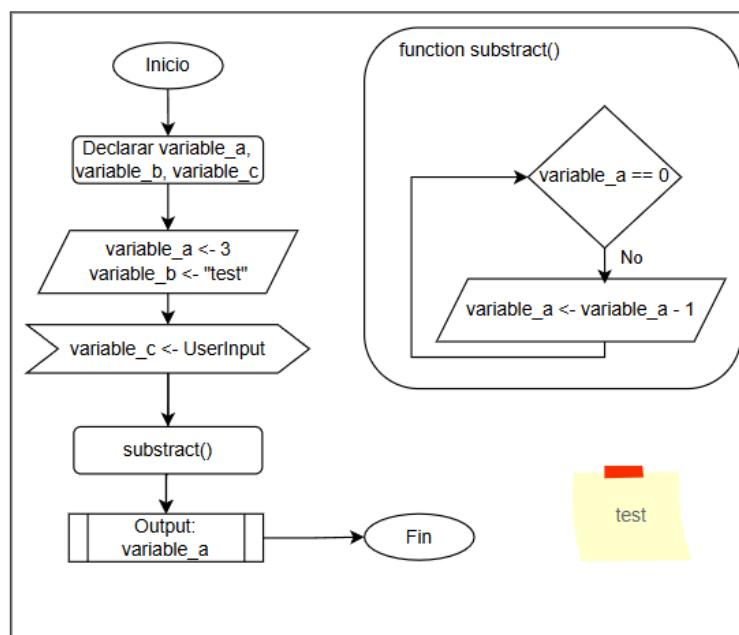


Fig. 5.3. Mockup: Área de trabajo principal.

## **PALETA DE BLOQUES**

De la misma forma, esta versión no sigue el estándar ISO 5807 para los símbolos.

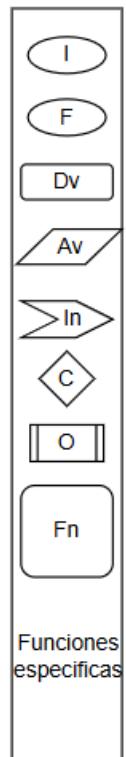


Fig. 5.4. Mockup: Paleta de bloques.

## **PANEL DE COMANDOS**

The mockup shows a window titled 'Comandos' with a tab bar also labeled 'Comandos' and 'Debugger'. The main area displays a command history:

```
> variable_a
 3
> variable_a + 1
 4
[User input required]
> 3
```

Fig. 5.5. Mockup: Panel de comandos.

## PANEL DE DEPURACIÓN

Comandos		Debugger	
Variables	Cadena ejecución		
<p>[flecha hacia abajo] variable_a: 5            2. Av_01: 5            1. Dv_01:</p> <p>&gt; variable_b: "test"            &gt; variable_c: 3.2</p>			
		<p>Inicio -&gt; Dv_01 -&gt; Av_01 -&gt; UI_01 -&gt;            subtract()_01 -&gt; O_01 -&gt; Fin</p>	

Fig. 5.6. Mockup: Panel de depuración, mostrando los apartados: variables y cadena de ejecución.

## PANEL DE DETALLES

Detalles	Código	Problemas	...								
Bloque: Dv_01											
<table border="1"> <thead> <tr> <th>Nombre</th> <th>Tipo</th> </tr> </thead> <tbody> <tr> <td>variable_a</td> <td>Integer</td> </tr> <tr> <td>variable_b</td> <td>String</td> </tr> <tr> <td>variable_c</td> <td>Float</td> </tr> </tbody> </table>				Nombre	Tipo	variable_a	Integer	variable_b	String	variable_c	Float
Nombre	Tipo										
variable_a	Integer										
variable_b	String										
variable_c	Float										
+											
Bloque: UI_01											
<table border="1"> <thead> <tr> <th>Variables disponibles</th> </tr> </thead> <tbody> <tr> <td>variable_a</td> </tr> <tr> <td>variable_b</td> </tr> <tr> <td>variable_c</td> </tr> </tbody> </table>		Variables disponibles	variable_a	variable_b	variable_c	<table border="1"> <thead> <tr> <th>Texto (opcional)</th> </tr> </thead> <tbody> <tr> <td>...</td> </tr> </tbody> </table>		Texto (opcional)	...		
Variables disponibles											
variable_a											
variable_b											
variable_c											
Texto (opcional)											
...											

Fig. 5.7. Mockup: Panel de detalles (1), mostrando las pantallas para los bloques de declaración y entrada.

Detalles	Código	Problemas	...					
Bloque: Av_01								
<table border="1"> <thead> <tr> <th>Variables a asignar [izquierda]</th> </tr> </thead> <tbody> <tr> <td>variable_a</td> </tr> <tr> <td>variable_b</td> </tr> <tr> <td>variable_c</td> </tr> </tbody> </table>				Variables a asignar [izquierda]	variable_a	variable_b	variable_c	
Variables a asignar [izquierda]								
variable_a								
variable_b								
variable_c								
<table border="1"> <thead> <tr> <th>Valor [derecha]</th> </tr> </thead> <tbody> <tr> <td>Expresión</td> </tr> <tr> <td>.....</td> </tr> </tbody> </table>		Valor [derecha]	Expresión	.....	<table border="1"> <thead> <tr> <th>Breakpoint</th> </tr> </thead> <tbody> <tr> <td><input type="checkbox"/></td> </tr> </tbody> </table>		Breakpoint	<input type="checkbox"/>
Valor [derecha]								
Expresión								
.....								
Breakpoint								
<input type="checkbox"/>								
<table border="1"> <thead> <tr> <th>Variables/Funciones/Hardcoded</th> </tr> </thead> <tbody> <tr> <td>variable_a</td> </tr> <tr> <td>String</td> </tr> </tbody> </table>		Variables/Funciones/Hardcoded	variable_a	String	<table border="1"> <thead> <tr> <th>Color de fondo</th> </tr> </thead> <tbody> <tr> <td>Color</td> </tr> </tbody> </table>		Color de fondo	Color
Variables/Funciones/Hardcoded								
variable_a								
String								
Color de fondo								
Color								
<table border="1"> <thead> <tr> <th>Variables/Funciones/Hardcoded</th> </tr> </thead> <tbody> <tr> <td>+ - / * ...</td> </tr> </tbody> </table>		Variables/Funciones/Hardcoded	+ - / * ...	<table border="1"> <thead> <tr> <th>Color de texto</th> </tr> </thead> <tbody> <tr> <td>Color</td> </tr> </tbody> </table>		Color de texto	Color	
Variables/Funciones/Hardcoded								
+ - / * ...								
Color de texto								
Color								
		<table border="1"> <thead> <tr> <th>Tamaño</th> </tr> </thead> <tbody> <tr> <td>Width</td> <td>Height</td> </tr> </tbody> </table>		Tamaño	Width	Height		
Tamaño								
Width	Height							
		<table border="1"> <thead> <tr> <th>Posición</th> </tr> </thead> <tbody> <tr> <td>X</td> <td>Y</td> </tr> </tbody> </table>		Posición	X	Y		
Posición								
X	Y							

Fig. 5.8. Mockup: Panel de detalles (2), mostrando la pantalla para el bloque de asignación y la pantalla general de propiedades de bloques.

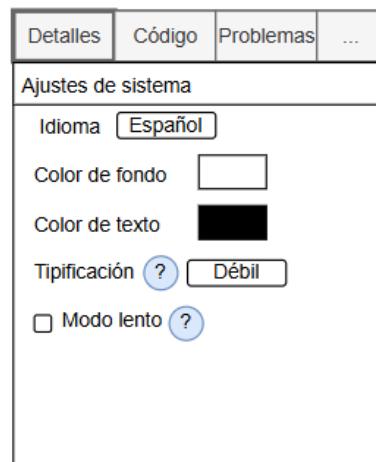


Fig. 5.9. Mockup: Panel de detalles (3), mostrando la pantalla para los ajustes de sistema.

### PANEL DE CÓDIGO

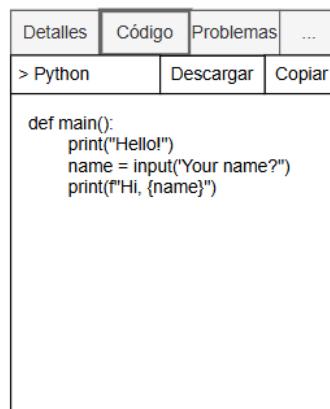


Fig. 5.10. Mockup: Panel de código.

### PANEL DE EJERCICIOS

The mockup shows a dashboard for exercises. On the left is a list of exercises with their status and completion percentage:

Título	Dificultad	Solución	%C
Hello_World	Fácil	Sí	100%
Test1	Difícil	Sí	39.52%
Test2	Medio	No	63.73%

On the right, details for the selected exercise ('Hello\_World') are shown:

- Título:** Hello\_World. Fácil
- Descripción:** lorem ipsum ddadadsadas dsad sadsad sadsa das dsa dsa das dsa dsa das
- Imagen:** [Imagen]
- Estado:** 0/67 test cases passed
- Opciones:** Solucionar (blue button), Ver solución (red button)

Fig. 5.11. Mockup: Panel de ejercicios, mostrando el listado de ejercicios y los detalles de un ejercicio seleccionado.

## PANEL DE COLABORACIÓN

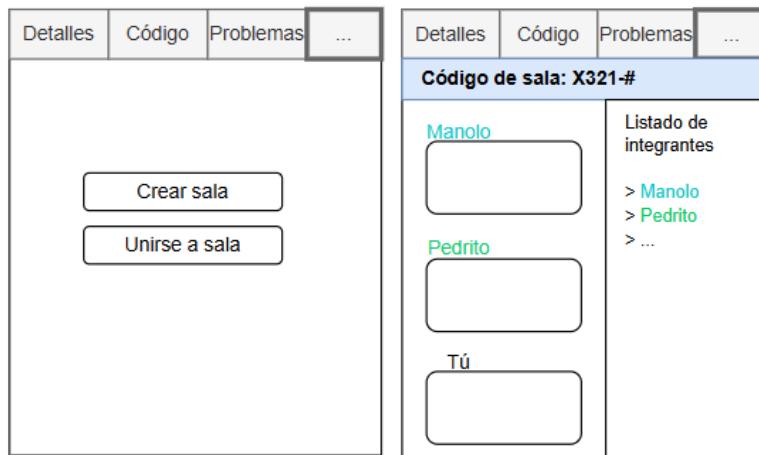


Fig. 5.12. Mockup: Panel de colaboración (1), mostrando la pantalla inicial y la pantalla de una sala desde la perspectiva del creador.

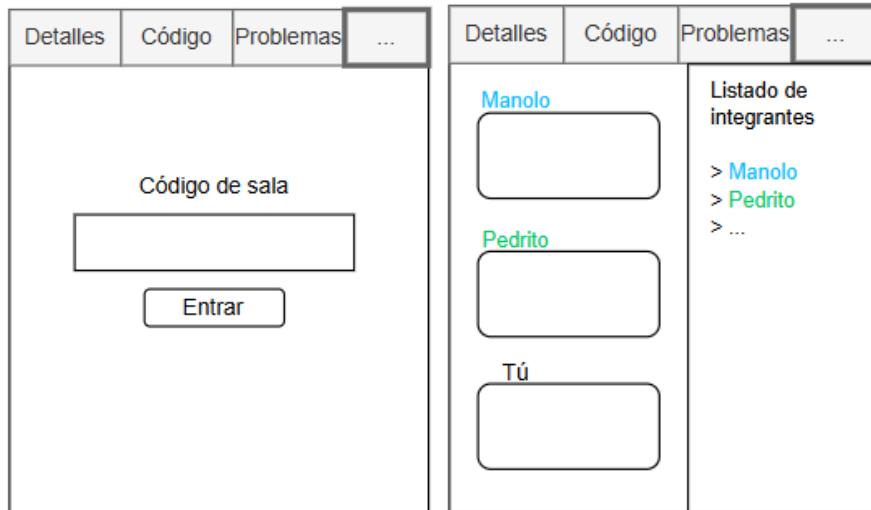


Fig. 5.13. Mockup: Panel de colaboración (2), mostrando la pantalla para unirse a una sala y la pantalla de una sala desde la perspectiva de un participante.

## 5.2 Identidad visual: logo y nombre

Para dotar al proyecto de una identidad propia y memorable, se llevó a cabo un proceso de *branding* que incluyó la elección de un nombre y el diseño de un logotipo. El nombre seleccionado para la aplicación es “**Flowming**”, una fusión de las palabras en inglés “*Flow*” (flujo) y “*Programming*” (programación), que encapsula el propósito central de la herramienta: aprender a programar a través de diagramas de flujo (no confundir con flujos de datos).

El proceso de creación del logotipo combinó el diseño conceptual a mano con herramientas de Inteligencia Artificial Generativa:

1. **Boceto conceptual:** El proceso comenzó con la creación de un boceto manual (Figura 5.14) que establecía la idea fundamental del logotipo: una tipografía fluida y manuscrita que integrara una flecha para simbolizar el flujo.
2. **Generación asistida por IA:** A partir de este boceto, se utilizó un modelo multimodal de lenguaje de IA para refinar la idea en un *prompt* descriptivo y detallado. Este *prompt* fue luego introducido en un modelo de generación de imágenes por IA para explorar múltiples variaciones visuales que materializaran y refinaran el boceto inicial.
3. **Refinamiento y vectorización:** Tras varias iteraciones, se seleccionó la imagen generada que mejor se alineaba con la visión original. Esta imagen fue importada al software de diseño vectorial Inkscape, donde se extrajo, modificó y refinó manualmente para obtener la versión final del logotipo (Figura 5.15). Este paso fue crucial para ajustar detalles finos, asegurar la escalabilidad del diseño y obtener un resultado profesional.



Fig. 5.14. Boceto inicial del logotipo.

El diseño del logotipo (Figura 5.15) busca transmitir los valores de simplicidad, modernidad y dinamismo. Este usa una tipografía manuscrita que confiere un carácter amigable, alineado con el enfoque pedagógico de la herramienta. El color azul principal transmite confianza y profesionalidad, mientras que la flecha integrada en la palabra “Flowming” refuerza visualmente los conceptos de “flujo” y “dirección”, elementos clave en un diagrama de flujo.



Fig. 5.15. Logotipo final de la aplicación.

Adicionalmente, se diseñó un favicon (Figura 5.16) para su uso en las pestañas de navegador y como ícono de la aplicación. Este ícono es una versión simplificada y

encapsulada del elemento de flecha del logotipo principal, garantizando su legibilidad en tamaños reducidos y manteniendo la coherencia visual de la marca.

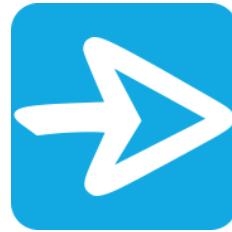


Fig. 5.16. Favicon (ícono) de la aplicación.

### 5.3 Prototipo funcional y evolución del diseño web

La fase de diseño culminó con la implementación de un prototipo funcional en la web, traduciendo los mockups y la identidad visual a una aplicación interactiva. El desarrollo de la interfaz se realizó de forma iterativa, partiendo de una versión inicial que implementaba la estructura básica y evolucionando hacia una versión final más refinada y completa.

#### 5.3.1 Versión inicial del prototipo

La primera versión del prototipo (Figura 5.17) se centró en establecer la estructura fundamental de la aplicación, validando la disposición de los paneles descrita en los mockups. En esta etapa, el diseño era minimalista y funcional, utilizando componentes básicos y estándar para asegurar que la arquitectura fuera sólida.

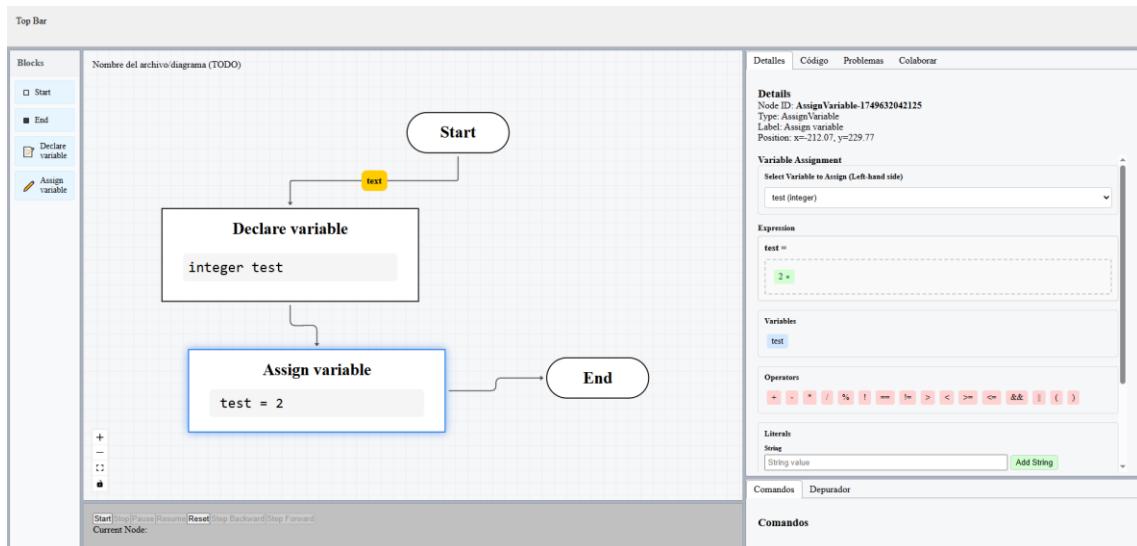


Fig. 5.17. Vista de la versión inicial del prototipo web.

Esta versión inicial ya implementaba las funcionalidades básicas:

- Un lienzo para renderizar nodos y conectores.
- Una paleta de bloques a la izquierda para añadir elementos.
- El panel lateral de Detalles para los bloques de declaración y asignación.
- Controles de ejecución básicos en la parte inferior.

