



ESCUELA DE INGENIERÍA DE FUENLABRADA

INGENIERÍA TELEMÁTICA

TRABAJO FIN DE GRADO

CODE-XR: PLUGIN DE VS CODE PARA EL ANÁLISIS DE
CÓDIGO EN REALIDAD EXTENDIDA

Autor : Adrián Montes Linares

Tutor : Dr. David Moreno Lumbreras

Curso académico 2025/2026



©2025 Adrián Montes Linares

Algunos derechos reservados

Este documento se distribuye bajo la licencia

“Atribución-CompartirIgual 4.0 Internacional” de Creative Commons,

disponible en

<https://creativecommons.org/licenses/by-sa/4.0/deed.es>

*Dedicado a
mi familia / mi abuelo / mi abuela*

Agradecimientos

Aquí vienen los agradecimientos... Aunque está bien acordarse de la pareja, no hay que olvidarse de dar las gracias a tu madre, que aunque a veces no lo parezca disfrutará tanto de tus logros como tú... Además, la pareja quizás no sea para siempre, pero tu madre sí.

Resumen

Code-XR es una propuesta pensada para cambiar la forma en que los desarrolladores analizan y entienden su código en tiempo real. Se trata de un plugin para Visual Studio Code, que permite visualizar a tiempo real métricas avanzadas del código como la complejidad ciclométrica entre otras métricas como media de parámetros por función, número de líneas totales del fichero...

Lo que realmente lo hace distinto es su integración con entornos de realidad extendida. Gracias a esto, el análisis de proyectos complejos se vuelve más intuitivo, inmersivo y accesible, incluso para quienes no están acostumbrados a interpretar métricas a simple vista.

La lógica principal del plugin está implementada en TypeScript, lo que permite integrar funcionalidades avanzadas directamente en Visual Studio Code y aprovechar al máximo su API. Para la visualización inmersiva, Code-XR utiliza A-Frame, un framework especializado en la creación de entornos de realidad virtual en la web, y BabiaXR, una herramienta que permite representar datos analíticos en gráficos 3D dentro del navegador. Las métricas de código se calculan mediante Python, mientras que la construcción de interfaces se realiza con HTML, CSS y JavaScript.

En resumen, Code-XR no es solo una herramienta técnica. Es una nueva forma de conectar con el código, de entenderlo y de trabajar con él de forma más clara, más visual y mucho más enriquecedora.

Summary

Code-XR is a proposal designed to change the way developers analyze and understand their code in real time. It's a plugin for Visual Studio Code that allows you to visualize advanced code metrics on the fly such as cyclomatic complexity, average number of parameters per function, total lines of code per file, and more.

What truly sets it apart is its integration with extended reality environments. Thanks to this, analyzing complex projects becomes more intuitive, immersive, and accessible even for those who aren't used to interpreting metrics at a glance.

The core logic of the plugin is implemented in TypeScript, allowing advanced functionality to be integrated directly into Visual Studio Code and making full use of its API. For immersive visualization, Code-XR leverages A-Frame, a framework specialized in building web-based virtual reality environments, and BabiaXR, a tool that enables analytical data to be represented as 3D graphics within the browser. Python is used to calculate code metrics, while HTML, CSS, and JavaScript are used for building the interface.

In short, Code-XR is more than just a technical tool. It's a new way to connect with your code making it clearer, more visual, and ultimately, a more enriching experience.

Índice general

1. Introducción	1
1.1. Contexto y motivación	1
1.2. Objetivo y aportación de Code-XR	2
1.3. Descripción general de Code-XR	3
1.4. Estructura de la memoria	4
2. Objetivos	7
2.1. Objetivo general	7
2.2. Objetivos específicos	7
2.3. Planificación temporal	9
3. Estado del arte	13
3.1. Visual Studio Code y su sistema de extensiones	13
3.2. BabiaXR	14
3.3. A-Frame	15
3.4. WebXR y Three.js	16
3.5. HTML	18
3.6. TypeScript y JavaScript	19
3.7. CSS	20
3.8. Python y Lizard	20
3.9. Gafas de Realidad Virtual: Meta Quest 3	21
3.10. Apoyo de herramientas de inteligencia artificial	22
4. Diseño e implementación	23
4.1. Arquitectura general	23

5. Experimentos y validación	25
6. Resultados	27
7. Conclusiones	29
7.1. Consecución de objetivos	29
7.2. Aplicación de lo aprendido	29
7.3. Lecciones aprendidas	30
7.4. Trabajos futuros	30
A. Manual de usuario	31
Bibliografía	33

Índice de figuras

2.1. Imagen de la reunión del 12 de marzo de 2025, donde se definieron los puntos principales del TFG.	12
3.1. Ejemplo de los diferentes gráficos de BabiaXR utilizados en Code-XR para la visualización de métricas.	15
4.1. Estructura del parser básico	24

Capítulo 1

Introducción

1.1. Contexto y motivación

Es ampliamente aceptado que, a medida que los proyectos de software crecen, entender su arquitectura interna y evaluar su calidad se vuelve cada vez más desafiante para los desarrolladores [7]. Las herramientas actuales de los entornos de desarrollo integrados (IDEs), como tablas, gráficos lineales o mapas de calor en 2D, ofrecen cierto apoyo al mostrar métricas estáticas, pero rara vez logran reflejar con claridad cómo se relacionan los distintos módulos del código. Esta limitación dificulta que los desarrolladores identifiquen, de forma rápida y precisa, problemas habituales de diseño, como la complejidad creciente o la duplicación de lógica [2, 9].