El objetivo de esta fase era establecer un esqueleto funcional sobre el cual construir y refinar las características más complejas.

### 5.3.2 Diseño final implementado

La versión final del prototipo (Figura 5.18) representa una evolución significativa, incorporando la identidad visual completa de “Flowming” y refinando cada componente para mejorar la usabilidad y la estética, alineándose estrechamente con los mockups de alta fidelidad.

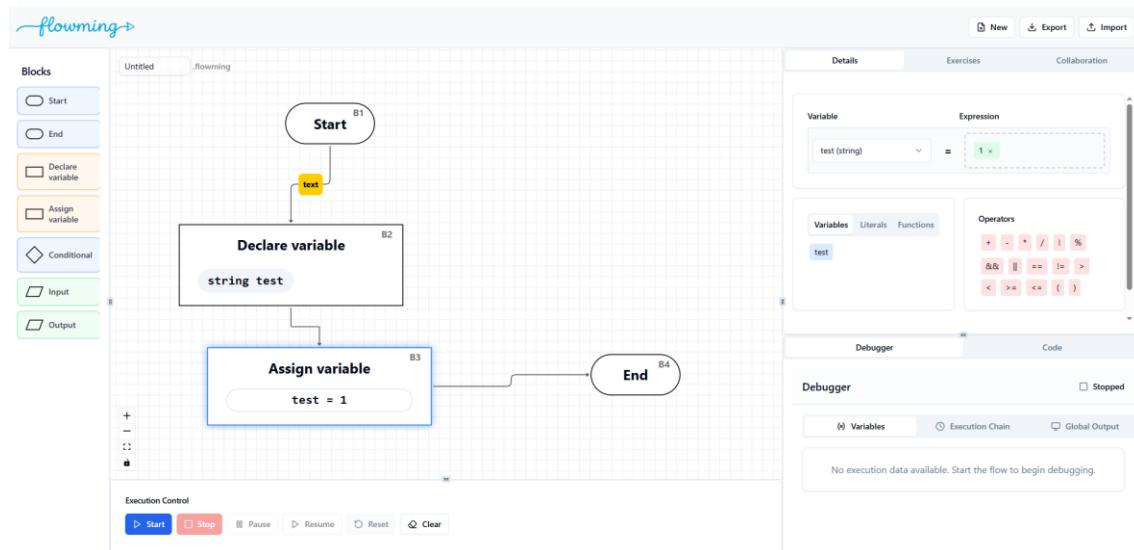


Fig. 5.18. Interfaz de la versión final del prototipo web.

Las mejoras y características implementadas en esta versión final incluyen:

- **Estilo visual coherente:** Los bloques, botones y paneles presentan un diseño unificado y moderno.
- **Iconografía y usabilidad:** Se utiliza una iconografía clara y consistente en toda la aplicación, y los controles se agrupan de manera lógica para un acceso intuitivo. El nombre del archivo y las opciones de importación/exportación están claramente visibles en la barra superior.

- **Rediseño a partir de retroalimentación:** Tras una revisión de usabilidad, se decidió eliminar el panel de Comandos. Se consideró que centralizar la interacción en el lienzo mejoraba el flujo de trabajo. Por ello, la entrada de usuario se realiza ahora a través de ventanas emergentes (*popups*) que aparecen en el momento necesario, y las salidas o errores se muestran como nodos informativos directamente en el lienzo. Además, tras recibir *feedback* del tutor, se remodeló la disposición de los elementos para la creación de expresiones en el bloque de Asignación.

Esta evolución desde un prototipo básico hasta una interfaz pulida y funcional demuestra la aplicación de un ciclo de desarrollo iterativo, donde la funcionalidad y la experiencia de usuario se construyen y mejoran progresivamente. El resultado es una aplicación que no solo cumple con los requisitos funcionales, sino que también ofrece un entorno de aprendizaje agradable y profesional.

## 6 ARQUITECTURA DEL SISTEMA

Este capítulo detalla la arquitectura de alto nivel del sistema, describiendo los componentes clave, su interrelación y la infraestructura subyacente que garantiza su funcionamiento, escalabilidad y mantenibilidad, además del flujo de trabajo de integración y despliegue continuo (CI/CD) que automatiza la publicación de nuevas versiones de la aplicación.

### 6.1 Arquitectura general

La solución sigue un modelo de arquitectura cliente-servidor para la entrega de la aplicación y un modelo *Peer-To-Peer* (P2P – Red entre pares) para la funcionalidad de colaboración. La arquitectura se compone de tres elementos principales: el cliente, el servidor web y un servidor de señalización para la colaboración en tiempo real.

- **Aplicación Frontend (Cliente):** Es una aplicación de página única (SPA – *Single Page Application*) desarrollada en React y empaquetada con Vite. Se ejecuta íntegramente en el navegador del usuario, gestionando toda la lógica de la interfaz, la interacción con el lienzo de diagramas de flujo y la comunicación directa con otros participantes durante una sesión colaborativa a través de la tecnología WebRTC.
- **Servidor web:** Un servidor Nginx se encarga de servir los archivos estáticos (HTML, CSS, JavaScript, imágenes) que componen la aplicación React al cliente. Nginx fue elegido por su alto rendimiento y eficiencia en la entrega de contenido estático, además de por su simplicidad de instalación y configuración.
- **Servidor de señalización:** Para habilitar la colaboración P2P en tiempo real a través de WebRTC, se ha desplegado un servidor de señalización simple utilizando Node.js y el protocolo WebSockets. Su única función es facilitar el “apretón de manos” (*handshake*) inicial entre los usuarios que desean colaborar. Cuando los usuarios se unen a una sesión, este servidor les permite intercambiar los metadatos de conexión necesarios para establecer un canal de comunicación directo. Una vez establecida la conexión WebRTC, toda la información sobre las modificaciones en el diagrama se transmite directamente de un navegador a otro, sin volver a pasar por el servidor, lo que garantiza una latencia mínima y reduce la carga y complejidad del servidor.

La interacción entre estos componentes se puede visualizar en la Figura 6.1.

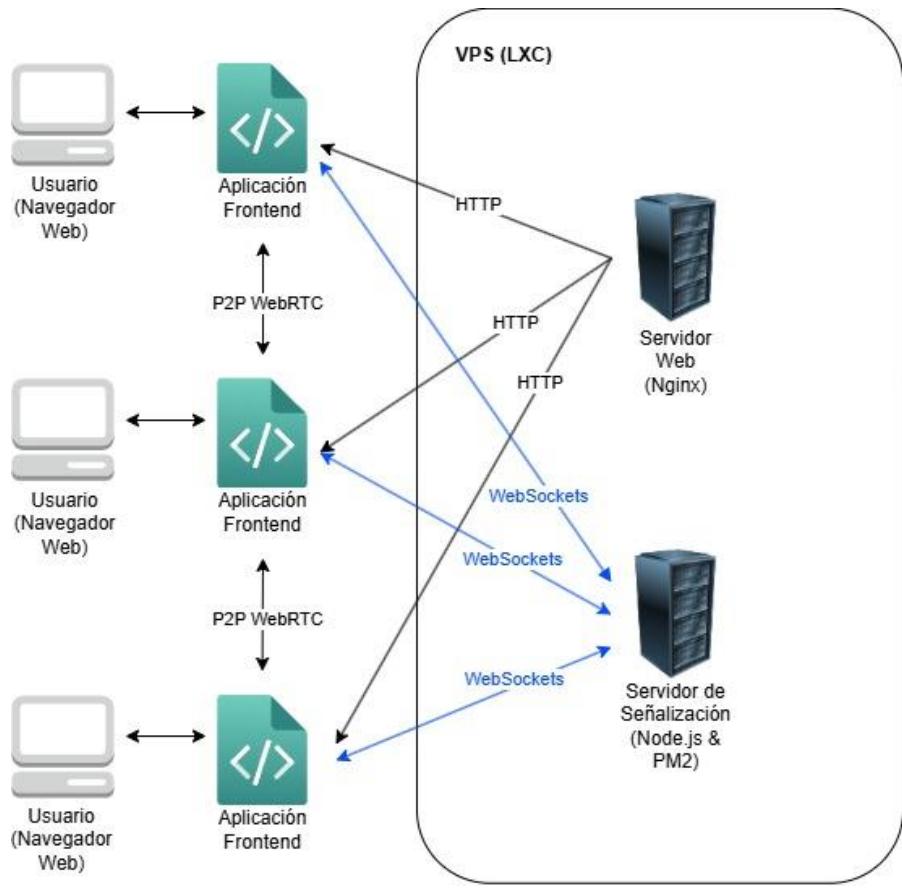


Fig. 6.1. Diagrama de la arquitectura general del sistema.

La elección de esta arquitectura P2P con WebRTC, en lugar de un modelo centralizado donde el servidor retransmite todos los datos, es una decisión de diseño que ofrece las siguientes ventajas:

- Rendimiento y baja latencia:** La comunicación directa elimina al servidor como intermediario para los datos de colaboración, minimizando la latencia de la red, de forma que las acciones de un usuario se reflejan de forma casi instantánea para los demás en condiciones de red óptimas. Esta minimización de la latencia es crucial para la usabilidad y la fluidez de la herramienta.
- Escalabilidad y eficiencia del servidor:** Al delegar la tarea de retransmisión de datos a los propios clientes, la carga del servidor se reduce drásticamente. Su responsabilidad se limita a orquestar la conexión inicial, lo que lo convierte en un servicio muy ligero, más económico de mantener y capaz de gestionar un mayor número de sesiones concurrentes sin convertirse en un cuello de botella.
- Simplicidad en lógica del servidor:** Un servidor centralizado requeriría una implementación más compleja para gestionar usuarios y la sincronización de datos. Dado el alcance y los plazos del proyecto, este enfoque permitió concentrar el esfuerzo de desarrollo en la aplicación Frontend, que es el núcleo

de la experiencia de usuario, en lugar de en la implementación de un backend complejo y con estado.

## 6.2 Arquitectura del servidor

La aplicación se aloja en una Máquina Virtual Privada (VPS – *Virtual Private Server*) proporcionada por la universidad, basada en la tecnología de contenedores de Linux (LXC – *Linux Container*). Este entorno proporciona un sistema operativo aislado y los recursos necesarios para ejecutar los servicios de la aplicación. Dentro de esta máquina se ejecutan los dos servicios de *backend*:

1. **Servidor web Nginx:** Configurado para servir el directorio de compilación generado por Vite. Cualquier petición al dominio es respondida con el “index.html” de la aplicación, permitiendo que React gestione el enrutamiento del lado del cliente.
2. **Servidor de señalización para WebRTC:** Este servicio, desarrollado en Node.js, se ejecuta de forma persistente gracias al gestor de procesos PM2. Este gestor asegura que el servidor se mantenga en funcionamiento de manera continua para facilitar los *handshakes* de nuevas sesiones colaborativas, reiniciándolo automáticamente en caso de fallo y simplificando su gestión.

## 6.3 Gestión de código y despliegue continuo (CI/CD)

Para garantizar un ciclo de desarrollo ágil, robusto y automatizado, se ha implementado un flujo de trabajo de Integración y Despliegue Continuo (CI/CD) mediante GitHub y GitHub Actions. El código fuente del proyecto se gestiona en un repositorio de GitHub, utilizando un sistema de ramas para organizar el desarrollo. Las nuevas funcionalidades se integran en la rama principal a través de un “*Pull Request*”, lo que permite una revisión del código antes de la fusión.

El flujo de CI/CD se divide en dos fases automatizadas principales: integración continua (CI – *Continuous Integration*) y despliegue continuo (CD – *Continuous Deployment*).

### 6.3.1 Integración continua y pruebas automatizadas (CI)

El objetivo de la integración continua es verificar la calidad del código y prevenir la introducción de errores en la base de código principal. Para ello, se ha configurado un flujo de trabajo de GitHub Actions que ejecuta una suite de pruebas automatizadas sobre los componentes más críticos de la lógica de la aplicación: el analizador sintáctico (*parser*) de expresiones y el generador de código.

Este flujo de trabajo se dispara automáticamente en dos momentos clave:

1. Al crear o actualizar un “*Pull Request*” que apunta a la rama principal. Esto actúa como una barrera de calidad, impidiendo que se fusioné código que no supere las pruebas.
2. Al realizar una modificación o “*commit*” directo a la rama principal. Esto sirve como una última verificación para mantener la integridad de la rama principal en todo momento.

Si alguna de las pruebas falla, GitHub marca el “*Pull Request*” o el “*commit*” como fallido, notificando al desarrollador para que corrija los problemas antes de continuar.

### 6.3.2 Despliegue continuo (CD)

Una vez que el código ha sido fusionado en la rama principal y ha superado con éxito todas las pruebas de integración, se inicia la fase de despliegue continuo. Este proceso automatiza la publicación de la nueva versión de la aplicación en el servidor de producción.

Debido a las restricciones de red de la universidad, que requieren el uso de una VPN para el acceso externo, se ha instalado un GitHub Runner autoalojado (*self-hosted*) directamente en la VPS. Este “*runner*” es el encargado de ejecutar el proceso de despliegue. El flujo de despliegue automatizado, que se activa tras una actualización exitosa a la rama principal, sigue los siguientes pasos:

1. GitHub Actions asigna un nuevo trabajo al “*runner*” autoalojado en la VPS.
2. El “*runner*” obtiene la última versión del código de la rama principal.
3. Ejecuta los comandos de construcción del proyecto para generar los archivos estáticos optimizados para la aplicación Frontend.
4. Una vez finalizada la construcción, el “*runner*” mueve los archivos del directorio de compilación al directorio raíz que sirve Nginx y reinicia este servicio.
5. La nueva versión de la aplicación queda instantáneamente disponible para todos los usuarios sin necesidad de intervención manual.

Esta arquitectura CI/CD completa garantiza que cada cambio en la aplicación sea validado automáticamente, reduciendo el riesgo de errores y permitiendo entregar nuevas funcionalidades de manera rápida y fiable.

## 7 TECNOLOGÍAS EMPLEADAS Y DETALLES DE LA IMPLEMENTACIÓN

Este capítulo describe los aspectos prácticos de la construcción de la aplicación. Primero se presentan y justifican las tecnologías y herramientas clave seleccionadas para el desarrollo del proyecto. A continuación, se profundizará en los detalles de diseño e implementación de los componentes más complejos del sistema, así como la metodología de gestión y la estructura del código fuente.

### 7.1 Gestión del proyecto y control de versiones

La gestión del proyecto se ha centrado en la agilidad y la trazabilidad, utilizando GitHub como plataforma central para el control de versiones, la gestión de tareas y la automatización. El control de versiones se ha realizado mediante Git, siguiendo el flujo de trabajo basado en ramas. Las nuevas funcionalidades y correcciones se desarrollaron en ramas aisladas para no comprometer la estabilidad de la rama principal. La integración de estos cambios se gestionó a través de “*Pull Requests*”, lo que facilitó la revisión del código y aseguró que solo el código verificado y funcional se fusionara.

Aunque el flujo de integración y despliegue continuo (CI/CD) ya se ha detallado en la [sección 6.3](#), cabe destacar su rol en la gestión del proyecto. La ejecución automática de pruebas en cada “*commit*” y “*Pull Request*” mediante GitHub Actions sirvió como un mecanismo de control de calidad continuo, permitiendo identificar errores de forma temprana y agilizar el ciclo de desarrollo.

### 7.2 Tecnologías empleadas

La elección de las tecnologías ha sido un factor crucial para cumplir con los requisitos funcionales y no funcionales del proyecto, así como para garantizar una experiencia de desarrollo productiva. A continuación, se detallan y justifican las herramientas clave utilizadas.

- **TypeScript:** Se eligió TypeScript sobre JavaScript estándar para dotar al proyecto de un sistema de tipado estático. Esta decisión fue fundamental para mejorar la robustez del código, facilitar el mantenimiento a largo plazo y reducir errores en tiempo de ejecución. La verificación de tipos que ofrece TypeScript mejoró la confianza en la lógica implementada, especialmente en componentes complejos como el motor de ejecución y el generador de código.
- **React y Vite:** Se seleccionó React como la biblioteca para construir la interfaz de usuario debido a su popularidad, su extenso ecosistema y su modelo de desarrollo basado en componentes, que facilita la creación de interfaces complejas y reutilizables [60]. Para el empaquetado y el servidor de desarrollo, se optó por Vite. A diferencia de herramientas tradicionales, Vite ofrece

tiempos de arranque y recarga en caliente casi instantáneos, lo que se traduce en una experiencia de desarrollo mucho más fluida y productiva [61].

- **React Flow:** Para la implementación del lienzo de diagramas de flujo, se eligió la biblioteca React Flow. Esta herramienta proporciona una solución completa y altamente personalizable para renderizar y manipular gráficos basados en nodos y aristas. Ofrece funcionalidades esenciales ya implementadas, como el arrastre de nodos, el zoom y la gestión de conexiones, permitiendo concentrar el esfuerzo en la lógica específica de los nodos del diagrama de flujo en lugar de en la implementación de un lienzo desde cero [46].
- **shadcn/ui y Lucide React:** La interfaz de usuario se ha construido utilizando una colección de componentes de shadcn/ui. A diferencia de las librerías de componentes tradicionales, shadcn/ui permite copiar y personalizar los componentes directamente en el código del proyecto, ofreciendo un control total sobre su estilo y comportamiento [32]. Esto, combinado con la librería de iconos Lucide React, ha permitido crear una interfaz de usuario moderna, consistente y accesible de forma muy eficiente [62].
- **Yjs y y-webrtc:** La funcionalidad de colaboración en tiempo real se ha implementado usando Yjs, una biblioteca de alto rendimiento para estructuras de datos replicadas sin conflictos (CRDTs). Yjs permite que múltiples usuarios modifiquen un documento compartido de forma concurrente sin necesidad de un servidor central que resuelva los conflictos [40]. A diferencia de otras soluciones basadas en la Transformación Operacional (OT), como ShareDB, que requieren un servidor central para mediar y transformar cada operación, el enfoque CRDT de Yjs simplifica drásticamente la arquitectura del *backend*. Esta decisión fue un factor decisivo, ya que permitió evitar la complejidad de implementar un servidor con estado, alineándose mejor con los plazos y el alcance del proyecto. Para la comunicación entre los clientes, se utilizó el proveedor y-webrtc, que integra Yjs con la tecnología WebRTC, permitiendo sincronizar el estado del diagrama de flujo directamente entre los navegadores de los usuarios y logrando una colaboración P2P de muy baja latencia.
- **Vitest:** Para las pruebas automatizadas, se seleccionó Vitest como el *framework* de *testing*. Al estar construido sobre Vite, se integra perfectamente en el entorno de desarrollo del proyecto, ofreciendo una ejecución de pruebas rápida [5]. Su rendimiento y facilidad de uso han sido claves para implementar la suite de pruebas del analizador sintáctico de expresiones y el generador de código de forma eficiente.