Ante estas limitaciones, la comunidad científica ha comenzado a explorar técnicas inmersivas como la Realidad Virtual (VR) y la Realidad Aumentada (AR), enmarcadas bajo el concepto más general de Realidad Extendida (XR). Estas tecnologías permiten representar el software en espacios tridimensionales, facilitando la interpretación espacial de artefactos y métricas [5]. Diversos trabajos han demostrado que el uso de entornos espaciales y la interacción corporal pueden mejorar tanto la comprensión como el nivel de compromiso del usuario [3].

Uno de los enfoques más influyentes ha sido el de la ciudad del software, introducido por herramientas como CodeCity [21] o ExplorViz [6], donde funciones, clases y módulos se representan como edificios y distritos de una ciudad tridimensional. Esta aproximación transforma métricas técnicas (como líneas de código, parámetros o complejidad ciclomática) en atributos visuales (altura, base o color), permitiendo una percepción más inmediata de las zonas problemáticas del sistema.

En este contexto, surge Code-XR, una herramienta que busca extender estas ideas al entorno real de trabajo de los desarrolladores: el propio IDE. El objetivo principal es integrar visualizaciones inmersivas y actualizadas en tiempo real directamente dentro de Visual Studio Code, sin necesidad de recurrir a dashboards externos ni cambiar de contexto. De este modo, se pretende facilitar una comprensión continua del estado del código durante el desarrollo, promoviendo un análisis más intuitivo y una toma de decisiones mejor fundamentada.

1.2. Objetivo y aportación de Code-XR

El objetivo principal de Code-XR es investigar y demostrar el potencial de la Realidad Extendida (XR) para representar métricas de código en tiempo real dentro del propio entorno de desarrollo, con el fin de facilitar la comprensión estructural del software y apoyar la toma de decisiones durante el desarrollo.

A diferencia de herramientas anteriores, que operan como sistemas independientes o requieren dashboards externos, Code-XR apuesta por una integración directa en el entorno de desarrollo (concretamente Visual Studio Code), permitiendo que los desarrolladores visualicen el impacto de sus cambios de forma continua mientras programan, sin necesidad de abandonar su flujo de trabajo [21, 6].

Su propuesta se basa en mapear propiedades estáticas del código como el número de líneas, parámetros o la complejidad ciclomática a atributos visuales tridimensionales en una ciudad virtual: la altura de un edificio representa el tamaño, la base refleja la cantidad de parámetros, y el color la complejidad [21]. Esta metáfora permite identificar de forma intuitiva patrones como zonas de alta complejidad, crecimiento irregular o duplicación de lógica, incluso en proyectos grandes.

Además, Code-XR introduce una segunda aportación relevante: el soporte opcional para la visualización inmersiva de la estructura DOM en aplicaciones web. Esto amplía el campo de aplicación de la herramienta más allá del análisis de código fuente, permitiendo explorar jerarquías HTML complejas en un espacio XR tridimensional [12].

En conjunto, las contribuciones principales de Code-XR son:

- La visualización inmersiva en tiempo real de métricas de código directamente en el IDE.

- El uso de tecnologías web XR accesibles, como A-Frame y BabiaXR, que facilitan su adopción sin necesidad de hardware especializado.
- La incorporación de modos de visualización alternativos (escritorio 3D, realidad aumentada y DOM) adaptables a diferentes contextos de uso.
- Una arquitectura ligera, extensible y configurable, pensada para ser utilizada tanto en entornos educativos como profesionales.

Estas aportaciones sientan las bases para futuras investigaciones empíricas que evalúen el impacto de las visualizaciones XR en la comprensión, la productividad y la calidad del software producido. También abren nuevas posibilidades en la colaboración entre desarrolladores, como sesiones de revisión de código compartidas en entornos virtuales [6].

1.3. Descripción general de Code-XR

Code-XR se integra directamente en Visual Studio Code, permitiendo al desarrollador analizar y visualizar métricas de código en tiempo real sin salir del entorno de desarrollo. El flujo de trabajo habitual comienza seleccionando un archivo o un directorio y lanzando el análisis desde el menú contextual. Las métricas extraídas líneas de código, número de parámetros, complejidad ciclomática entre otras métricas se actualizan automáticamente a medida que el usuario modifica el código.

Para ello, Code-XR emplea un servidor local que comunica los cambios en tiempo real usando Server-Sent Events (SSE), permitiendo que la representación visual se mantenga sincronizada sin necesidad de refrescar manualmente.

El sistema ofrece tres modos de visualización principales:

- **LivePanel:** muestra las métricas extraídas en una interfaz 2D tradicional tipo panel. Es útil para obtener una vista rápida, numérica o textual de los datos mientras se codifica.
- **XR Mode:** presenta una ciudad 3D donde cada función se representa como un edificio, con altura, base y color según sus métricas. Este modo es accesible tanto desde el navegador (modo escritorio) como desde un dispositivo de realidad aumentada (AR), conectándose al mismo servidor local [21, 6].

- **DOM Mode:** pensado para proyectos web, representa la estructura del DOM HTML en 3D, facilitando la exploración jerárquica y la detección de anidamientos complejos [12].

Todos los modos están diseñados para ser ligeros, accesibles y configurables, adaptándose a distintos contextos de uso (educativo, profesional, exploratorio).

Nota: Además del modelo de ciudad 3D, Code-XR permite alternar entre otros gráficos para las visualizaciones de los datos, gracias a la integración con BabiaXR [13].

1.4. Estructura de la memoria

Esta memoria se estructura de la siguiente manera:

- **Capítulo 1, Introducción:** Se presenta el contexto general del trabajo, las motivaciones que lo impulsan, el objetivo principal del proyecto y la aportación que supone Code-XR en el ámbito de la visualización de métricas de software. Además, se ofrece una descripción general de la herramienta y una guía sobre la estructura del documento.
- **Capítulo 2, Objetivos y planificación:** Se detallan los objetivos generales y específicos del proyecto, así como la planificación temporal seguida para su desarrollo, incluyendo una visión de los hitos principales y la metodología empleada.
- **Capítulo 3, Estado del arte:** Se analiza el estado del arte en visualización de software, con especial énfasis en técnicas inmersivas y herramientas previas relevantes. Se revisan también los enfoques más comunes para la representación de métricas estructurales en entornos de desarrollo.
- **Capítulo 4, Diseño e implementación:** Se describe el diseño e implementación de Code-XR, incluyendo su arquitectura, los componentes principales de la herramienta, las decisiones técnicas adoptadas y los diferentes modos de visualización que ofrece.
- **Capítulo 5, Validación del prototipo:** Se exponen los experimentos realizados para validar el prototipo, incluyendo los escenarios de prueba, los criterios de evaluación empleados y la documentación del comportamiento observado durante su uso en proyectos reales.

- **Capítulo 6, Resultados:** Se recogen los resultados obtenidos tras la evaluación, valorando en qué medida se han cumplido los objetivos propuestos y analizando las ventajas y limitaciones del enfoque adoptado.
- **Capítulo 7, Conclusiones y líneas futuras:** Se presentan las conclusiones generales del trabajo, se reflexiona sobre el aprendizaje obtenido durante el desarrollo del proyecto y se plantean posibles líneas de mejora o extensión futura de Code-XR.
- **Apéndices y bibliografía:** La memoria se complementa con un apéndice que incluye un manual de uso de la herramienta, así como la bibliografía consultada a lo largo

Capítulo 2

Objetivos

2.1. Objetivo general

El objetivo principal de este trabajo es desarrollar una herramienta de visualización de métricas de código en tiempo real, integrada en Visual Studio Code a modo de plugin usando tecnologías de realidad extendida (XR), que facilite la comprensión estructural del software durante el desarrollo.

El propósito es ofrecer una solución ligera, accesible y adaptable, capaz de representar propiedades clave del código como tamaño, complejidad o estructura jerárquica mediante metáforas visuales tridimensionales. Esta herramienta debe proporcionar retroalimentación continua sin abandonar el flujo de trabajo habitual del programador, fomentando así un análisis más intuitivo y dinámico de la calidad del software.

2.2. Objetivos específicos

Además del objetivo general, el desarrollo de Code-XR plantea una serie de objetivos específicos orientados a dotar a la herramienta de funcionalidades concretas y mejorar su aplicabilidad en entornos reales de desarrollo. Estos objetivos se resumen a continuación:

- **Diseñar una arquitectura modular para la extensión:** Establecer una estructura interna clara y escalable que permita añadir fácilmente nuevos lenguajes, métricas o modos de visualización sin modificar el núcleo del sistema.

- **Implementar un sistema de análisis estático de código:** Desarrollar un conjunto de scripts en Python que, utilizando la librería Lizard [18], extraigan métricas relevantes como líneas de código, número de parámetros y complejidad ciclomática a partir de archivos individuales o directorios completos.
- **Desarrollar una visualización 3D inmersiva con BabiaXR:** Construir una interfaz gráfica basada en BabiaXR, A-Frame y WebXR que represente los elementos del código como edificios dentro de una ciudad tridimensional. Esta interfaz debe soportar interacción en tiempo real, diferentes estilos visuales y ser accesible desde navegador o dispositivos XR.
- **Establecer comunicación en tiempo real mediante servidor local:** Lanzar un servidor HTTPS que gestione las peticiones de análisis y envíe los datos al cliente visual usando Server-Sent Events (SSE), permitiendo una actualización continua del modelo 3D sin necesidad de recargar la vista manualmente.
- **Integrar Code-XR dentro del entorno Visual Studio Code:** Utilizar la API oficial de VS Code para proporcionar comandos, menús contextuales y vistas personalizadas que permitan al usuario ejecutar el análisis y acceder a las visualizaciones sin abandonar el editor.
- **Ofrecer múltiples modos de visualización complementarios:** Desarrollar tres formas de explorar las métricas extraídas:
 - **LivePanel**, un panel 2D con métricas detalladas.
 - **XR Mode**, escenarios 3D accesibles vía navegador o dispositivos XR a través de localhost.
 - **DOM Mode**, una vista específica para estructuras HTML en entornos web.
- **Validar la herramienta en proyectos reales de software:** Aplicar Code-XR a repositorios reales con distintos lenguajes y estructuras, con el fin de comprobar su utilidad práctica, detectar patrones complejos y obtener retroalimentación sobre su usabilidad.