### 7.3 Estructura del proyecto

Una organización de código clara y mantenible ha sido un pilar fundamental durante el desarrollo. El proyecto se ha estructurado en directorios con responsabilidades bien definidas, facilitando la navegación, la reutilización de componentes y las futuras ampliaciones. El repositorio principal se divide en tres directorios raíz:

- `y-webrtc-server/`: Contiene el código del servidor de señalización. Dado su rol específico y limitado, consta de un archivo único “`server.js`” que implementa el servidor de WebSocket para el *handshake* de WebRTC.
- `flowming/`: Alberga todo el código fuente de la aplicación frontend desarrollada con React y Vite.
- `.github/workflows/`: Contiene los archivos YAML (“`deploy.yaml`”, “`test.yml`”) que definen los flujos de trabajo de CI/CD de GitHub Actions.

A continuación, se detalla la estructura del directorio “`flowming/`”, que constituye el núcleo del proyecto.

#### 7.3.1 Estructura a nivel raíz

El directorio raíz de la aplicación frontend contiene los archivos de configuración estándar para un proyecto Vite con TypeScript, como “`vite.config.ts`”, “`tsconfig.json`” y “`package.json`”. Los directorios más relevantes son:

- `public/`: Almacena los archivos estáticos, como el logotipo y los favicons.
- `src/`: Es el directorio principal que contiene todo el código fuente de la aplicación.

#### 7.3.2 Estructura del directorio “`src/`”

La organización interna del directorio “`src/`” se ha diseñado siguiendo el principio de separación de responsabilidades y modularidad.

- `components/`: Este es el directorio más importante y contiene todos los componentes de React que conforman la interfaz de usuario. Está subdividido por funcionalidad para mantener la organización:
  - `Flow/`: Agrupa los componentes relacionados con el lienzo interactivo.
    - `FlowContent.tsx`: Componente central que renderiza el lienzo de ReactFlow y gestiona las interacciones principales como la adición de nodos.
    - `Nodes/`: Contiene la definición de cada tipo de nodo del diagrama de flujo (Inicio, Fin, Asignación, Declaración, Condicional, Entrada, Salida). Cada componente de nodo no

solo define su apariencia visual, sino también la lógica de ejecución que se invoca cuando el motor alcanza dicho nodo.

- Panel/: Incluye los componentes del panel lateral derecho.
  - Tabs/: Contiene cada una de las pestañas del panel. Destaca la carpeta “editors/”, que alberga los formularios específicos para editar las propiedades de cada tipo de nodo en el panel de Detalles (“DetailsTab”).
- Toolbar/: Componentes de la barra de herramientas lateral izquierda desde donde se arrastran los nuevos nodos al lienzo.
- ui/: Directorio donde residen los componentes base de shadcn/ui, permitiendo su personalización directa dentro del proyecto.
- context/: Contiene los proveedores de contexto de React, utilizados para la gestión del estado global de la aplicación. Se ha optado por un enfoque de múltiples contextos para separar estados con diferentes responsabilidades, evitando así renderizados innecesarios y manteniendo la lógica de estado desacoplada y organizada.
- hooks/: Alberga los “hooks” personalizados de React, que encapsulan lógica con estado y efectos secundarios para ser reutilizada en diferentes componentes.
  - useFlowExecutor.ts: Proporciona funciones para controlar el ciclo de vida de la ejecución, gestionando el estado interno del motor.
- models/: Define las estructuras de datos y clases centrales para la lógica del negocio.
  - Variables.ts y ValuedVariable.ts: Representan el modelo de una variable. “Variable” se usa para la declaración inicial, mientras que “ValuedVariable” representa la variable con su valor que se pasa durante la ejecución.
  - Expression.ts: Define la estructura de una expresión, que consiste en una secuencia de elementos (“ExpressionElement”) como variables, operadores o valores literales.
  - ExpressionParser.ts: Implementa el analizador sintáctico (*parser*) que convierte una secuencia de elementos de expresión en un Árbol de Sintaxis Abstracta (AST), utilizado por “Expression” para su evaluación y por “codeGeneration” para la generación de código.
- data/: Almacena los datos estáticos que definen el contenido del módulo de ejercicios. Esta arquitectura guiada por datos permite añadir nuevos problemas, casos de prueba o soluciones simplemente modificando archivos JSON, sin necesidad de alterar el código de la aplicación.

- exercises.json: Archivo central que define la lista de ejercicios, incluyendo sus metadatos (identificador, título, dificultad, descripción) y sus correspondientes casos de prueba.
- solutions/: Directorio que contiene los diagramas de solución en formato JSON. Cada archivo se corresponde con un identificador de ejercicio definido en “exercises.json”, permitiendo al sistema localizar y cargar la solución de forma automática.
- utils/: Alberga funciones de utilidad reutilizables y la lógica más compleja y desacoplada de la interfaz.
  - codeGeneration.ts: Implementa la lógica que recorre la estructura de datos del AST del diagrama y sus expresiones para generar el código fuente equivalente en Python.
  - flowExecutorUtils.ts: Contiene el núcleo del motor de ejecución del diagrama de flujo.
  - flowTestRunner.ts: Implementa un motor de pruebas optimizado. Este motor ejecuta de forma síncrona el diagrama del usuario contra los casos de prueba de un ejercicio, omitiendo elementos visuales como animaciones o retardos, para obtener y validar los resultados de forma eficiente.
- tests/: Contiene los archivos de pruebas unitarias implementados con Vitest para validar el correcto funcionamiento de componentes críticos, como el analizador sintáctico de expresiones y el generador de código.

Esta estructura modular ha facilitado el desarrollo y la depuración, y sienta las bases para futuras expansiones de la herramienta de una manera ordenada y escalable.

## 7.4 Detalles clave de la implementación

Más allá de la elección de tecnologías, el núcleo funcional de la aplicación requirió el diseño e implementación de varias lógicas complejas. Esta sección detalla algunas de las decisiones de implementación más significativas que definen el comportamiento del motor de ejecución y el generador de código.

### 7.4.1 Procesamiento de expresiones mediante AST

A diferencia de los entornos de programación textuales, donde las expresiones se escriben como una cadena de texto, este proyecto adopta un enfoque visual. El usuario construye las expresiones de forma interactiva, arrastrando y soltando elementos como variables, valores literales, operadores y funciones de conversión de tipo en un área de edición designada para formar una secuencia plana de componentes de expresión (“ExpressionElement”). Aunque este enfoque previene errores sintácticos básicos, la

secuencia resultante carece de estructura jerárquica y podría ser sintácticamente inválida (por ejemplo, contener dos operadores consecutivos).

Para interpretar esta secuencia y validar su corrección, se ha implementado un analizador sintáctico de descenso recursivo (*recursive descent parser*). La función principal de este *parser* es transformar la secuencia plana en un Árbol de Sintaxis Abstracta (AST). Durante este proceso de construcción del árbol, el *parser* no solo aplica las reglas de precedencia y asociatividad de los operadores, sino que también valida que la expresión esté bien formada. Si la secuencia no puede ser convertida a un AST válido, se detecta un error estructural.

El uso de esta arquitectura de dos pasos (construcción visual seguida de un análisis sintáctico a AST) ofrece ventajas cruciales:

1. **Validación y estructura definida:** El AST se convierte en la representación de una expresión válida. El proceso de *parsing* garantiza que solo las expresiones bien formadas sean procesadas por el resto del sistema, sirviendo como una capa robusta de validación.
2. **Fuente única de verdad lógica:** El *parser* centraliza todas las reglas de la lógica de expresiones. Asegura que las expresiones se interpreten consistentemente tanto para la evaluación como para la generación de código.
3. **Flexibilidad y separación de responsabilidades:** Una vez obtenido el AST, el motor de ejecución puede recorrerlo para evaluar la expresión con un sistema de tipado fuerte, mientras que el generador de código lo recorre para traducirlo a la sintaxis de un lenguaje específico, manteniendo ambos módulos desacoplados.

#### 7.4.2 Sistema de tipado y gestión de variables

Para fomentar buenas prácticas de programación y simplificar la lógica interna, se tomaron varias decisiones de diseño respecto a la gestión de variables y sus tipos.

- **Bloque de Declaración explícito:** En la aplicación es obligatorio declarar las variables previamente en un bloque de “Declaración”. Esta decisión se justifica por dos razones:
  - Valor pedagógico: Introduce a los estudiantes a la buena práctica de definir explícitamente las variables y sus tipos antes de su uso.
  - Claridad y usabilidad: Centralizar la declaración de variables evita la ambigüedad sobre su alcance. El usuario tiene una visión clara de las variables disponibles en todo el diagrama.
- **Tipado fuerte y conversión explícita en Entrada:** El motor de ejecución implementa un sistema de tipado fuerte (*strongly typed*). Para facilitar la interacción, los nodos de “Entrada” realizan una conversión de tipo explícita y

automática al tipo de variable de destino. La implementación de un sistema de tipado débil (*weakly typed*) se consideró una mejora futura, pero se descartó en la versión actual debido a su complejidad y a la falta de consistencia entre cómo los diferentes lenguajes de tipado débil manejan ciertas conversiones y operaciones.

#### 7.4.3 Implementación de listas

Para el manejo de colecciones de datos, se ha implementado el tipo de dato de lista (*array*) con las siguientes características de diseño:

- Tamaño fijo e indexación base cero: Al declarar una lista, el usuario especifica un tamaño fijo. El acceso a los elementos se realiza mediante un índice que comienza en 0 (*zero-based indexing*).
- Operaciones basadas en índices: La manipulación de los datos de la lista se realiza operando directamente sobre sus índices (asignando un valor a una posición o a un rango). No se han incluido operaciones de alto nivel que modifiquen dinámicamente su tamaño.

Esta aproximación, más cercana a la gestión de memoria de bajo nivel, se justifica por su simplicidad de implementación dentro de los plazos del proyecto y por su valor pedagógico, ya que anima a los estudiantes a comprender los fundamentos de la manipulación de listas. Para mantener la funcionalidad, se proporciona una función “length” que devuelve el tamaño de la lista.



## 8 EVALUACIÓN CON USUARIOS

La fase de evaluación es un componente crítico en el ciclo de vida del desarrollo de software, ya que permite validar la usabilidad de la aplicación desde la perspectiva del usuario final. Este capítulo detalla el plan y la metodología que se siguieron para medir la usabilidad de la aplicación y recoger las impresiones de los usuarios sobre su potencial como recurso de aprendizaje. Se describen los instrumentos diseñados, el protocolo de pruebas y, finalmente, se presentarán y discutirán los resultados obtenidos.

### 8.1 Objetivos de la evaluación

El objetivo principal de la evaluación con usuarios externos fue medir la usabilidad general de la aplicación e identificar áreas de mejora en su diseño e interacción. Para ello, se establecieron los siguientes objetivos específicos:

- Evaluar la facilidad de uso y la curva de aprendizaje de la interfaz para la creación, ejecución y depuración de diagramas de flujo.
- Medir la satisfacción y usabilidad percibida de la aplicación a través de métricas estandarizadas.
- Identificar posibles problemas de usabilidad o áreas de confusión en el flujo de trabajo del usuario.
- Recopilar la retroalimentación cualitativa sobre la claridad de las funcionalidades clave y la experiencia de usuario en general.
- Conocer la percepción de los usuarios sobre si la herramienta pudiera serles útiles para comprender los conceptos de la lógica algorítmica.

### 8.2 Diseño de las pruebas y recopilación de datos

Para alcanzar los objetivos propuestos, se diseñó un método de evaluación mixto que combina la recopilación de datos cuantitativos y cualitativos, centrado en la usabilidad. Este se compuso de los siguientes elementos:

- **Tareas de usuario:** Se definieron tres tareas representativas que los participantes debían completar. Estas tareas, de dificultad incremental, fueron diseñadas para guiar al usuario a través de las funcionalidades principales: (1) replicar un diagrama simple, (2) crear un algoritmo para generar los primeros cinco números pares, que requiere el uso de un bucle, una condición y una salida, y (3) utilizar las herramientas de depuración y generación de código. El documento detallado con las instrucciones proporcionadas a los participantes se puede consultar en el [Anexo E](#).

- **Instrumentos de recopilación:** Se utilizó un único cuestionario online, creado con Google Forms, que los participantes completaron tras realizar las tareas. Este cuestionario se estructuró en dos secciones principales:
  1. **Cuestionario de usabilidad (SUS):** Se empleó el cuestionario estándar *System Usability Scale* (SUS) [3] para obtener una puntuación cuantitativa global sobre la usabilidad percibida de la aplicación.
  2. **Cuestionario cualitativo:** Se incluyó un conjunto de preguntas abiertas para recoger opiniones detalladas sobre la experiencia de uso, los aspectos más positivos y las áreas de mejora. Véase el cuestionario de Google Forms en el [Anexo F](#).

### 8.3 Protocolo de evaluación

Debido a limitaciones logísticas y de plazos, se optó por la elaboración de dos protocolos de participación independientes. El objetivo era combinar la amplitud de datos de un grupo numeroso con la profundidad de análisis cualitativo de un grupo más reducido. Los participantes fueron asignados a uno de los dos siguientes protocolos:

- **Evaluación asíncrona:** Dirigido a la mayoría de participantes, este protocolo se centró en la recopilación de datos cuantitativos y cualitativos de forma autónoma. A este grupo de usuarios se le proporcionó el documento de evaluación (véase [Anexo E](#)), que incluía una introducción al proyecto, las tareas a realizar y el enlace al formulario de Google Forms. Los participantes completaron las tareas a su propio ritmo y enviaron sus respuestas a través del cuestionario.
- **Evaluación síncrona:** Un subconjunto de usuarios participó en sesiones individuales por videollamada para un análisis cualitativo en profundidad. A estos participantes se les pidió que compartieran su pantalla y aplicaran el método *think-aloud* (pensar en voz alta) mientras interactuaban con la aplicación. Este enfoque permitió observar directamente sus acciones, dudas y razonamientos, proporcionando un contexto valioso para interpretar los resultados generales. Al igual que el grupo asíncrono, completaron el mismo cuestionario al finalizar las tareas.

### 8.4 Resultados

La evaluación con usuarios resultó en un conjunto de datos tanto cuantitativos como cualitativos que permiten valorar la usabilidad y el potencial pedagógico de la aplicación. En la evaluación participaron un total de 9 usuarios con un perfil diverso en cuanto a su experiencia previa en programación. Este grupo heterogéneo incluyó desde compañeros del Grado en Ingeniería Informática, con conocimientos técnicos avanzados, hasta amigos y familiares con poca o ninguna exposición formal a la programación. Esta diversidad fue intencionada para validar si la herramienta era accesible tanto para los usuarios

principiantes, su público objetivo principal, como comprensible para aquellos con más experiencia. Las respuestas de este grupo se recopilaron a través del formulario de Google Forms y de las notas tomadas durante las sesiones síncronas.

#### 8.4.1 Resultados cuantitativos

El análisis de las respuestas al cuestionario *System Usability Scale* (SUS) resultó en una puntuación media de 88,61 sobre 100. Según la escala curva de puntuación de Sauro y Lewis [19, 29], una puntuación por encima de 84,1 se considera “Lo Mejor Imaginable” y sitúa a la aplicación en el percentil A+, dentro del 4% de los productos con mejor usabilidad percibida.

El desglose del cálculo para cada uno de los 9 participantes, detallado en la Figura 8.1, muestra una notable consistencia en la alta valoración, con puntuaciones individuales que oscilan entre 77,5 y 97,5. Esto indica que la experiencia fue uniformemente positiva para la gran mayoría de los evaluadores, sin que existieran casos atípicos con una percepción negativa. Para un análisis más detallado, la distribución de respuestas para cada pregunta se presenta en el [Anexo H](#).

ID participante	SUS Q1	SUS Q2	SUS Q3	SUS Q4	SUS Q5	SUS Q6	SUS Q7	SUS Q8	SUS Q9	SUS Q10	SUS Total
1	2	2	4	2	4	1	5	2	4	1	77,5
2	4	2	4	2	5	1	5	2	5	1	87,5
3	5	1	4	1	4	1	5	1	5	1	95
4	3	2	4	2	5	1	4	1	3	1	80
5	3	4	4	2	5	2	5	1	5	1	80
6	4	2	4	2	5	1	5	1	5	1	90
7	4	1	5	1	5	1	5	1	4	1	95
8	5	1	4	1	5	1	5	1	4	1	95
9	4	1	5	1	5	1	5	1	5	1	97,5
											TOTAL 88,61

Fig. 8.1. Desglose del cálculo de la puntuación SUS por participante.