2.3. Planificación temporal

Este Trabajo Fin de Grado lo he desarrollado a lo largo de aproximadamente seis meses, compaginándolo con cinco asignaturas del segundo cuatrimestre. Debido a esta carga académica, la mayor parte del trabajo se concentró en fines de semana, festivos y semanas sin clases. A continuación, detallo cómo ha sido la planificación temporal real del proyecto:

Enero de 2025: El 22 de enero, tras finalizar el examen de la asignatura *Laboratorio de Bases de Datos*, mi profesor David Moreno Lumbreras me propuso la posibilidad de realizar el TFG con él. Aunque inicialmente no tenía previsto hacer el TFG ese curso, unos días después decidí escribirle para conocer mejor las posibles temáticas. El 24 de enero inicié el contacto formal por correo electrónico, y a finales de mes mantuvimos una primera reunión online para hablar sobre distintas líneas de trabajo.

Febrero de 2025: Durante este mes valoré varias propuestas que me ofreció mi tutor. El 13 de febrero le escribí indicándole las tres ideas que más me interesaban y le propuse una reunión presencial para discutir las en profundidad. Entre ellas estaba la opción de crear un plugin para Visual Studio Code, que me atrajo especialmente por ser el editor que he utilizado durante toda la carrera y por el reto de aprender nuevas tecnologías.

Marzo de 2025: El 4 de marzo me decidí finalmente por la opción de desarrollar un plugin para Visual Studio Code, y ese mismo día acordamos celebrar una reunión presencial para definir los primeros pasos del proyecto. La reunión se programó para el 12 de marzo.

Durante los días previos, estuve explorando cómo funcionan las extensiones de VS Code y desarrollé pequeños plugins de prueba. El 11 de marzo creé el repositorio de Code-XR y construí una primera versión muy básica del plugin, que lanzaba un servidor HTTP con una escena generada a partir de una plantilla sencilla usando A-Frame.

En la reunión del día 12 presenté esa versión inicial y, a partir de ella, definimos los cuatro pilares fundamentales del TFG. El primero fue implementar el soporte para servidores HTTPS, necesario para que las escenas pudieran visualizarse correctamente en dispositivos de realidad virtual por motivos de seguridad. También se acordó la integración con BabiaXR y el desarrollo del sistema de análisis de métricas, centrado inicialmente en calcular la complejidad ciclomática (CCN), las líneas de código (LOC) y el número de

funciones, entre otras. La figura 2.1 muestra una imagen de dicha reunión, donde quedaron recogidos los puntos principales del proyecto.

Abril de 2025: Entre el 3 y el 10 de abril implementé la funcionalidad para analizar ficheros de código, configurando entornos virtuales en Python y utilizando la librería Lizard para extraer métricas como líneas de código, parámetros y complejidad ciclomática. También desarrollé el modo LivePanel para visualizar estos datos en una interfaz 2D, como paso previo a la visualización en XR. El día 10 publiqué oficialmente el proyecto bajo el nombre Code-XR.

Durante la Semana Santa (del 10 al 20 de abril), me centré en una de las funcionalidades clave del proyecto: el análisis en activo. Implementé un servidor local con Server-Sent Events (SSE) que actualiza automáticamente las métricas en cuanto se detectan cambios en el código. También apliqué esta funcionalidad al modo LivePanel.

Mayo de 2025: Durante este mes no trabajé activamente en el proyecto debido a los exámenes finales de las asignaturas del cuatrimestre.

Junio de 2025: Retomé el trabajo del 20 al 23 de junio, añadiendo una nueva funcionalidad: VisualizeDOM, que permite representar la jerarquía DOM de un archivo HTML mediante el componente babia-html.

Julio de 2025: Durante el mes de julio desarrollé el análisis de directorios completos, tanto en modo XR como en modo LivePanel, con la opción de exploración profunda (deep) o superficial. Además, mejoré varias funcionalidades ya existentes:

- Permití seleccionar el gráfico BabiaXR a utilizar en cada análisis.
- Añadí opciones de configuración avanzadas en cada sección (lanzamiento de servidores, ver datos de un json de usuario y análisis de ficheros o directorios).
- Incorporé perfiles personalizados para guardar y reutilizar configuraciones.
- Soporte oficial para todos los lenguajes de programación soportados por Lizard.
- Mejoras varias como en la interfaz gráfica u otros aspectos técnicos.

Finalmente, el 29 de julio publiqué la versión 1.0.0 de Code-XR y creé la página web

oficial del proyecto [8], donde se incluyen vídeos demostrativos, una guía rápida de instalación y documentación completa sobre el funcionamiento del plugin.