#### 8.4.2 Resultados cualitativos

Las respuestas a las preguntas abiertas y las observaciones de las sesiones síncronas proporcionaron un contexto más profundo a los datos cuantitativos. Los hallazgos se pueden agrupar en dos categorías principales:

##### Aspectos más valorados

- El depurador visual fue el elemento más elogiado. Los comentarios destacaron específicamente la animación visual de la ejecución y la funcionalidad del panel de Depuración, lo que demuestra que la visualización del flujo de ejecución y la evolución de las variables son funcionalidades clave y muy apreciadas.
- Los usuarios también valoraron positivamente la claridad general de las funciones y la facilidad de uso del sistema de arrastrar y soltar (*drag-and-drop*).

## Dificultades y áreas de mejora

- Una dificultad recurrente fue la confusión inicial al enfrentarse al diagrama por primera vez. El estado por defecto del diagrama, con un bloque de “Inicio” conectado a uno de “Fin” y una etiqueta en el conector que indicaba “double click me to edit”, generó dudas sobre cómo conectar nuevos nodos. Un participante admitió haber intentado hacer clic directamente en la etiqueta del conector, pensando que esa era la acción necesaria para modificar el flujo. Se sugirió eliminar este mensaje y ofrecer una indicación más clara sobre el mecanismo de conexión entre bloques.
- El editor de expresiones del bloque de Asignación también fue un punto de fricción. Varios usuarios mencionaron dificultades iniciales para entender cómo construir expresiones complejas, especialmente la concatenación de texto y el uso de funciones de conversión de tipo.
- Se identificó como una fuente de gran confusión el modo en el que el sistema asigna etiquetas “Sí” y “No” a las ramas de un bloque Condicional. Para ello, se emplea un patrón de alternancia global, de modo que la primera rama creada es “Sí”, la siguiente es “No”, y así sucesivamente, independientemente de a qué nodo condicional pertenezca. Este comportamiento fue percibido por los usuarios como impredecible, ya que esperaban que cada bloque Condicional tuviera su propio comportamiento de asignación consistente (por ejemplo, que la primera rama de cualquier bloque Condicional fuera siempre “Sí”).
- Finalmente, la sesión síncrona reveló que algunas funcionalidades no eran fácilmente descubribles, como la posibilidad de reordenar elementos en el editor de expresiones.

## 8.5 Discusión de datos

Los resultados de la evaluación son mayoritariamente positivos y validan la hipótesis central del proyecto. La puntuación SUS de **88,61** es un indicador cuantitativo robusto de que la interfaz diseñada es eficaz y satisfactoria.

Sin embargo, el análisis cualitativo revela puntos clave de mejora en la predictibilidad y el control del usuario. La confusión reportada en el punto de partida y, de forma más significativa, en el comportamiento de las ramas condicionales, revela una desconexión entre el modelo de implementación del sistema y el modelo mental del usuario. El patrón de alternancia global para las etiquetas “Sí/No” viola el **Principio de Mínima Sorpresa** (*Principle of Least Astonishment*). Los usuarios esperan que cada componente de la interfaz (en este caso, cada bloque Condicional) se comporte de una manera autónoma y predecible. Al hacer que el estado de una nueva conexión dependa de la última conexión creada en todo el diagrama, el sistema se vuelve opaco e incontrolable para el usuario. Una solución más adecuada sería garantizar un comportamiento consistente por bloque

(la primera conexión es siempre “Sí”) o, idealmente, proporcionar al usuario un control explícito para cambiar la etiqueta de la rama tras su creación.

Esta necesidad de una interacción más predecible se alinea con la confusión reportada en la experiencia inicial. La etiqueta “double click me to edit” en el conector inicial, aunque bien intencionada, resultó ser una instrucción ambigua que desvió la atención del usuario del mecanismo principal de interacción: arrastrar bloques desde la paleta para insertarlos al flujo y conectarlos mediante los *handles* (mangos) presentes en ellos. Ambos hallazgos subrayan que, más allá de hacer las funciones descubribles, es crucial que su comportamiento sea consistente, transparente y controlable para fomentar la confianza del usuario.

Por otro lado, el éxito rotundo del depurador visual confirma una de las hipótesis pedagógicas centrales del proyecto: la retroalimentación visual e inmediata del flujo de ejecución es una herramienta de aprendizaje de gran valor. El hecho de que esta funcionalidad fuera tan apreciada, a pesar de las fricciones iniciales en la creación del diagrama, sugiere que el concepto fundamental de la herramienta es sólido.

En síntesis, la discusión revela que la aplicación es una herramienta conceptualmente fuerte y funcionalmente rica, cuyo potencial se ve parcialmente limitado por problemas de usabilidad específicos en su capa de interacción. La alta puntuación SUS sugiere que, una vez superada la curva de aprendizaje inicial, su experiencia de uso es fluida y satisfactoria.

## 8.6 Conclusión de la evaluación con usuarios

La evaluación con usuarios ha cumplido satisfactoriamente sus objetivos, proporcionando una validación empírica sólida tanto de la usabilidad de la aplicación como de su potencial como recurso educativo.

Se concluye que la aplicación presenta un nivel de usabilidad global excelente, como lo demuestra la puntuación media SUS de 88,61. Este dato cuantitativo confirma que los usuarios, una vez familiarizados con el sistema, encontraron la herramienta eficaz, satisfactoria y fácil de usar. Las características más valoradas, como el depurador visual, ratifican que los pilares conceptuales del proyecto son acertados y potencialmente responden de manera efectiva a las necesidades de los estudiantes principiantes.

Asimismo, la evaluación ha sido fundamental para identificar áreas de mejora clave y accionables que no habrían sido evidentes sin la retroalimentación directa de los usuarios. Los problemas detectados en la experiencia inicial, y de forma notable, la falta de predictibilidad en el comportamiento de las ramas condicionales, constituyen una hoja de ruta clara para futuras iteraciones del desarrollo. Estos hallazgos no invalidan la calidad del producto, sino que demuestran el valor del proceso de evaluación para refinar la interacción y alinearla completamente con el modelo mental del usuario.

En definitiva, la evaluación confirma que este Trabajo de Fin de Grado ha culminado en el desarrollo de una herramienta funcional y bien recibida que aborda eficazmente el

problema planteado. Valida su enfoque como un recurso pedagógico prometedor y, a través de sus hallazgos, proporciona las claves para elevar la experiencia de usuario de la aplicación a un nivel de excelencia que iguale su ya demostrada potencia funcional.

## 9 PLANIFICACIÓN FINAL

Este capítulo presenta una comparativa entre la planificación inicial del proyecto, detallada en la [sección 4.2](#), y la ejecución real del mismo. Como es común en el desarrollo de software, especialmente en un entorno académico, surgieron desviaciones significativas con respecto al cronograma previsto. A continuación, se presenta el diagrama de Gantt que refleja la dedicación temporal real en la Figura 9.1 y se analizan las principales causas de estas desviaciones.

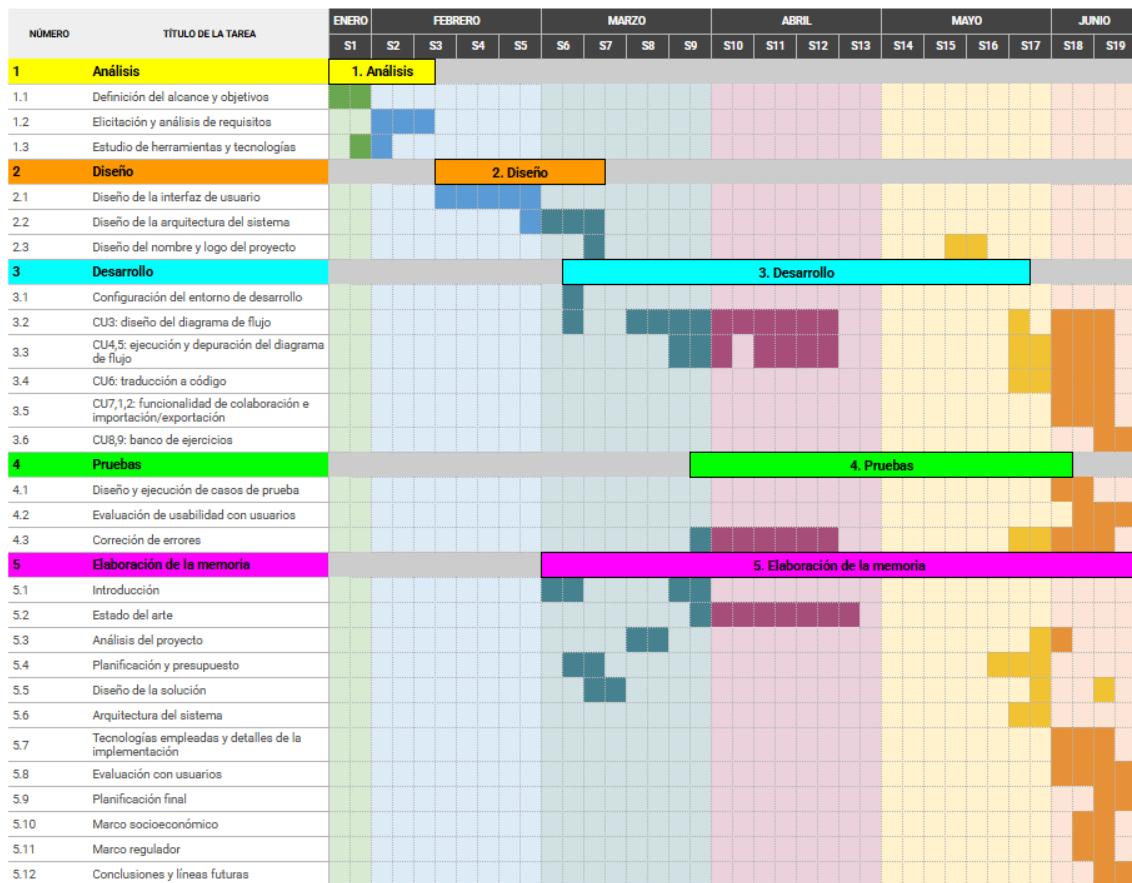


Fig. 9.1. Diagrama de Gantt final que refleja la ejecución real del proyecto.

El análisis de la ejecución real del proyecto revela varios factores interrelacionados que causaron las desviaciones respecto a la planificación inicial:

- 1. Subestimación de la complejidad técnica:** La estimación inicial del esfuerzo subestimó la dificultad real de implementar el núcleo funcional de la aplicación. En particular, la integración del motor de ejecución, el analizador sintáctico (*parser*) de expresiones y los editores visuales para construir dichas expresiones requirió un esfuerzo considerablemente mayor al previsto. La gestión del estado asociado a estos componentes interactivos y la

comunicación entre la lógica pura y la interfaz de usuario resultaron ser tareas más complejas y laboriosas de lo que se había planificado.

2. **Impacto de la carga académica:** La planificación inicial no contempló adecuadamente el impacto de la carga académica concurrente del cuatrimestre. La coincidencia de las fases del desarrollo más intensivas con períodos de exámenes finales y entregas de trabajos de otras asignaturas provocó retrasos inesperados y redujo las horas de dedicación semanales efectivas al proyecto.
3. **Falta de consistencia en la dedicación:** A nivel personal, la gestión del tiempo no siempre fue óptima. Hubo períodos de alta productividad intercalados con otros de menor dedicación, lo que generó una falta de consistencia diaria. Un ritmo de trabajo más consistente y distribuido habría ayudado a mitigar algunos de los retrasos y a mantener un progreso más predecible.

A pesar de estos desafíos, la gestión adaptativa del tiempo y la priorización de los requisitos funcionales permitieron cumplir con los objetivos principales del proyecto y entregar un producto funcional, robusto y validado. Esta experiencia ha servido, además, como un valioso aprendizaje sobre la importancia de una planificación rigurosa y la gestión de expectativas en un proyecto de software real.

## 10 MARCO SOCIOECONÓMICO

Este capítulo analiza el impacto del proyecto desde una perspectiva socioeconómica, considerando su presupuesto, su valor como recurso pedagógico y su contribución potencial a la sociedad y al sector educativo.

### 10.1 Presupuesto

La viabilidad económica de un proyecto de estas características se ha justificado mediante la estimación presupuestaria detallada en la [sección 4.3](#), que sitúa el coste de ejecución en 9.343,69 € y el precio de venta teórico en **11.679,61 €** antes de impuestos. Este presupuesto, que contempla los recursos humanos, la infraestructura y los márgenes empresariales, se ve optimizado por el uso de tecnologías de código abierto que eliminan los costes de licencia de software, demostrando que se puede crear una herramienta potente de manera económicamente eficiente.

### 10.2 Impacto socioeconómico

El principal impacto del proyecto es de carácter social y educativo, contribuyendo directamente al **Objetivo de Desarrollo Sostenible (ODS) 4: Educación de Calidad** [24]. La aplicación ataca un problema fundamental: la elevada curva de aprendizaje en programación. Al ser una herramienta gratuita, accesible desde cualquier navegador y sin necesidad de instalación, democratiza el acceso a recursos de aprendizaje de calidad. Permite a los estudiantes centrarse en el pensamiento computacional y la lógica algorítmica, con el objetivo de reducir la frustración que a menudo provoca la sintaxis y causa abandonos en asignaturas introductorias. La funcionalidad de colaboración fomenta, además, el aprendizaje entre pares y el desarrollo de habilidades de trabajo en equipo.

Desde el punto de vista económico y de sostenibilidad, la aplicación se posiciona como una alternativa sin coste a herramientas comerciales, lo que puede suponer un ahorro para instituciones educativas y estudiantes. A largo plazo, al facilitar la formación de profesionales con una base sólida en programación, el proyecto ayuda a construir una fuerza laboral más cualificada. La arquitectura P2P de la funcionalidad colaborativa y el uso de un servidor de señalización muy ligero garantizan que los costes de mantenimiento y escalado de la plataforma sean mínimos.

Como producto distribuido digitalmente, la aplicación web posee una huella medioambiental reducida, eliminando el impacto asociado a la producción y distribución de soportes físicos y alineándose así con las prácticas de desarrollo sostenible del sector tecnológico.



## 11 MARCO REGULADOR

Este capítulo aborda el marco normativo y los estándares técnicos que afectan al desarrollo y despliegue de la aplicación web. El objetivo es asegurar que el proyecto cumple con la legislación aplicable, sentando las bases para una posible comercialización y garantizando la calidad y seguridad de la herramienta.

### 11.1 Propiedad intelectual e industrial

La propiedad intelectual del proyecto se gestiona de forma diferenciada para sus distintos componentes. La presente memoria, como documento escrito, se distribuye bajo la **licencia Creative Commons Reconocimiento – No Comercial – Sin Obra Derivada (CC BY-NC-ND)**, tal como se indica en su portada. Esto permite su libre difusión y su uso con fines académicos, siempre que se reconozca la autoría original, no se utilice comercialmente y no se realicen obras derivadas. Por otro lado, el código fuente de la aplicación se publica bajo la licencia **Creative Commons Reconocimiento – No Comercial – Compartir Igual 4.0 Internacional (CC BY-NC-SA 4.0)**. Esta licencia se elige estratégicamente para permitir que la comunidad pueda estudiar y modificar el código, pero prohíbe su uso comercial por parte de terceros y obliga a que cualquier obra derivada se comparta bajo las mismas condiciones.

Complementariamente, para poder aceptar contribuciones de la comunidad, se requiere la firma de un **Acuerdo de Licencia de Colaborador (CLA – Contributor License Agreement)**, otorgando al autor los derechos necesarios para integrar y relicenciar dichas aportaciones, incluso con fines comerciales. En contraste con el código, el nombre “Flowming” y su logotipo no se rigen por esta licencia y quedan protegidos por la **Ley de Propiedad Intelectual**, que reserva todos los derechos de autor. Adicionalmente, podrían ser registrados como marca en un futuro para proteger la identidad del producto.

Es fundamental señalar también el marco legal de las tecnologías de terceros sobre las que se construye la aplicación. El proyecto se apoya en un conjunto de bibliotecas de código abierto. La mayoría de estas, incluyendo herramientas clave como React, Vite, React Flow y Yjs, se distribuyen bajo la **licencia MIT**. Esta es una licencia de software libre muy permisiva que permite su uso, modificación y distribución, incluso en proyectos comerciales, con la única condición principal de mantener el aviso de derechos de autor y la licencia original. La elección de bibliotecas con licencias permisivas como la MIT es una decisión estratégica que garantiza la solidez legal del proyecto y ofrece flexibilidad para su evolución futura, sin imponer restricciones sobre el licenciamiento del producto final.

### 11.2 Privacidad y protección de datos

En materia de privacidad y protección de datos, la aplicación se ha diseñado para minimizar la recopilación de información personal, en línea con los principios del

**Reglamento General de Protección de Datos (GDPR – General Data Protection Regulation)** de la Unión Europea. El modo de uso individual se ejecuta íntegramente en el navegador del cliente y no almacena datos en servidores externos. Sin embargo, la funcionalidad colaborativa, aunque basada en una arquitectura *Peer-To-Peer* (P2P), requiere un servidor de señalización para el intercambio temporal de metadatos de conexión, como las direcciones IP. Para cumplir con el GDPR, se informaría al usuario de este intercambio antes de iniciar una sesión, el servidor de señalización no almacenaría datos de forma persistente y la comunicación se realizaría a través de canales seguros.

### 11.3 Estándares técnicos

El desarrollo se adhiere a estándares técnicos para garantizar su calidad e interoperabilidad. La representación visual de los nodos del diagrama de flujo se basa en los símbolos de la norma ISO 5807 [37], asegurando que los diagramas sean universalmente comprensibles. La aplicación está construida sobre tecnologías web estándar del W3C, como HTML5 y CSS3. Además, se ha prestado atención a la accesibilidad, tomando como referencia las Pautas de Accesibilidad para el Contenido Web (WCAG) 2.2 [63]. El objetivo es que la herramienta sea usable por el mayor número de estudiantes posible, incluyendo aquellos con diversidad funcional.