Cabe destacar que muchas de estas tareas se solaparon en el tiempo, y que el ritmo de trabajo se adaptó a mi disponibilidad. Las reuniones con mi tutor se celebraron cuando era posible, en función del avance y del calendario académico.

TODO: Diagrama de Gantt de Code-XR

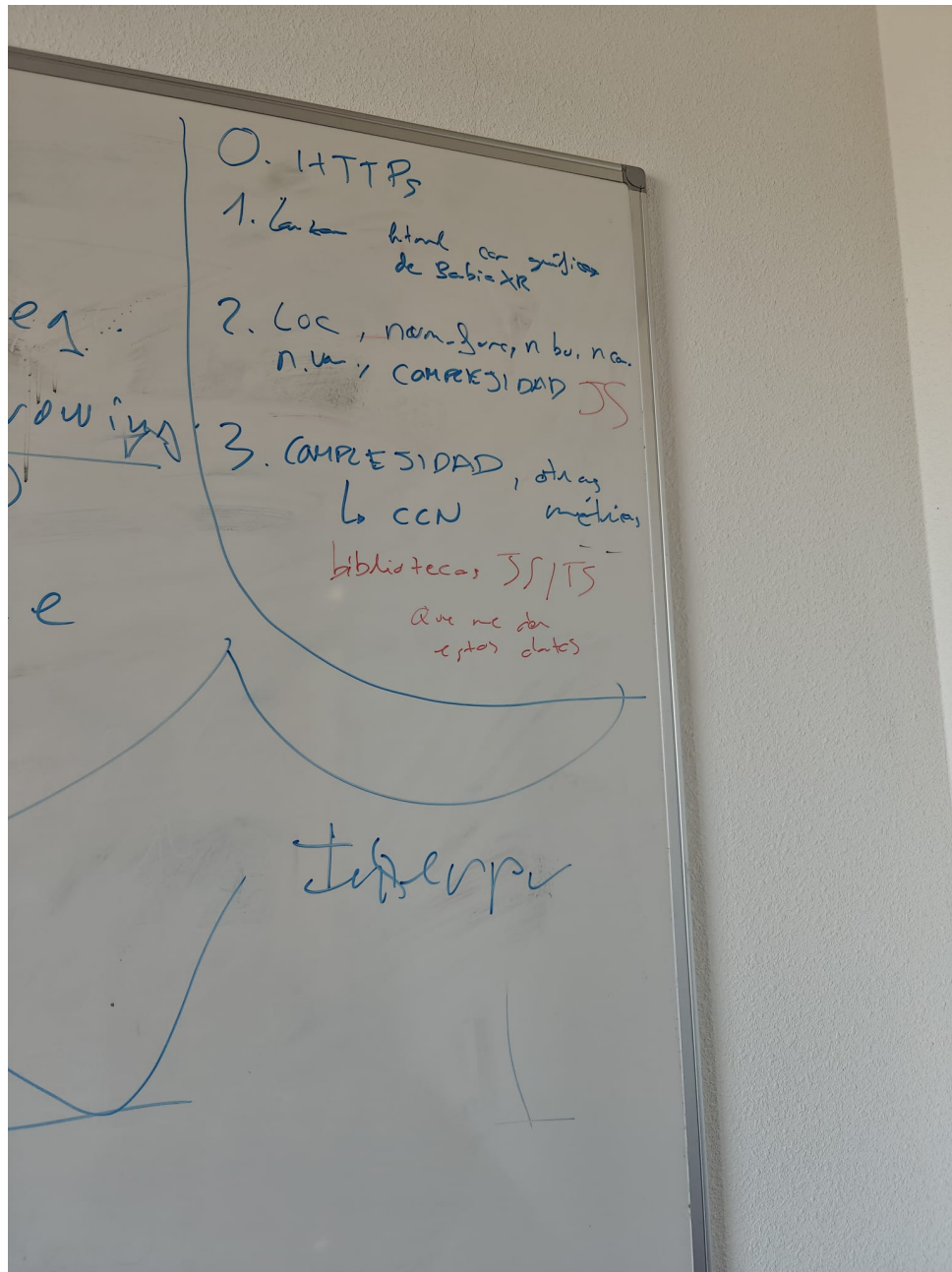


Figura 2.1: Imagen de la reunión del 12 de marzo de 2025, donde se definieron los puntos principales del TFG.

Capítulo 3

Estado del arte

3.1. Visual Studio Code y su sistema de extensiones

Visual Studio Code (VS Code) es un editor de código fuente desarrollado por Microsoft, ampliamente utilizado por la comunidad de desarrolladores debido a su ligereza, extensibilidad y compatibilidad con múltiples lenguajes de programación. Desde su lanzamiento, ha ganado una gran popularidad gracias a características como el autocompletado inteligente, el depurador integrado, la integración con Git y su ecosistema de extensiones.

Una de las características más potentes de VS Code es su sistema de extensiones, que permite ampliar las funcionalidades del editor mediante plugins desarrollados en tecnologías web, principalmente TypeScript, JavaScript, HTML y CSS. Estas extensiones pueden interactuar directamente con el entorno mediante la API oficial de VS Code [11], la cual expone funcionalidades como el acceso al árbol de archivos, la edición del contenido de los documentos, la creación de paneles personalizados, la ejecución de comandos o la escucha de eventos internos del editor.