## 12 CONCLUSIONES Y LÍNEAS FUTURAS

### 12.1 Conclusiones generales

El objetivo fundamental de este Trabajo de Fin de Grado era suavizar la pronunciada curva de aprendizaje de la programación, diseñando y desarrollando una herramienta web que permitiera a los estudiantes centrarse en la lógica algorítmica antes de enfrentarse a la complejidad sintáctica. Se puede concluir que este objetivo se ha alcanzado con éxito mediante la creación de “Flowming”, una aplicación web completamente funcional.

En cumplimiento de los objetivos específicos, el proyecto ha culminado en la entrega de una solución de software completa que integra un conjunto de funcionalidades clave. Se ha diseñado e implementado una interfaz web intuitiva para la creación visual de diagramas de flujo, un robusto motor de ejecución capaz de interpretar dichos diagramas, y funcionalidades interactivas esenciales como la entrada de datos, la visualización de salidas y el seguimiento paso a paso del flujo de ejecución. Además, se han incorporado capacidades de depuración visual y un generador de código fuente, que traduce los diagramas a lenguajes de programación como Python, cumpliendo así con el propósito de servir como puente hacia la programación textual.

Si bien se priorizó la robustez de estas funcionalidades principales sobre la implementación completa de algunos requisitos secundarios o no funcionales, como el diseño totalmente *responsive* o un control más granular de la ejecución, se han sentado las bases técnicas para los objetivos secundarios, como la colaboración en tiempo real mediante una arquitectura *Peer-to-Peer*. Asimismo, se ha implementado un módulo de ejercicios dotado de un sistema de evaluación automática que valida las soluciones del usuario contra casos de prueba predefinidos, además de ofrecer la opción de visualizar una solución de referencia. Esta funcionalidad consolida el valor pedagógico de la herramienta al ofrecer un entorno práctico y estructurado para la autoevaluación.

La aplicación desarrollada responde directamente a las limitaciones de las herramientas existentes identificadas en el estado del arte. Al ser una aplicación web, supera la barrera de accesibilidad de las soluciones de escritorio tradicionales como RAPTOR o Flowgorithm, y ofrece una alternativa a los lenguajes basados en bloques cuya transición a la programación textual es a menudo objeto de debate. El resultado es un recurso pedagógico que combina la claridad conceptual de los diagramas de flujo con la modernidad y accesibilidad de las tecnologías web actuales.

La validación del trabajo se ha realizado a través de dos vertientes: la técnica y la de usuario. Desde el punto de vista técnico, la superación de las pruebas automatizadas y el correcto funcionamiento del ciclo de CI/CD garantizan la robustez y fiabilidad del software implementado. En cuanto a la validación de la usabilidad y el potencial pedagógico de la herramienta, el riguroso proceso de evaluación con usuarios ha confirmado empíricamente el éxito del proyecto. La obtención de una puntuación media SUS de 88,61 sitúa a la aplicación en un nivel de usabilidad excelente, y el feedback

cualitativo ha ratificado que funcionalidades como el depurador visual son altamente valoradas por los usuarios y contribuyen directamente a la comprensión de los algoritmos.

No obstante, este mismo proceso de evaluación ha sido fundamental para identificar áreas de mejora clave. Los problemas de usabilidad detectados, como la confusión en la interacción inicial y la falta de predictibilidad en el comportamiento de las ramas condicionales, son hallazgos de gran valor. Lejos de invalidar el producto, estos resultados proporcionan una hoja de ruta clara para futuras iteraciones, orientada a refinar la interacción y alinearla de manera más precisa con el modelo mental del usuario. La capacidad de detectar y diagnosticar estas fricciones demuestra la madurez del ciclo de desarrollo y la importancia de la validación con usuarios en la creación de herramientas educativas eficaces.

En definitiva, este proyecto culmina con la entrega de un producto de software complejo, funcional y bien diseñado, y representa una contribución tangible al campo de las herramientas educativas para la programación. Se ha demostrado la viabilidad de crear una plataforma web moderna que mitiga las dificultades iniciales del aprendizaje del pensamiento computacional, validando así la hipótesis de partida y cumpliendo los objetivos planteados al inicio de este trabajo.

## 12.2 Conclusiones personales

La realización de este Trabajo de Fin de Grado ha sido una experiencia muy completa y desafiante en mi trayectoria académica. Ha sido un verdadero puente entre la formación teórica adquirida durante el grado y la práctica profesional, obligándome a integrar conocimientos de múltiples disciplinas para llevar a cabo un proyecto de software desde su concepción hasta su despliegue y validación.

Desde el punto de vista de la gestión y la ingeniería, el proyecto ha sido un ejercicio práctico de las metodologías aprendidas en asignaturas como Dirección de Proyectos del Desarrollo de Software e Ingeniería del Software. La definición de requisitos, el diseño de casos de uso, la creación de matrices de trazabilidad, la estimación del esfuerzo mediante Puntos de Caso de Uso (UCP) y la planificación temporal fueron contribuciones fundamentales de estas asignaturas. Asimismo, los principios de Desarrollo de Software y Desarrollo y Operación de Sistemas de Software (DevOps) guiaron la arquitectura del sistema, buscando siempre alta cohesión y bajo acoplamiento, y se materializaron en la implementación de pruebas automatizadas y un flujo de integración y despliegue continuo (CI/CD).

El núcleo técnico de la aplicación, compuesto por el motor de ejecución de diagramas y el generador de código, ha sido el reto más complejo y gratificante. Aquí fue crucial la aplicación de los conocimientos de Procesadores del Lenguaje (cursada como *Compilers* en la Universidad de Waterloo) y su fundamento en la Teoría de Autómatas y Lenguajes Formales. El diseño de un analizador sintáctico (*parser*) para transformar las expresiones en un Árbol de Sintaxis Abstracta (AST) y la posterior implementación del generador de código fueron aplicaciones directas de estos conceptos. Todo ello, por supuesto, se

construyó sobre las bases sólidas provenientes de las asignaturas de Programación y Estructura de Datos y Algoritmos.

En paralelo, el desarrollo de la interfaz de usuario me permitió aplicar los conocimientos de diseño y evaluación de la experiencia de usuario. Asignaturas como Sistemas Interactivos y Ubicuos (cursada como *Human-Computer Interaction* en Waterloo) y Accesibilidad y Diseño para Todos me proporcionaron el marco teórico para crear prototipos y diseñar una interfaz intuitiva, culminando en una evaluación formal mediante el cuestionario SUS y la consideración de las pautas WCAG 2.2. La implementación práctica fue posible gracias a las habilidades adquiridas en Interfaces de Usuario (cursada como *User Interfaces* en Waterloo), donde pude trabajar con tecnologías como TypeScript y React, que han sido el corazón del frontend de este proyecto.

Más allá de la aplicación de conocimientos específicos, este TFG me ha enseñado sobre la importancia de la gestión del tiempo, la resolución de problemas complejos bajo incertidumbre y la perseverancia. Ver cómo conceptos abstractos de diferentes asignaturas convergen para crear un producto tangible y funcional ha sido una experiencia inmensamente satisfactoria. Este trabajo no solo representa la culminación de mi grado, sino que también consolida mi confianza y mis habilidades para afrontar los desafíos del mundo profesional de la ingeniería de software.

### 12.3 Líneas futuras

Aunque el proyecto ha cumplido con sus objetivos fundamentales, su desarrollo ha sentado las bases para numerosas mejoras y ampliaciones futuras. Estas líneas de trabajo se pueden agrupar en varias categorías principales:

#### Mejoras de usabilidad y experiencia de usuario (corto plazo)

La prioridad inmediata sería abordar directamente los hallazgos de la evaluación con usuarios para refinar la experiencia y alinearla completamente con el modelo mental del usuario. Esto incluye:

- **Rediseñar la experiencia inicial:** Sustituir la conexión inicial por defecto por una guía más clara e interactiva sobre cómo crear y conectar los primeros bloques. Esto puede incluir un tutorial interactivo que guía al usuario sobre el uso de las funcionalidades principales de la aplicación.
- **Corregir el comportamiento de las ramas condicionales:** Implementar una lógica de asignación “Sí/No” que sea predecible y controlable por el usuario, eliminando el patrón de alternancia global.
- **Implementar un diseño totalmente responsive:** Adaptar la interfaz para garantizar su plena funcionalidad y una experiencia de usuario óptima en dispositivos con pantallas más pequeñas, como móviles. Si bien se considera

que la plataforma principal de uso será un ordenador, dado que es el entorno habitual para tareas de programación y el equipo que se suele requerir en clases relacionadas, ofrecer un soporte completo para dispositivos táctiles ampliaría enormemente la accesibilidad y los casos de uso de la herramienta, como la consulta o revisión rápida de diagramas.

- **Implementar un control de ejecución más granular:** Añadir los controles de ejecución para avanzar o retroceder paso a paso que fueron concebidos en los *mockups* iniciales. Su implementación requiere una refactorización del motor de ejecución para gestionar correctamente el estado del siguiente nodo a ser ejecutado y no se priorizó al no considerarse esencial, pero enriquecería la capacidad de análisis del estudiante.

### **Ampliación funcional y pedagógica**

Una vez consolidada la experiencia de usuario, el siguiente paso sería enriquecer la funcionalidad de la herramienta para cubrir escenarios de aprendizaje más avanzados:

- **Encapsulación de lógica en funciones:** Permitir al usuario agrupar un conjunto de bloques en un subdiagrama considerado como función que pueda ser llamado desde múltiples puntos del diagrama principal. Esto introduciría conceptos clave como la abstracción, la reutilización de código y la recursividad.
- **Ampliación de los lenguajes de destino:** Añadir la capacidad de generar código en otros lenguajes de programación relevantes para la enseñanza, como Java.
- **Implementación de un sistema de tipado débil:** Ofrecer al usuario la posibilidad de cambiar a un modo de tipado débil (*weakly typed*), donde el sistema permita operaciones entre tipos de datos diferentes mediante conversiones implícitas. Esto contrastaría con el sistema actual de tipado fuerte (*strongly typed*) y alinearía la herramienta con la filosofía de lenguajes como JavaScript, ofreciendo una mayor flexibilidad para diferentes enfoques pedagógicos.
- **Integración con plataformas de gestión del aprendizaje (LMS):** Desarrollar un módulo de integración que permita incrustar la aplicación en plataformas como Moodle. Esto facilitaría a los docentes la creación de tareas, la evaluación automática de los diagramas de los estudiantes y el seguimiento de su progreso, convirtiendo la herramienta en un recurso educativo plenamente integrado en el aula virtual.

## Mejoras de personalización y accesibilidad

- **Personalización de la interfaz:** Permitir al usuario modificar la apariencia de la aplicación, incluyendo un modo oscuro (*dark mode*), cambiar los colores del lienzo o del texto, y reorganizar la disposición de los paneles para crear un espacio de trabajo más personal.
- **Profundizar en la accesibilidad:** Realizar una auditoría completa de la aplicación contra las pautas WCAG 2.2 e implementar los cambios necesarios para garantizar que la herramienta sea usable por el mayor número posible de estudiantes, incluyendo aquellos con diversidad funcional.

## Despliegue y arquitectura

- **Puesta en producción de la colaboración en tiempo real:** La funcionalidad de colaboración, aunque técnicamente implementada, no está operativa debido a restricciones de la red de la universidad que bloquean las conexiones WebSocket del servidor de señalización. La línea de trabajo futuro consistiría en coordinar con el departamento de IT de la universidad o, como alternativa más pragmática, migrar el servidor de señalización a una plataforma de *hosting* externa que no imponga estas restricciones.

Estas líneas futuras transformarían la herramienta de un producto funcional y validado a una plataforma educativa aún más completa, robusta y adaptada a las diversas necesidades de los estudiantes de programación.



## BIBLIOGRAFÍA

- [1] Andrew Begel. 1996. LogoBlocks: A Graphical Programming Language for Interacting with the World. Tesis de pregrado avanzada, MIT Media Laboratory. Massachusetts Institute of Technology, Cambridge, MA, Estados Unidos. Retrieved from <https://andrewbegel.com/mit/begel-aup.pdf>
- [2] Karen Brennan and Mitchel Resnick. 2012. New frameworks for studying and assessing the development of computational thinking. (2012).
- [3] John Brooke. SUS - A quick and dirty usability scale.
- [4] Martin C. Carlisle, Terry A. Wilson, Jeffrey W. Humphries, and Steven M. Hadfield. 2005. RAPTOR: a visual programming environment for teaching algorithmic problem solving. In *Proceedings of the 36th SIGCSE technical symposium on Computer science education*, February 23, 2005. ACM, St. Louis Missouri USA, 176–180. <https://doi.org/10.1145/1047344.1047411>
- [5] Vladimir contributors Anthony Fu, Ari Perkkiö, Hiroshi Ogawa, Patak, Joaquín Sánchez and Vitest. 2024. Vitest. Retrieved June 12, 2025 from <https://vitest.dev/>
- [6] Stephen Cooper, Wanda Dann, and Randy Pausch. 2000. Alice: a 3-D tool for introductory programming concepts. *J Comput Sci Coll* 15, 5 (April 2000), 107–116.
- [7] Data Processing Division. 1970. *Flowcharting Techniques*. International Business Machines Corporation, Estados Unidos.
- [8] T. O. Ellis, J. F. Heafner, and W. L. Sibley. 1969. *The GRAIL Project: An Experiment in Man-Machine Communications*. Retrieved May 14, 2025 from [https://www.rand.org/pubs/research\\_memoranda/RM5999.html](https://www.rand.org/pubs/research_memoranda/RM5999.html)
- [9] John C. Giordano and Martin Carlisle. 2006. Toward a more effective visualization tool to teach novice programmers. In *Proceedings of the 7th conference on Information technology education (SIGITE '06)*, October 19, 2006. Association for Computing Machinery, New York, NY, USA, 115–122. <https://doi.org/10.1145/1168812.1168841>
- [10] Herman Heine Goldstine and John von Neumann. 1947. *Planning and coding of problems for an electronic computing instrument*. Institute for Advanced Study, Estados Unidos.
- [11] Anabela Gomes and António José Mendes. 2007. An environment to improve programming education. In *Proceedings of the 2007 international conference on Computer systems and technologies - CompSysTech '07*, 2007. ACM Press, Bulgaria, 1. <https://doi.org/10.1145/1330598.1330691>
- [12] Ewa Graczyńska. 2010. ALICE as a tool for programming at schools. *Nat. Sci.* 2, 2 (March 2010), 124–129. <https://doi.org/10.4236/ns.2010.22021>
- [13] Dee Gudmundsen, Lisa Olivieri, and Namita Sarawagi. 2011. Using visual logic©: three different approaches in different courses - general education, CS0, and CS1. *J Comput Sci Coll* 26, 6 (June 2011), 23–29.

- [14] Erica Haugvedt and Duane Lewis Abata. 2021. Ada Lovelace: First Computer Programmer and Hacker? July 26, 2021. . Retrieved May 22, 2025 from <https://peer.asee.org/ada-lovelace-first-computer-programmer-and-hacker>
- [15] David Heise and Tomas Joyner. Prograph: A Visual Programming Language.
- [16] Tony Jenkins. 2002. On the Difficulty of Learning to Program. In *Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences*, 2002. Loughborough, UK, 53–58.
- [17] Jeffrey Kodosky. 2020. LabVIEW. *Proc. ACM Program. Lang.* 4, HOPL (June 2020), 1–54. <https://doi.org/10.1145/3386328>
- [18] Mathieu K Kourouma. Capabilities and Features of Raptor, Visual Logic, and Flowgorithm for Program Logic and Design.
- [19] James R. Lewis. 2018. The System Usability Scale: Past, Present, and Future. *Int. J. Human–Computer Interact.* 34, 7 (July 2018), 577–590. <https://doi.org/10.1080/10447318.2018.1455307>
- [20] John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. 2010. The Scratch Programming Language and Environment. *ACM Trans. Comput. Educ.* 10, 4 (November 2010), 1–15. <https://doi.org/10.1145/1868358.1868363>
- [21] S. Matwin and T. Pietrzykowski. 1985. PROGRAPH: A preliminary report. *Comput. Lang.* 10, 2 (January 1985), 91–126. [https://doi.org/10.1016/0096-0551\(85\)90002-5](https://doi.org/10.1016/0096-0551(85)90002-5)
- [22] Luigi Federico Menabrea. 1843. Sketch of the Analytical Engine Invented by Charles Babbage (translated with notes by A. A. Lovelace). *Sci. Mem.* 3, (1843), 666–731.
- [23] Ministerio de Educación y Formación Profesional. 2022. *Real Decreto 217/2022, de 29 de marzo, por el que se establece la ordenación y las enseñanzas mínimas de la Educación Secundaria Obligatoria*. Retrieved May 13, 2025 from <https://www.boe.es/eli/es/rd/2022/03/29/217>
- [24] Mirtha Moran. Educación. *Desarrollo Sostenible*. Retrieved June 12, 2025 from <https://www.un.org/sustainabledevelopment/es/education/>
- [25] Brad A. Myers. 1990. Taxonomies of visual programming and program visualization. *J. Vis. Lang. Comput.* 1, 1 (March 1990), 97–123. [https://doi.org/10.1016/S1045-926X\(05\)80036-9](https://doi.org/10.1016/S1045-926X(05)80036-9)
- [26] Seymour Papert. 1980. *Mindstorms: children, computers, and powerful ideas*. Basic Books, New York.
- [27] V. D. Parondzhanov. 1995. Visual syntax of the DRAGON language. *Program. Comput. Softw.* 21, 3 (May 1995), 142–153.
- [28] Erik Pasternak, Rachel Fenichel, and Andrew N. Marshall. 2017. Tips for creating a block language with blockly. In *2017 IEEE Blocks and Beyond Workshop (B&B)*, October 2017. 21–24. <https://doi.org/10.1109/BLOCKS.2017.8120404>
- [29] Jeff Sauro PhD. 5 Ways to Interpret a SUS Score – MeasuringU. Retrieved June 15, 2025 from <https://measuringu.com/interpret-sus-score/>

- [30] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, and Yasmin Kafai. 2009. Scratch: programming for all. *Commun. ACM* 52, 11 (November 2009), 60–67. <https://doi.org/10.1145/1592761.1592779>
- [31] Rizki Hardian Sakti, Nizwardi Jalinus, Sukardi Sukardi, Hendra Hidayat, Rizky Ema Wulansari, Chau Trung Tin, and Firas Tayseer Mohammad Ayasrah. 2024. Diving into the Future: Unravelling the Impact of Flowgorithm and Discord Fusion on Algorithm and Programming Courses and Fostering Computational Thinking. *Int. J. Learn. Teach. Educ. Res.* 23, 7 (August 2024). Retrieved April 29, 2025 from <https://ijlter.org/index.php/ijlter/article/view/10726>
- [32] shadcn. Introduction. Retrieved June 12, 2025 from <https://ui.shadcn.com/docs>
- [33] David Canfield Smith. 1975. PYGMALION: A Creative Programming Environment. Tesis doctoral, Computer Science Department. Universidad de Stanford, Stanford, CA, Estados Unidos. Retrieved from <https://apps.dtic.mil/sti/pdfs/ADA016811.pdf>
- [34] Ivan E. Sutherland. 1963. Sketchpad: a man-machine graphical communication system. In *Proceedings of the May 21-23, 1963, spring joint computer conference (AFIPS '63 (Spring))*, May 21, 1963. Association for Computing Machinery, New York, NY, USA, 329–346. <https://doi.org/10.1145/1461551.1461591>
- [35] David Weintrop. 2019. Block-based programming in computer science education. *Commun ACM* 62, 8 (July 2019), 22–25. <https://doi.org/10.1145/3341221>
- [36] 1970. Flowchart symbols and their usage in information processing.
- [37] 1985. Information processing — Documentation symbols and conventions for data, program and system flowcharts, program network charts and system resources charts.
- [38] 2016. Scratch Has a Marketing Problem. *freeCodeCamp.org*. Retrieved May 25, 2025 from <https://www.freecodecamp.org/news/scratch-has-a-marketing-problem-f84626bd18ef/>
- [39] 2022. Visual Logic. *Wikipedia*. Retrieved May 25, 2025 from [https://en.wikipedia.org/w/index.php?title=Visual\\_Logic&oldid=1091091113](https://en.wikipedia.org/w/index.php?title=Visual_Logic&oldid=1091091113)
- [40] 2024. Introduction | Yjs Docs. Retrieved June 12, 2025 from <https://docs.yjs.dev>
- [41] 2024. Alice (software). *Wikipedia*. Retrieved May 25, 2025 from [https://en.wikipedia.org/w/index.php?title=Alice\\_\(software\)&oldid=1260949131](https://en.wikipedia.org/w/index.php?title=Alice_(software)&oldid=1260949131)
- [42] 2025. DRAKON. *Wikipedia*. Retrieved May 25, 2025 from <https://en.wikipedia.org/w/index.php?title=DRAKON&oldid=1268615315>
- [43] 2025. Blockly. *Wikipedia*. Retrieved May 25, 2025 from <https://en.wikipedia.org/w/index.php?title=Blockly&oldid=1287042294>
- [44] 2025. Esto es lo que paga una empresa por cada trabajador | Blog Anfix. Retrieved June 11, 2025 from <https://www.anfix.com/blog/cuanto-paga-una-empresa-por-un-trabajador>
- [45] 2025. LabVIEW. *Wikipedia*. Retrieved May 25, 2025 from <https://en.wikipedia.org/w/index.php?title=LabVIEW&oldid=1291850200>

- [46] 2025. Node-Based UIs in React. *React Flow*. Retrieved June 12, 2025 from <https://reactflow.dev>
- [47] RAPTOR - Flowchart Interpreter. Retrieved April 29, 2025 from <https://raptor.martincarlisle.com/>
- [48] Flowgorithm - Flowchart Programming Language. Retrieved April 29, 2025 from <http://www.flowgorithm.org/>
- [49] Figure 3: Dataflow programming in Graph. Here the programmer is... *ResearchGate*. Retrieved May 25, 2025 from [https://www.researchgate.net/figure/Dataflow-programming-in-Graph-Here-the-programmer-is-using-the-low-level-primitive\\_fig2\\_220391588](https://www.researchgate.net/figure/Dataflow-programming-in-Graph-Here-the-programmer-is-using-the-low-level-primitive_fig2_220391588)
- [50] ¿Qué es NI LabVIEW? Programación gráfica para pruebas y medidas. Retrieved May 15, 2025 from <https://www.ni.com/es/shop/labview.html>
- [51] Alice – Tell Stories. Build Games. Learn to Program. Retrieved May 15, 2025 from <https://www.alice.org/>
- [52] Добро пожаловать на сайт «Визуальный язык ДРАКОН» [Визуальный язык ДРАКОН]. Retrieved May 15, 2025 from <https://drakon.su/start>
- [53] DRAKON Editor. Retrieved May 14, 2025 from <https://drakon-editor.sourceforge.net/editor.html>
- [54] Scratch - Imagine, Program, Share. Retrieved May 15, 2025 from <https://scratch.mit.edu/>
- [55] Blockly. *Google for Developers*. Retrieved April 29, 2025 from <https://developers.google.com/blockly?hl=es-419>
- [56] ¿Cuánto cobra un/a Jefe de proyecto en Madrid? | InfoJobs Salarios. Retrieved June 2, 2025 from <https://salarios.infojobs.net/jefe-de-proyecto/madrid>
- [57] ¿Cuánto cobra un/a Ingeniero de software en Madrid? | InfoJobs Salarios. Retrieved June 2, 2025 from <https://salarios.infojobs.net/ingeniero-de-software/madrid>
- [58] ¿Cuánto cobra un/a Diseñador UI en Madrid? | InfoJobs Salarios. Retrieved June 2, 2025 from <https://salarios.infojobs.net/dise%C3%B1ador-ui/madrid>
- [59] ¿Cuánto cobra un/a Diseñador UX en Madrid? | InfoJobs Salarios. Retrieved June 2, 2025 from <https://salarios.infojobs.net/dise%C3%B1ador-ux/madrid>
- [60] React. Retrieved June 12, 2025 from [https://react.dev/](https://react.dev)
- [61] Vite. *vitejs*. Retrieved June 12, 2025 from <https://vite.dev>
- [62] Lucide Icons. *Lucide*. Retrieved June 12, 2025 from <https://lucide.dev>
- [63] Web Content Accessibility Guidelines (WCAG) 2.2. Retrieved June 12, 2025 from <https://www.w3.org/TR/WCAG22/>

## A. ANEXO: DOCUMENTO DE REQUISITOS DE USUARIO

**uc3m | Universidad Carlos III de Madrid**

### **Flowming**

“Aplicación web para el diseño y  
ejecución de diagramas de flujo”

*Documento de Requisitos de Usuario*

---

Fecha: **29/04/25** - Versión: **2.0**

Autor: **Daniel Pérez Fernández**

Tutor: **Álvaro Montero Montes**

## TABLA DE CONTENIDOS

<b>1 INTRODUCCIÓN</b>	<b>3</b>
1.1 Objetivo de este documento	3
1.2 Visión del proyecto	3
1.3 Público objetivo	3
<b>2 PERFILES DE USUARIO Y OBJETIVOS</b>	<b>4</b>
2.1 Descripciones de los usuarios objetivo	4
2.2 Metas de los usuarios	4
<b>3 REQUISITOS</b>	<b>6</b>
3.1 Clasificación de requisitos	6
3.2 Plantilla para los requisitos	6
3.3 Requisitos de capacidad (UR)	7
3.4 Requisitos de restricción (UCR)	9
<b>4 ALCANCE Y PRIORIZACIÓN</b>	<b>11</b>
4.1 Escala de priorización (MoSCoW)	11
4.2 Dentro del alcance	11
4.3 Fuera del alcance	12
<b>5 GLOSARIO</b>	<b>13</b>
<b>6 APROBACIÓN</b>	<b>14</b>

## 1 INTRODUCCIÓN

Este capítulo presenta una descripción general del documento.

### 1.1 Objetivo de este documento

La intención de este documento es definir una lista completa de los requisitos de usuario para la aplicación web Flowming. Su resultado servirá de fundación para los consecutivos requisitos de software, ya que traduce las necesidades del usuario en especificaciones técnicas que el equipo de desarrollo podrá implementar.

### 1.2 Visión del proyecto

Crear una herramienta fácil de usar y accesible desde la web que permita a estudiantes tener un primer contacto con la programación gracias a la posibilidad de diseñar y ejecutar algoritmos de manera visual usando diagramas de flujo, ayudándolos así a entender la lógica fundamental de la programación sin la complicación de sintaxis compleja y específica.

### 1.3 Público objetivo

Este documentado está destinado principalmente para:

- **El cliente (tutor):** Para evaluar la comprensión de las necesidades y el alcance del proyecto.
- **El equipo de desarrollo:** Como guía fundamental para el diseño y desarrollo de la aplicación web.

## 2 PERFILES DE USUARIO Y OBJETIVOS

Este capítulo describe quiénes usarán la aplicación y qué esperan conseguir con ella.

### 2.1 Descripciones de los usuarios objetivo

Los usuarios principales de la aplicación serán:

- **Estudiantes secundarios y universitarios:** Especialmente aquellos que estén cursando asignaturas introductorias a la programación o tengan interés en iniciarse en ella.
- **Educadores:** Profesores que buscan una herramienta visual para enseñar la lógica fundamental de programación de forma intuitiva.
- **Autodidactas:** Personas interesadas en aprender los fundamentos de programación por su cuenta.

Se espera que los usuarios tengan por lo menos las siguientes características que les permitirán el uso de la aplicación:

- **Habilidades técnicas:** Se espera que tengan habilidad básica de manejo de ordenadores y navegación web. No se asume conocimiento previo en programación.
- **Entorno:** Usarán la aplicación en ordenadores o móviles con conexión a internet.

### 2.2 Metas de los usuarios

Los usuarios utilizarán la aplicación para lograr los siguientes objetivos:

- **Comprender conceptos fundamentales:** Entender visualmente la secuencia de pasos, las decisiones (condicionales) y la repetición (bucles) en la lógica de un programa.
- **Practicar sin código:** Enfocarse en la lógica para resolver ejercicios simples sin preocuparse de la sintaxis de un lenguaje de programación específico.

- **Visualizar la ejecución:** Ver cómo un conjunto de instrucciones lógicas se ejecuta paso a paso y cómo fluye el control.
- **Obtener retroalimentación inmediata:** Poder ejecutar sus diagramas de flujo y ver si la lógica produce el resultado esperado.

### 3 REQUISITOS

Esta sección detalla las necesidades y expectativas de los usuarios respecto a la aplicación.

#### 3.1 Clasificación de requisitos

Los requisitos de usuario descritos en este documento se dividen en dos categorías principales:

- **Requisitos de capacidad (UR - User Requirements)**: Describen las funcionalidades que el usuario necesita poder realizar con el sistema para alcanzar sus metas.
- **Requisitos de restricción (UCR - User Constraint Requirements)**: Describen las cualidades o limitaciones que el sistema debe cumplir desde la perspectiva del usuario.

#### 3.2 Plantilla para los requisitos

Cada requisito en las subsecciones siguientes se presentará usando la siguiente plantilla:

- **ID**: Identificador único para trazabilidad.
- **Requisito**: Descripción clara y concisa del requisito.
- **Módulo**: Agrupación lógica de requisitos relacionados.
- **Prioridad**: Nivel de importancia del requisito según la escala MoSCoW (Must, Should, Could, Won't), definida en la sección 4.
- **Estado**: Estado actual del requisito (Aceptado, Eliminado, Revisado, Propuesto).

### 3.3 Requisitos de capacidad (UR)

ID	REQUISITO	MÓDULO	PRIORIDAD	ESTADO
UR-01	El usuario dispondrá de un lienzo sobre el que podrá crear diagramas de flujo	Diseño	M	Aceptado
UR-02	El usuario podrá añadir los siguientes símbolos al diagrama de flujo: Inicio/Fin, Decisión, Proceso, Entrada/Salida	Diseño	M	Aceptado
UR-03	El usuario podrá conectar los símbolos del diagrama de flujo entre ellos mediante conectores	Diseño	M	Aceptado
UR-04	El usuario especificará manualmente el contenido (expresión y/o variable) para los símbolos Decisión, Proceso y Entrada/Salida	Diseño	M	Aceptado
UR-05	El usuario podrá eliminar símbolos y conectores del diagrama de flujo	Diseño	M	Aceptado
UR-06	El usuario podrá mover los símbolos en el lienzo	Diseño	M	Aceptado
UR-07	El usuario podrá exportar y/o importar la configuración de un diagrama de flujo	Diseño	M	Aceptado
UR-08	El usuario podrá iniciar, pausar y/o finalizar la ejecución del diagrama de flujo actual	Ejecución	M	Aceptado
UR-09	El usuario proporcionará datos de entrada cuando el flujo de ejecución alcance un símbolo de Entrada	Ejecución	M	Aceptado

UR-10	El usuario visualizará los datos de salida generados por el símbolo de Salida cuando el flujo de ejecución lo alcance	Ejecución	M	Aceptado
UR-11	El usuario observará el valor actual de cada variable definida durante la ejecución del diagrama de flujo	Ejecución	S	Aceptado
UR-12	El usuario visualizará el flujo de ejecución a través de indicaciones visuales en los símbolos y conectores del diagrama de flujo	Ejecución	M	Aceptado
UR-13	El usuario podrá visualizar el historial de cambios de valor de las variables tras la ejecución del diagrama de flujo	Depuración	S	Aceptado
UR-14	El usuario podrá visualizar la cadena de ejecución tras la ejecución del diagrama de flujo	Depuración	S	Aceptado
UR-15	El usuario podrá traducir el diagrama de flujo a código en el lenguaje de programación seleccionado	Otros	M	Aceptado
UR-16	El usuario podrá acceder a un banco de ejercicios en los que encontrará descripciones de ejercicios proporcionados	Ejercicios	C	Aceptado
UR-17	El usuario podrá comparar el diagrama de flujo actual con la solución al ejercicio seleccionado	Ejercicios	C	Aceptado
UR-18	El usuario podrá colaborar en linea con otros usuarios en el mismo diagrama de flujo	Colaboración	C	Aceptado
UR-19	El usuario podrá acceder a una videoconferencia con otros usuarios,	Colaboración	C	Aceptado

	en la que podrán compartir audio y video			
--	--	--	--	--

### 3.4 Requisitos de restricción (UCR)

ID	REQUISITO	MÓDULO	PRIORIDAD	ESTADO
UCR-01	La aplicación presentará una interfaz intuitiva para las operaciones de diseño [UR-01..06]	Interfaz	M	Aceptado
UCR-02	La aplicación mostrará los símbolos del diagrama de flujo [UR-02] siguiendo la representación visual estándar y claramente distinguible	Interfaz	M	Aceptado
UCR-03	La aplicación mostrará indicaciones visuales claras del flujo de ejecución durante la ejecución del diagrama de flujo [UR-12]	Interfaz	M	Aceptado
UCR-04	La aplicación presentará los datos de entrada y salida [UR-9, UR-10] de forma clara y comprensible para el usuario	Interfaz	M	Aceptado
UCR-05	Todos los mensajes de error o advertencia mostrados por la aplicación deberán estar redactados en un lenguaje claro y comprensible para el usuario, incluyendo una sugerencia para solucionar el problema	Interfaz	M	Aceptado
UCR-06	La aplicación proporcionará adaptabilidad de la interfaz y funcionalidad en dispositivos	Interfaz	S	Aceptado

	móviles			
UCR-07	La aplicación ejecutará la lógica del diagrama de flujo [UR-8] de acuerdo con la semántica estándar de los diagramas de flujo	Fiabilidad	M	Aceptado
UCR-08	La aplicación generará código [UR-19] sintácticamente correcto en el lenguaje de programación seleccionado y que represente la lógica del diagrama de flujo fiablemente	Fiabilidad	M	Aceptado
UCR-09	La aplicación permitirá la traducción a código [UR-15] por lo menos para los siguientes lenguajes de programación: Python, Javascript	Alcance	M	Aceptado
UCR-10	La aplicación soportará los siguientes tipo de datos [UR-04] para las variables y expresiones: texto, número, punto flotante, lógico, lista	Alcance	M	Aceptado
UCR-11	La aplicación operará en las versiones estables más recientes de los navegadores web	Alcance	M	Aceptado

## 4 ALCANCE Y PRIORIZACIÓN

Este capítulo define la escala de prioridad usada en los requisitos y el alcance del proyecto.

### 4.1 Escala de priorización (MoSCoW)

Se utiliza la siguiente escala para indicar la importancia relativa de cada requisito.

- **M (Must Have - Debe Tener):** Requisito esencial. Sin él, el sistema no cumple con su propósito fundamental y es esencialmente inútil para el usuario objetivo.
- **S (Should Have - Debería Tener):** Requisito importante que añade valor significativo, pero el sistema podría funcionar sin él (de forma limitada). Se intentará añadir si es posible.
- **C (Could Have - Podría Tener):** Requisito deseable pero no necesario. Se incluirá si el tiempo y recursos lo permiten después de completar los M y S.
- **W (Won't Have - No Tendrá):** Requisito explicitamente excluido de esta versión del proyecto.