En el contexto de este Trabajo Fin de Grado, he empleado VS Code no solo como entorno principal de desarrollo, sino también como plataforma de despliegue para la herramienta desarrollada: Code-XR. La elección de esta plataforma ha respondido tanto a motivos técnicos como personales. Por un lado, el ecosistema de extensiones de VS Code permite integrar la visualización de métricas directamente en el entorno de trabajo del programador, lo que evita saltos de contexto y mejora la fluidez del análisis. Por otro lado, se trata del editor que he utilizado a lo largo de toda mi formación universitaria, lo que favorece una mayor familiaridad con su

interfaz y flujo de trabajo.

Gracias al uso de la API de extensiones [11], se han podido registrar comandos personalizados, interceptar eventos como la apertura o modificación de archivos y lanzar servidores locales para comunicar el análisis del código con las interfaces XR que forman parte del sistema. El diseño modular de VS Code y su modelo basado en eventos lo convierten en un entorno especialmente adecuado para prototipar herramientas como Code-XR, que requieren integración en tiempo real con el proceso de edición de código.

3.2. BabiaXR

BabiaXR [13] es una plataforma de visualización tridimensional e inmersiva desarrollada por investigadores del Grupo de Sistemas y Comunicaciones (GSyC) de la Universidad Rey Juan Carlos (URJC). Está diseñada para facilitar la creación de escenas interactivas en realidad extendida (XR), incluyendo realidad virtual (VR) y aumentada (AR), utilizando tecnologías web abiertas y accesibles como A-Frame, Three.js y WebXR.

El objetivo de BabiaXR es proporcionar un entorno modular y fácilmente integrable que permita a desarrolladores e investigadores visualizar información compleja de forma espacial, sin necesidad de conocimientos avanzados de gráficos 3D. Gracias a su arquitectura basada en componentes personalizables, BabiaXR permite representar datos estructurados como visualizaciones 3D interactivas, directamente desde el navegador y sin necesidad de instalar software adicional. Esto lo convierte en una herramienta especialmente útil en contextos educativos y científicos, donde la simplicidad de despliegue y la portabilidad son factores clave.

En el contexto de este TFG, BabiaXR se ha utilizado como motor base para renderizar las visualizaciones XR de métricas de software generadas por el plugin Code-XR. Concretamente, se han empleado varios de sus componentes gráficos para representar las distintas métricas sobre ficheros o directorios.

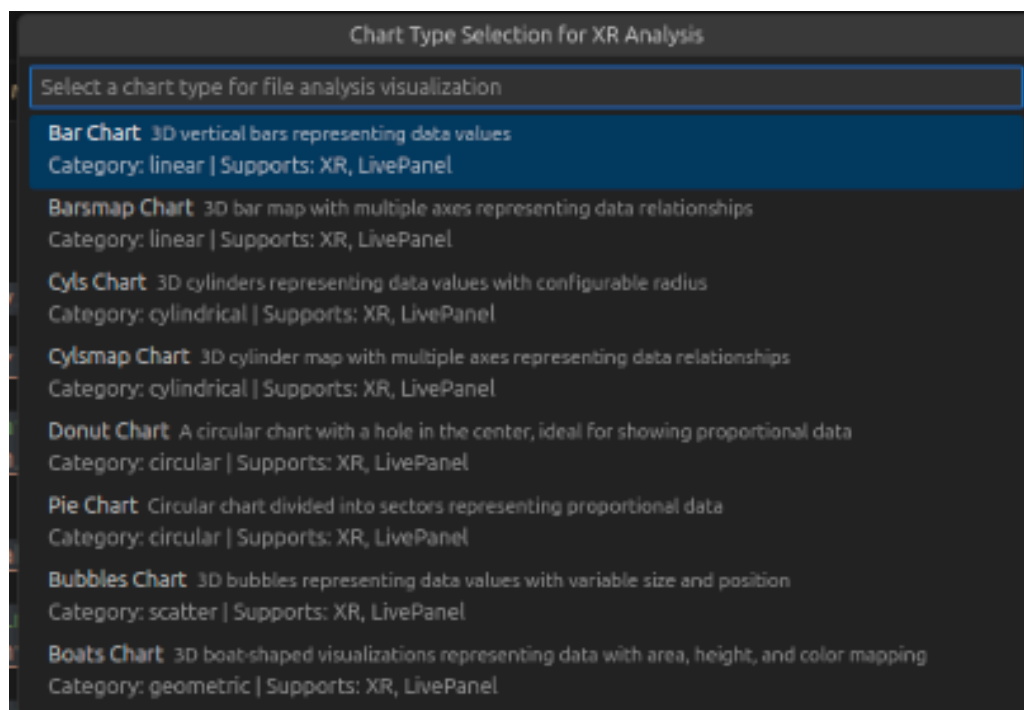


Figura 3.1: Ejemplo de los diferentes gráficos de BabiaXR utilizados en Code-XR para la visualización de métricas.

3.3. A-Frame

A-Frame es un framework de código abierto desarrollado inicialmente por Mozilla, orientado a la creación de experiencias inmersivas de realidad virtual y aumentada directamente desde el navegador [14]. Su principal objetivo es reducir la complejidad del desarrollo de escenas 3D, ofreciendo una sintaxis declarativa basada en HTML que permite a los desarrolladores definir entornos tridimensionales mediante etiquetas comprensibles y reutilizables.

Una de las principales ventajas de A-Frame es que se construye sobre Three.js, una biblioteca de bajo nivel para gráficos 3D en WebGL. Gracias a ello, A-Frame ofrece una capa de abstracción que simplifica el uso de luces, cámaras, geometrías y materiales, sin renunciar a la potencia gráfica que subyace bajo el capó. Además, A-Frame está diseñado de forma modular y extensible, lo que permite incorporar componentes personalizados y paquetes externos que enriquecen las capacidades básicas del framework.

En el contexto de este TFG, A-Frame actúa como motor de visualización principal dentro del sistema de representación XR de Code-XR. Toda la escena 3D que se genera al lanzar un

análisis con visualización inmersiva —ya sea en modo escritorio o mediante gafas de realidad virtual— está construida como una etiqueta `<a-scene>` de A-Frame. Dentro de esta escena, se cargan dinámicamente elementos como:

- el entorno (`<a-entity environment=...>`),
- los datos obtenidos del análisis (`<a-entity babia-queryjson=...>`),
- los gráficos 3D (como `<a-entity chart-bar>` o `<a-entity chart-city>`),
- los controladores para interacción con ratón o mandos XR (`laser-controls`, `raycaster`, etc.).

El sistema también incluye componentes como `movement-controls`, que permiten al usuario desplazarse libremente por la escena, y `cursor`, que habilita la interacción mediante puntero en modo escritorio. Esta estructura hace posible una experiencia inmersiva altamente personalizable, donde el usuario puede explorar las métricas del código como si caminara dentro de una ciudad tridimensional interactiva, cambiando parámetros o activando filtros en tiempo real.

Además, se han utilizado extensiones adicionales como:

- **aframe-extras**: para compatibilidad mejorada con mandos y controles de navegación,
- **aframe-environment-component**: para construir entornos virtuales preconfigurados,
- **aframe-geometry-merger-component**: para mejorar el rendimiento gráfico en escenas complejas.

Gracias a A-Frame, Code-XR consigue integrar visualización inmersiva XR basada en datos de análisis estático dentro del flujo de trabajo habitual del desarrollador, sin requerir instalaciones externas ni conocimientos especializados de gráficos 3D o motores complejos.

3.4. WebXR y Three.js

WebXR es un estándar impulsado por el World Wide Web Consortium (W3C) que proporciona una interfaz unificada para acceder a dispositivos de realidad virtual (VR) y aumentada

(AR) desde navegadores web modernos [20]. Esta API permite detectar dispositivos compatibles —como gafas de realidad virtual, controladores de movimiento o sensores de orientación— y renderizar contenido inmersivo directamente en el navegador, sin necesidad de plugins o instalaciones adicionales.

A diferencia de su predecesor, WebVR, el estándar WebXR está diseñado para cubrir tanto experiencias de realidad virtual como aumentada desde una misma API. Esto facilita el desarrollo de aplicaciones XR multiplataforma que funcionen de manera transparente en dispositivos como las Meta Quest, móviles compatibles con AR o incluso navegadores de escritorio. Gracias a WebXR, los entornos construidos con tecnologías web como A-Frame pueden ejecutarse en modo inmersivo y responder en tiempo real a la posición y orientación del usuario, ofreciendo una experiencia más natural e interactiva.

Aunque WebXR no se ha utilizado de forma directa en el código de Code-XR, sí está presente en la infraestructura base proporcionada por BabiaXR y A-Frame. La capacidad de lanzar visualizaciones XR directamente desde el navegador, y que estas puedan ser exploradas en dispositivos VR, se apoya en el soporte nativo de WebXR ofrecido por las capas subyacentes. Esto permite que el usuario final, sin necesidad de configurar nada adicional, pueda sumergirse en una visualización tridimensional del código simplemente accediendo a una URL generada por el plugin.

Por su parte, Three.js es una biblioteca JavaScript de alto nivel para el renderizado de gráficos 3D en WebGL [4]. Proporciona una interfaz sencilla para trabajar con cámaras, luces, materiales, geometrías y animaciones, y ha sido adoptada como base por muchos frameworks más accesibles como A-Frame. En otras palabras, Three.js actúa como el motor gráfico real que renderiza las escenas construidas en A-Frame, traduciéndolas a instrucciones de bajo nivel ejecutadas en la GPU mediante WebGL.

Aunque en este proyecto tampoco se ha programado directamente sobre la API de Three.js, su papel como capa de renderizado es esencial para que Code-XR pueda mostrar métricas en forma de ciudades tridimensionales, edificios interactivos o jerarquías DOM navegables. Gracias a la madurez y optimización de Three.js, la visualización generada es fluida, compatible con múltiples navegadores y capaz de mantener el rendimiento incluso en escenas complejas con decenas o cientos de elementos renderizados.