### 4.2 Dentro del alcance

La aplicación permitirá a los usuarios realizar las siguientes acciones principales:

- Diseño visual de diagramas de flujo
  - Añadir símbolos estándar.
  - Conectar los símbolos para definir el flujo lógico.
  - Especificar el contenido dentro de los símbolos apropiados.
  - Modificar el diagrama de flujo (mover o eliminar símbolos y/o conectores).
- Ejecución visual
  - Iniciar, pausar y finalizar la ejecución del diagrama.
  - Visualizar el progreso de la ejecución.
  - Interactuar con la ejecución proporcionando entradas y visualizando las salidas correspondientes.
- Traducción a código fuente equivalente al diagrama en uno de los lenguajes de programación seleccionados.

- Colaborar en línea con otros usuarios.
- Acceder a un banco de ejercicios y comparar el diagrama de solución actual con la solución propuesta.

El enfoque principal se basa en proporcionar una herramienta funcional para el diseño y ejecución de diagramas de flujo de una forma altamente visual e intuitiva, orientada al aprendizaje de la lógica fundamental de programación. Además, será posible traducir el diagrama de flujo a código fuente equivalente de un lenguaje de programación dado. Adicionalmente, esta herramienta permitirá la colaboración en línea y el acceso a un banco de ejercicios según la viabilidad y el tiempo disponible.

#### 4.3 Fuera del alcance

Las siguientes funcionalidades no forman parte del alcance de esta versión del proyecto:

- Soporte para símbolos de diagrama de flujo no especificados.
- Generación de código para lenguajes de programación no especificados.
- Soporte para estructuras de datos complejas.
- Soporte para programación orientada a objetos.
- Soporte para concurrencia explícita en el programa.

## 5 GLOSARIO

Este capítulo define los términos clave usados a lo largo del documento para asegurar una compresión común.

- **Banco de ejercicios:** Una colección de enunciados de ejercicios de programación o lógica proporcionados dentro de la aplicación para que los usuarios practiquen.
- **Cadena de ejecución:** La secuencia ordenada de símbolos del diagrama de flujo que fueron ejecutados durante el flujo de ejecución.
- **Código fuente:** Texto escrito en un lenguaje de programación específico.
- **Colaboración en línea:** Funcionalidad que permite a múltiples usuarios editar el mismo diagrama de flujo simultáneamente a través de la red.
- **Conector:** Línea dirigida que une dos símbolos en el diagrama de flujo, representando la transferencia del control de ejecución.
- **Comando:** Instrucción textual introducida por el usuario durante la depuración para interactuar con el estado de ejecución.
- **Contenido (de símbolo):** La información (expresión, variable seleccionada) asociada a un símbolo específico que define su comportamiento.
- **Ejecución:** El proceso de simular el comportamiento definido por el diagrama de flujo, siguiendo los conectores y ejecutando las acciones de los símbolos.
- **Expresión:** Combinación de variables, valores constantes y operadores (aritméticos, lógicos, de comparación) que se evalúa para producir un resultado.
- **Flujo de ejecución:** La ruta o secuencia seguida a través de los símbolos y conectores del diagrama de flujo durante la ejecución.
- **Historial de cambios de valor:** Registro de los valores que una variable específica ha tomado a lo largo de la ejecución.

## 6 APROBACIÓN

Las firmas a continuación indican la revisión y acuerdo sobre el contenido de este Documento de Requisitos de Usuario, versión 2.0

Fecha: 29/04/2025

### - AUTOR

Nombre: Daniel Pérez Fernández

Firma: 

### - TUTOR

Nombre: Álvaro Montero Montes

Firma: 

Doy el visto bueno a los requisitos.

## B. ANEXO: MATRIZ DE TRAZABILIDAD DE REQUISITOS DE USUARIO A REQUISITOS DE SOFTWARE

**UR-07**

**UR-08**

**UR-09**

**UR-10**

**UR-11**

**UR-12**

**UR-13**

**UR-14**

**X X**

**X**

**X**

**X**

**X**

**X**

**X**

**X**

UR-15

UR-16

UR-17

UR-18

UR-19

UCR-01 UCR-02 UCR-03

X

X

X

X

X

X X X X

X

X

**UCR-04**

**UCR-05**

**UCR-06**

**UCR-07**

**UCR-08**

**UCR-09**

**UCR-10**

**UCR-11**

**X**

**X**

**X**

**X**

**X**

**X**

**X**

**X X**

## C. ANEXO: MATRIZ DE TRAZABILIDAD DE REQUISITOS DE SOFTWARE A CASOS DE USO

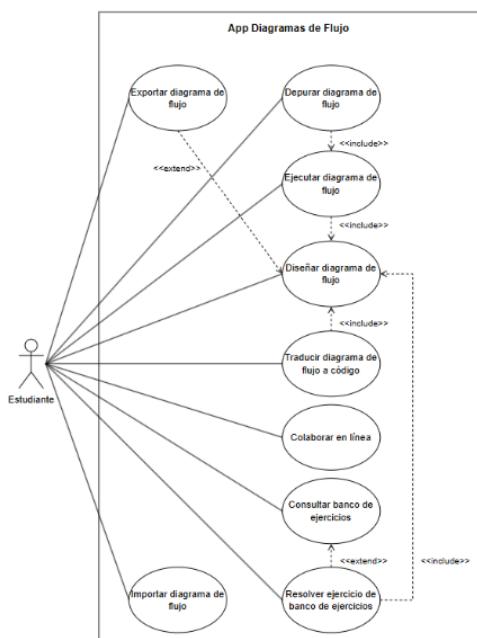
	CU01	CU02	CU03	CU04	CU05	CU06
NFR-14						
NFR-13						
NFR-12	X	X				
NFR-11						
NFR-10						
NFR-09						
NFR-08	X	X				
NFR-07						
NFR-06						
NFR-05	X					
NFR-04						
NFR-03	X					
NFR-02	X	X				
NFR-01	X	X	X			
FR-20						
FR-19						
FR-18						
FR-17						
FR-16						
FR-15						
FR-14						
FR-13						
FR-12						
FR-11						
FR-10						
FR-09						
FR-08						
FR-07	X					
FR-06	X	X				
FR-05	X	X				
FR-04	X	X				
FR-03	X	X				
FR-02	X	X				
FR-01	X					



## D. ANEXO: DESGLOSE COMPLETO DE LA ESTIMACIÓN UCP

### D.1. VALORACIÓN DE LOS CASOS DE USO

Código	Caso de uso	Número de transacciones	Complejidad algorítmica [1-3]	Complejidad del caso de uso	Simple	Medio	Complejo
1	Exportar diagrama de flujo	2	1 Simple	1 Simple	4	3	2
2	Importar diagrama de flujo	2	1 Simple	1 Simple			
3	Diseñar diagrama de flujo	5	1 Medio	2 Complejo			
4	Ejecutar diagrama de flujo	9	2 Complejo	2 Simple			
5	Depurar diagrama de flujo	2	2 Simple	3 Complejo			
6	Traducir diagrama de flujo a código	8	2 Medio	2 Medio			
7	Colaborar en línea	5	2 Complejo	2 Simple			
8	Consultar banco de ejercicios	1	2 Medio	2 Simple			
9	Resolver ejercicio de banco de ejercicios	5	1 Medio	1 Medio			



## D.2. CÁLCULOS DETALLADOS

Use Case Point Estimation Worksheet

Project Name: Flowming								
Project Manager Name: Daniel Pérez Fernández								
<b>Unadjusted Actor Weighting Table</b>								
Simple	Description: External System with well-defined API	Weight: 1	Number: 0	Value: 0				
Average	Description: External System using a protocol-based interface, e.g., HTTP, TCP/IP, or a	Weight: 2	Number: 0	Value: 0				
Complex	Description: Human	Weight: 3	Number: 1	Value: 3				
<b>Unadjusted Actor Weight Total (UAW)</b>			3					
<b>Unadjusted Use Case Weighting Table (Based on the number of)</b>								
Simple	Description: 1 – 3 transactions	Weight: 5	Number: 4	Value: 20				
Average	Description: 4 – 7 transactions	Weight: 10	Number: 3	Value: 30				
Complex	Description: > 7 transactions	Weight: 15	Number: 2	Value: 30				
<b>Unadjusted Use Case Weight Total (UUCW)</b>			80					
<b>Unadjusted Use Case Points (UUCP) = UAW + UUCW</b>								
83								
<b>Technical Complexity Factors</b>								
Distributed system	Scale: 0=no important, 5=essential	Weight: 2	Number: 1	Rationale: 2 Funcionalidad de colaboración básica con impacto limitado en la distribución del sistema.				
Response time or throughput performance objectives	0=no important, 5=essential	1	2	A pesar de que es beneficioso para la experiencia de usuario, no es estrictamente esencial.				
End-user online efficiency	0=no important, 5=essential	1	4	4 Se prioriza una interacción fluida y eficiente para el usuario dentro del alcance del proyecto.				
Complex internal processing	0=no important, 5=essential	1	3	El procesamiento de la lógica, su ejecución y depuración, conjuntados con la traducción a código y resolución de problemas implican un procesamiento interno intermedio.				
Reusability of code	0=no important, 5=essential	1	3	1 La reusabilidad es deseable pero no un foco primario.				
Easy to install	0=no important, 5=essential	1	1	0.5 Al ser una aplicación web, es accesible desde cualquier navegador y dispositivo compatible.				
Ease of use	0=no important, 5=essential	0.5	1	2.5 El sistema debe ser intuitivo y fácil de usar para estudiantes y profesores de todos los orígenes.				
Portability	0=no important, 5=essential	0.5	5	8 Alta portabilidad al ser una aplicación web estándar, con la intención de ser compatible con navegadores.				
Ease of change	0=no important, 5=essential	2	4	1 No se esperan cambios frecuentes debido al estándar de diagramas de flujo.				
Concurrency	0=no important, 5=essential	1	1	1 Conurrencia básica para funcionalidad de colaboración, similar al impacto en distribución.				
Special security objectives included	0=no important, 5=essential	1	1	2 La seguridad es necesaria para evitar ataques en el navegador, pero no se transmiten datos sensibles.				
Direct access for third parties	0=no important, 5=essential	1	2	No se prevé una interacción extensa con terceros para el alcance actual, pero puede ser beneficiosa para una mejor experiencia de usuario y de aprendizaje.				
Special User training required	0=no important, 5=essential	1	1	1 La complejidad del sistema se basa en el conocimiento sobre diagramas de flujo.				
<b>Technical Factor Value (TFactor)</b>			29					
<b>Technical Complexity Factor (TCF) = 0.6 + (0.01 * TFactor)</b>			0.89					
<b>Environmental Factors (affected by the team)</b>								
Familiarity with system development process being used	Scale 0 to 5	Weight	Number	Value				
0 = no experience, 3 = mid experience, 5= expert	1.5	4	6	Rationale: El equipo tiene bases en desarrollo de software, pudiendo alcanzar un nivel medio en el que comprende los procesos generales, pero aún no tiene la experiencia para considerarse experto.				
Application experience	0 = no experience, 3 = mid experience, 5= expert	0.5	4	2 Experiencia relevante a través de múltiples proyectos académicos y autoaprendizaje en tecnologías web.				
Object-oriented experience	0 = no experience, 3 = mid experience, 5= expert	1	4	4 Sólida comprensión y aplicación de OOP adquirida en el grado y proyectos previos.				
Lead analyst capability	0 = no experience, 3 = mid experience, 5= expert	0.5	3	1.5 Capacidad de análisis y diseño de sistemas desarrollada a través de la formación del grado.				
Motivation	0 = none, 3 = mid, 5= high	1	5	5 Alto interés personal y académico en el tema del proyecto, asegurando una fuerte motivación.				
Requirements stability	0 = completely unstable, 5 = completely stable	2	4	8 Requerimientos centrales bien definidos y estables desde la fase de planificación, aunque en la práctica pueden variar.				
Part time staff	0 = full time, 5 = part time	-1	2	Dedicatoria a tiempo parcial, pero con bloques de tiempo concentrados y bien gestionados a pesar de la -2 compaginación con estudios.				
Difficulty of programming language	0 = easy, 3 = mid, 5=difficult	-1	1	-1 TypeScript y su ecosistema ofrecen alta productividad y facilidad de aprendizaje para el proyecto.				
<b>Environmental Factor Value (EFactor)</b>			23.5					
<b>Environmental Factor (EF) = 1.4 + (-0.03 * EFactor)</b>			0.695					
<b>Adjusted Use Case Points (UCP) = UUCP * TCF * EF</b>								
Person Hours Multiplier (PhM) (Per use case)	* A value of 0 means too risky to proceed							
Effort in Person Hours whole project = UCP * PhM	51.33965							
Esfuerzo Meses Persona Estimados en el Proyecto	7 hours use-case							
359.37755 hours man								
2,246109688 MM								

Hours worked per month	160
<b>Phase</b>	
Analysis	15,00%
Design	20,00%
Coding	35,00%
Test	15,00%
<b>Extra (other activities)</b>	15,00%
<b>Phase</b>	
Analysis	53,9066325
Design	71,87551
<b>Coding</b>	<b>125,7821425</b>
Test	53,9066325
<b>Extra (other activities)</b>	53,9066325
<b>Total</b>	<b>359,37755</b>
<b>Hours.man</b>	
Analysis	0,33691
Design	0,44922
<b>Coding</b>	<b>0,78613</b>
Test	0,33691
<b>Extra (other activities)</b>	0,33691
<b>Total</b>	<b>2,24610</b>

## E. ANEXO: DOCUMENTO DE EVALUACIÓN ASÍNCRONA

**uc3m** | Universidad **Carlos III** de Madrid



### **Flowming**

“Aplicación web para el diseño y  
ejecución de diagramas de flujo”

### *Documento para la Evaluación de Usabilidad Asíncrona*

¡Muchas gracias por tu interés y tiempo!

Tu participación es fundamental para mejorar esta herramienta  
educativa.

---

Autor: **Daniel Pérez Fernández**

Tutor: **Álvaro Montero Montes**

## 1 ¿Qué es Flowming?

Flowming (*Flow + Programming*) es una aplicación web educativa diseñada para ayudar a estudiantes a aprender los fundamentos de la programación de una manera visual e interactiva. En lugar de escribir código, los usuarios pueden crear, ejecutar y depurar algoritmos utilizando diagramas de flujo estándar.

El objetivo principal es permitir que los nuevos programadores se concentren en la lógica y el pensamiento algorítmico, sin la barrera inicial de la sintaxis de un lenguaje de programación.

## 2 ¿Cómo se realizará la evaluación?

El proceso es muy sencillo y te llevará aproximadamente **15-20 minutos**. Sigue estos pasos:

1. **Consentimiento informado:** Antes de empezar, es necesario que leas y completes un breve formulario de consentimiento. Enlace al formulario: <https://forms.gle/TaSySmN6muTyH13S6>
2. **Accede a la aplicación:** Abre el siguiente enlace en tu navegador (preferiblemente Chrome): <https://dei.inf.uc3m.es/flowming/>.  
IMPORTANTE: Por favor, realiza la prueba en un ordenador, ya que la aplicación aún no está diseñada para tablets o móviles.
3. **Realiza las tareas:** Sigue las instrucciones de las 3 tareas descritas en la siguiente página. El objetivo no es que las hagas perfectamente, sino que interactúes con la herramienta.
4. **Completa el cuestionario:** Una vez hayas intentado realizar las tareas, por favor, completa el siguiente cuestionario para darnos tu opinión. Tu feedback es anónimo y muy valioso.

**Enlace al cuestionario:** <https://forms.gle/tkxjuR9M3pMNFAiZ6>

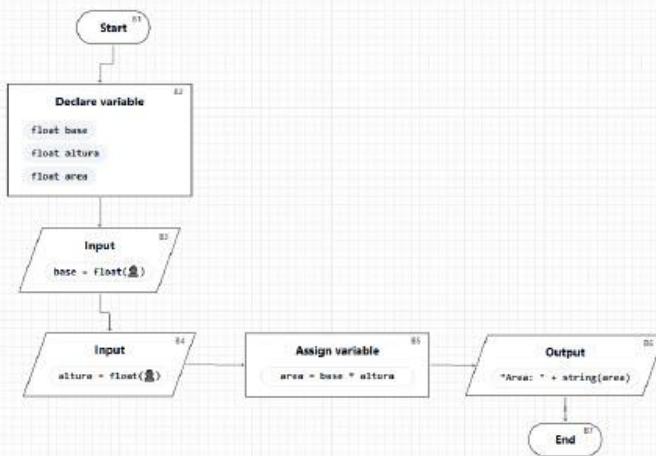
### 3 Tareas de Evaluación

Por favor, intenta realizar las siguientes tareas en el orden indicado.

#### 3.1 Tarea 1: Replicar y Ejecutar un Diagrama Existente

El objetivo de esta tarea es familiarizarte con la interfaz básica de la aplicación.

1. **Observa** el siguiente diagrama de flujo que calcula el área de un rectángulo:



2. **Replica** este diagrama en la aplicación. Arrastra los bloques desde la barra de herramientas de la izquierda y conéctalos. Para editar el contenido de un bloque, dale click y ve a la pestaña de “Details”.
3. **Ejecuta** el diagrama usando el botón de “Start”.
4. **Introduce** `base=10`, `altura=5` y **comprueba** que el resultado que se muestra sea el correcto: 50.

### 3.2 Tarea 2: Crear un Algoritmo para Números Pares

Ahora, crea un diagrama desde cero para resolver un problema.

1. **Diseña** un diagrama de flujo que muestre por pantalla los primeros 5 números pares (2, 4, 6, 8, 10).
  - **Pista:** Puedes crear un bucle al conectar una arista a un bloque de un paso anterior.
  - **Pista para saber si un número es par:** Un número es par si el resto de su división entre 2 es igual a 0. En programación, esto se suele escribir con el operador % (módulo). Por ejemplo,  $6 \% 2 = 0$ .

### 3.3 Tarea 3: Depurar y Generar Código

El objetivo de esta tarea es probar las herramientas complementarias.

1. Vuelve al diagrama de la **Tarea 2** y añade un **punto de interrupción (breakpoint)** en el bloque que muestra por pantalla un número par. Para ello, haz click derecho sobre el bloque y selecciona la opción correspondiente.
2. **Ejecuta** el programa de nuevo. La ejecución debería detenerse cada vez que se encuentre un número par.
3. Abre el **panel de depuración (Debugger)**. Observa cómo cambia el valor de tu variable contadora en el **historial de cambios (Variables)**.
4. Finalmente, abre el **panel de código (Code)**. Copia el código equivalente en Python e intenta ejecutarlo en un entorno de Python local u online (como [Online Python](#)) para verificar que funciona correctamente.

**¡Listo!**

Una vez hayas finalizado o intentado estas tareas, por favor, no te olvides de completar el cuestionario. ¡Muchísimas gracias por tu colaboración! 😊

Enlace al cuestionario: <https://forms.gle/tkxjuR9M3pMNFaiZ6>

## F. ANEXO: CUESTIONARIO DE LA EVALUACIÓN EN GOOGLE FORMS

### [TFG] Evaluación de Usabilidad de la Aplicación Flowming

Gracias por probar Flowming. Este cuestionario es anónimo y nos ayudará a mejorar la herramienta. Por favor, responde a las siguientes preguntas basándote en tu experiencia al realizar las tareas.

\* Indica que la pregunta es obligatoria



#### Sección 1: System Usability Scale (SUS)

Para cada una de las siguientes afirmaciones, indica tu grado de acuerdo.

1. Creo que me gustaría usar esta aplicación con frecuencia. \*

1    2    3    4    5

Totalmente en desacuerdo

Totalmente en acuerdo

2. Encontré la aplicación innecesariamente compleja. \*

1    2    3    4    5

Totalmente en desacuerdo

Totalmente en acuerdo

3. Me pareció que la aplicación era fácil de usar. \*

1    2    3    4    5

Totalmente en desacuerdo      Totalmente en acuerdo

4. Creo que necesitaría la ayuda de una persona con conocimientos técnicos para poder usar esta aplicación. \*

1    2    3    4    5

Totalmente en desacuerdo      Totalmente en acuerdo

5. Encontré las diversas funciones de esta aplicación bien integradas. \*

1    2    3    4    5

Totalmente en desacuerdo      Totalmente en acuerdo

6. Me pareció que había demasiada inconsistencia en esta aplicación. \*

1    2    3    4    5

Totalmente en desacuerdo      Totalmente en acuerdo

7. Imagino que la mayoría de la gente aprendería a usar esta aplicación muy rápidamente. \*

1    2    3    4    5

Totalmente en desacuerdo      Totalmente en acuerdo

8. Encontré la aplicación muy incómoda/engorrosa de usar. \*

1    2    3    4    5

Totalmente en desacuerdo      Totalmente en acuerdo

9. Me sentí muy seguro/a usando la aplicación. \*

1    2    3    4    5

Totalmente en desacuerdo      Totalmente en acuerdo

10. Necesité aprender muchas cosas nuevas antes de poder ser eficaz con esta \* aplicación.

1    2    3    4    5

Totalmente en desacuerdo      Totalmente en acuerdo

### Sección 2: Preguntas Abiertas

Por favor, responde brevemente a las siguientes preguntas. Tu opinión sincera es lo más importante.

¿Qué fue lo que más te gustó o te pareció más fácil de usar en la aplicación?

Tu respuesta

¿Qué fue lo que más te costó, te resultó más confuso o mejorarías de la aplicación?

Tu respuesta

¿Tienes algún otro comentario o sugerencia?

Tu respuesta

¿Has encontrado algún error por parte de la aplicación durante tus pruebas?

Tu respuesta

Enviar

Página 1 de 1

Borrar formulario

¿Tienes algún otro comentario o sugerencia?

Tu respuesta

¿Has encontrado algún error por parte de la aplicación durante tus pruebas?

Tu respuesta

Enviar

Página 1 de 1

Borrar formulario

## G. ANEXO: FORMULARIO DE CONSENTIMIENTO INFORMADO

### Formulario de Consentimiento Informado - Evaluación de Flowming

#### Propósito y descripción de la evaluación

El propósito de esta evaluación es estudiar la usabilidad de "Flowming", una aplicación web educativa para el aprendizaje de la programación mediante diagramas de flujo. El objetivo es identificar los beneficios y las posibles áreas de mejora de la herramienta desde la perspectiva del usuario.

La evaluación consistirá en la realización de una serie de tareas utilizando la aplicación. Antes de comenzar, se le proporcionará un documento con las instrucciones. Al finalizar las tareas, se le pedirá que rellene un cuestionario de usabilidad y de opinión. La duración estimada de todo el proceso es de 15-20 minutos.

#### Confidencialidad

Durante la evaluación, sus respuestas en los cuestionarios serán recopiladas. En el caso de participar en una sesión síncrona, la sesión será grabada (audio y vídeo de la pantalla compartida) para su posterior análisis. De cualquier manera, toda la información recopilada será almacenada y cifrada/protegida con una contraseña para protegerla y codificada con el fin de proteger su anonimidad en cualquier documento o presentación resultante de este trabajo.

#### Registro del consentimiento

Al marcar la casilla de abajo, usted indica que ha comprendido la información al respecto del consentimiento y consiente su participación. La participación es de carácter voluntario y le permite negarse a responder ciertas preguntas del cuestionario o retirarse del estudio en cualquier momento sin ningún tipo de penalización. Usted recibirá una copia del formulario de consentimiento.

Si tiene alguna duda o pregunta al respecto, por favor, contacte con Daniel Pérez Fernández en 100472350@alumnos.uc3m.es.

\* Indica que la pregunta es obligatoria

#### Correo electrónico \*

Registrar 100472350@alumnos.uc3m.es como el correo que se incluirá al enviar mi respuesta

Consentimiento de participación \*

- He leído y comprendido la información anterior y acepto voluntariamente participar en la evaluación.

Nombre del participante \*

Tu respuesta

---

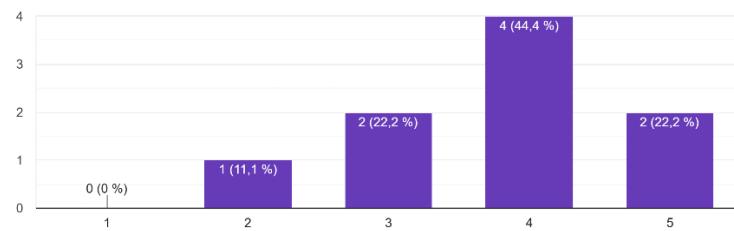
Una copia de tus respuestas se enviará por correo a 100472350@alumnos.uc3m.es.

[Enviar](#)

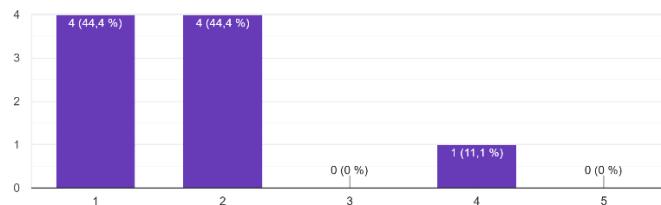
[Borrar formulario](#)

## H. RESULTADOS DESGLOSADOS DEL CUESTIONARIO DE USABILIDAD (SUS)

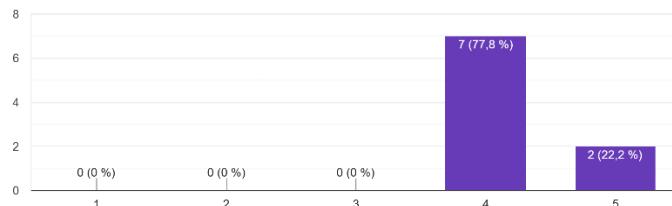
1. Creo que me gustaría usar esta aplicación con frecuencia.  
9 respuestas



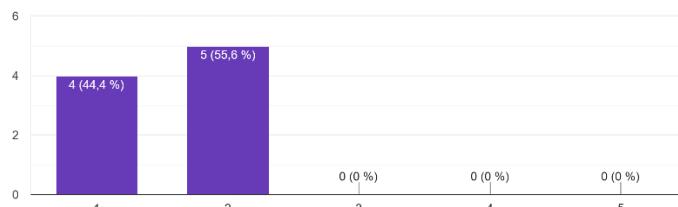
2. Encontré la aplicación innecesariamente compleja.  
9 respuestas



3. Me pareció que la aplicación era fácil de usar.  
9 respuestas

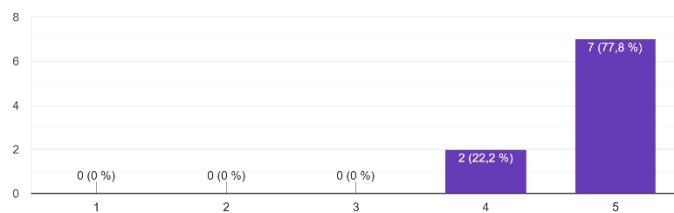


4. Creo que necesitaría la ayuda de una persona con conocimientos técnicos para poder usar esta aplicación.  
9 respuestas



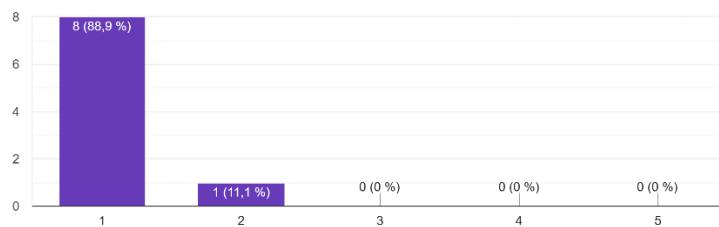
5. Encontré las diversas funciones de esta aplicación bien integradas.

9 respuestas



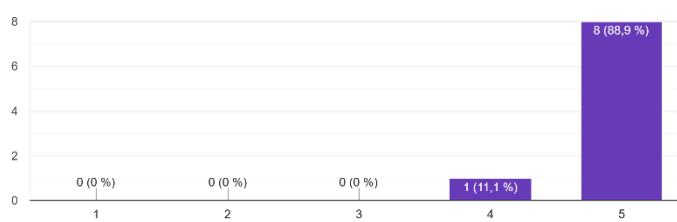
6. Me pareció que había demasiada inconsistencia en esta aplicación.

9 respuestas



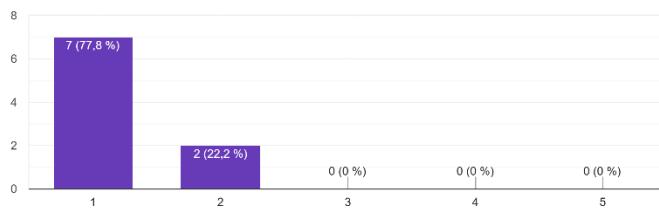
7. Imagino que la mayoría de la gente aprendería a usar esta aplicación muy rápidamente.

9 respuestas



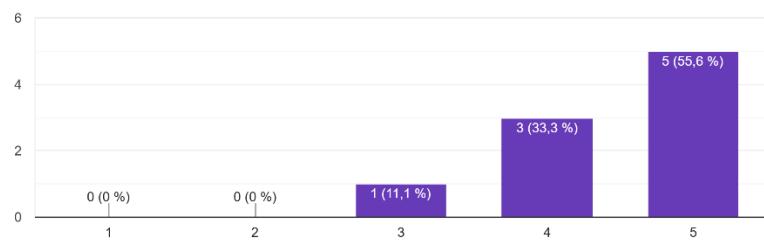
8. Encontré la aplicación muy incómoda/engorrosa de usar.

9 respuestas



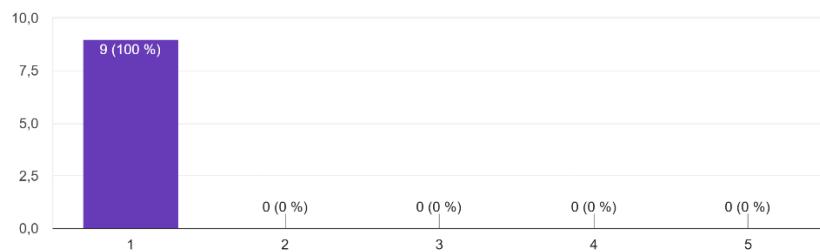
9. Me sentí muy seguro/a usando la aplicación.

9 respuestas



10. Necesité aprender muchas cosas nuevas antes de poder ser eficaz con esta aplicación.

9 respuestas



## I. ANEXO: DECLARACIÓN DE IA



### DECLARACIÓN DE USO DE IA GENERATIVA EN EL TRABAJO DE FIN DE GRADO

He usado IA Generativa en este trabajo  
*Marca lo que corresponda:*

SI  NO

SI

Si has marcado SI, completa las siguientes 3 partes de este documento:

#### Parte 1: reflexión sobre comportamiento ético y responsable

Ten presente que el uso de IA Generativa conlleva unos riesgos y puede generar una serie de consecuencias que afecten a la integridad moral de tu actuación con ella. Por eso, te pedimos que contestes con honestidad a las siguientes preguntas (*Marca lo que corresponda*):

Pregunta			
1. En mi interacción con herramientas de IA Generativa he remitido <b>datos de carácter sensible</b> con la debida autorización de los interesados.			
Sí, he usado estos datos con autorización	NO, he usado estos datos sin autorización	NO, no he usado datos de carácter sensible	
2. En mi interacción con herramientas de IA Generativa he remitido <b>materiales protegidos por derechos de autor</b> con la debida autorización de los interesados.			
Sí, he usado estos materiales con autorización	NO, he usado estos materiales sin autorización	NO, no he usado materiales protegidos	
3. En mi interacción con herramientas de IA Generativa he remitido <b>datos de carácter personal</b> con la debida autorización de los interesados.			
Sí, he usado estos datos con autorización	NO, he usado estos datos sin autorización	NO, no he usado datos de carácter personal	

<p>4. Mi utilización de la herramienta de IA Generativa ha <b>respetado sus términos de uso</b>, así como los principios éticos esenciales, no orientándola de manera maliciosa a obtener un resultado inapropiado para el trabajo presentado, es decir, que produzca una impresión o conocimiento contrario a la realidad de los resultados obtenidos, que suplante mi propio trabajo o que pueda resultar en un perjuicio para las personas.</p>	
SI	NO

Si NO has contado con la autorización de los interesados en alguna de las preguntas 1, 2 ó 3, explica brevemente el motivo (*por ejemplo, "los materiales estaban protegidos pero permitían su uso para este fin" o "los términos de uso, que se pueden encontrar en esta dirección (...), impiden el uso que he hecho, pero era imprescindible dada la naturaleza del trabajo"*).

#### Parte 2: declaración de uso técnico

Utiliza el siguiente modelo de declaración tantas veces como sea necesario, a fin de reflejar todos los tipos de iteración que has tenido con herramientas de IA Generativa. Incluye un ejemplo por cada tipo de uso realizado donde se indique: [Añade un ejemplo].

Declaro haber hecho uso del sistema de IA Generativa (Gemini 2.5 Pro Preview & Gemini Imagen 3.0) para:

#### Documentación y redacción:

- Revisión párrafos redactados previamente

[Añade un ejemplo] Por ejemplo: He solicitado la revisión de párrafos para asegurar que no hubiera errores de forma rápida.

- Búsqueda de información o respuesta a preguntas concretas

**[Añade un ejemplo]** Por ejemplo: He solicitado una lista de herramientas similares a las del proyecto para obtener una visión global de las herramientas competencias a profundizar en el Estado del Arte y sus artículos académicos correspondientes (complementado con búsqueda, verificación y profundización manual).

- Búsqueda de bibliografía
- Resumen de bibliografía consultada

#### **Desarrollar contenido específico**

Se ha hecho uso de IA Generativa como herramienta de soporte para el desarrollo del contenido específico del TFG, incluyendo:

- Asistencia en el desarrollo de líneas de código (programación)

**[Añade un ejemplo del flujo habitual de uso para programar]** Por ejemplo: He usado la IA Generativa para acelerar el proceso de refactorizaciones.

- Generación de esquemas, imágenes, audios o videos

**[indicar su uso en este apartado, así como mediante una nota al pie de la figura/elemento generado por IA Generativa]** Refinación de boceto de logotipo para obtener una versión más profesional.

- Procesos de optimización

**[Añade un ejemplo del flujo habitual de uso para optimizar]** Por ejemplo: He pedido explicaciones sobre errores relacionados con el estado de React y sus posibles causas.

### **Parte 3: reflexión sobre utilidad**

Por favor, aporta una valoración personal (formato libre) sobre las fortalezas y debilidades que has identificado en el uso de herramientas de IA Generativa en el desarrollo de tu trabajo. Menciona si te ha servido en el proceso de aprendizaje, o en el desarrollo o en la extracción de conclusiones de tu trabajo.

La mayor fortaleza que me ha aportado el uso de la IA Generativa ha sido un gran incremento en la productividad y eficiencia. Sin ella, es muy posible que no me hubiese dado el tiempo para incluir algunas funcionalidades que inicialmente habían sido planeadas. De forma específica, su habilidad para leer todo el código y saber de forma exacta dónde encontrar ciertas líneas fue clave en procesos de refactorización o depuración. Además, me ha ayudado a encontrar pequeños errores en la redacción que el detector de Word no había detectado ya que técnicamente eran palabras correctas, pero no las que quería incluir.

Como debilidad, noté que ni la refactorización ni la revisión de errores son siempre fiables. Muchas veces acabé haciéndolo manualmente ya que es muy posible que se deje algo que, aunque pequeño, crucial, sobre todo en casos donde hay muchos archivos de código.

Definitivamente me ha servido para aprender y profundizar más, sobre todo en los errores encontrados durante el proceso de desarrollo, sus causas y el motivo subyacente de ellas.