En resumen, tanto WebXR como Three.js son tecnologías que, aunque no se han utilizado

directamente en el código fuente desarrollado, sustentan gran parte de las capacidades inmersivas y gráficas de Code-XR a través de BabiaXR y A-Frame, y son piezas clave del ecosistema web XR moderno.

3.5. HTML

HTML (HyperText Markup Language) es el lenguaje de marcado fundamental para la estructuración de contenido en la web [22]. En el contexto de Code-XR, HTML actúa como la base común sobre la que se construyen todas las interfaces visuales de la herramienta, independientemente del modo de análisis utilizado: ya sea LivePanel, XR Mode o VisualizeDOM.

Cada vez que el usuario lanza un análisis, el sistema genera dinámicamente una plantilla HTML que se adapta al modo de visualización elegido. Estas plantillas incluyen la carga de scripts, componentes gráficos, referencias a los datos analizados y configuraciones específicas para el entorno XR o la interfaz 2D. Además, todas estas interfaces se sirven desde un servidor local (localhost), lo que permite que el usuario pueda abrirlas tanto en el navegador como desde un panel dentro del propio Visual Studio Code.

En el caso del modo LivePanel, HTML organiza la presentación de métricas en tablas, tarjetas o paneles interactivos, facilitando una exploración clásica y accesible de la información. Para el modo XR, el documento HTML contiene la escena tridimensional definida mediante A-Frame, donde los elementos `<a-scene>` y `<a-entity>` representan funciones o archivos del proyecto como estructuras visuales. En el modo VisualizeDOM, el contenido de un fichero HTML es analizado y su estructura DOM se representa como un árbol navegable en 3D, también dentro de una plantilla HTML generada.

La principal ventaja de utilizar HTML como base es su versatilidad y compatibilidad con el ecosistema web, lo que permite incorporar fácilmente bibliotecas, componentes personalizados y sistemas de interacción. Esta modularidad ha permitido a Code-XR mantener una arquitectura visual consistente, reutilizable y expandible, capaz de adaptarse a distintos modos de visualización sin cambiar la infraestructura técnica subyacente.

3.6. TypeScript y JavaScript

TypeScript es un lenguaje de programación desarrollado por Microsoft que extiende JavaScript con tipado estático opcional y características orientadas a objetos [10]. Se compila a JavaScript y es especialmente útil para desarrollar aplicaciones a gran escala. En el caso de Code-XR, TypeScript actúa como el motor principal del plugin para Visual Studio Code, gestionando tanto la lógica de control como la interacción con la API del editor.

Gracias a su sistema de tipos y su compatibilidad con las definiciones de la API oficial de VS Code, TypeScript permite escribir un código más robusto, mantenible y fácilmente depurable. En este TFG, se ha utilizado para estructurar los comandos del plugin, gestionar los flujos de análisis de ficheros o directorios, lanzar servidores locales, y registrar las distintas vistas disponibles (LivePanel, XR Mode, VisualizeDOM). También se encarga de orquestar cuándo y cómo se genera cada visualización, y de enviar los datos de análisis a las interfaces correspondientes.

Por su parte, JavaScript ha sido el lenguaje utilizado en la parte cliente (frontend) de la herramienta, desempeñando un papel fundamental en la actualización dinámica de las visualizaciones [15]. En particular, es el encargado de implementar el sistema de comunicación en tiempo real mediante Server-Sent Events (SSE). Cada vez que se detecta un cambio en el código fuente, el backend (escrito en TypeScript) actualiza el análisis y envía los datos mediante SSE, que son recibidos en el cliente JavaScript para actualizar la vista correspondiente.

En el caso concreto del modo LivePanel, JavaScript también es responsable de inyectar dinámicamente el contenido HTML que representa las métricas del análisis. Esto incluye la creación y actualización de elementos visuales como tarjetas, tablas o gráficos, en función de los datos recibidos. Gracias a esta lógica dinámica, la vista LivePanel puede reaccionar de forma instantánea a los cambios sin necesidad de recargar la página ni ejecutar ningún código adicional por parte del usuario.

La combinación de TypeScript para el control lógico del plugin y JavaScript para la actualización dinámica del frontend ha permitido construir una herramienta modular, eficiente y altamente interactiva. Ambos lenguajes forman parte del ecosistema web moderno y ofrecen un equilibrio ideal entre rendimiento, facilidad de desarrollo y mantenibilidad del código.

3.7. CSS

CSS (Cascading Style Sheets) es el lenguaje estándar utilizado para definir la presentación visual de documentos HTML [19]. Permite aplicar estilos a elementos web como colores, fuentes, márgenes, disposición de contenido o animaciones. En el contexto de Code-XR, CSS se utiliza exclusivamente para dar estilo a las interfaces generadas en el modo LivePanel, es decir, en las visualizaciones bidimensionales de métricas de código.

Cuando el usuario selecciona este modo de análisis, el sistema genera dinámicamente una plantilla HTML que contiene las métricas estructurales del código, como líneas de código, complejidad ciclomática o número de parámetros. Es en esa plantilla donde CSS entra en juego para mejorar la presentación de los datos y facilitar su interpretación. Por ejemplo, se utiliza para definir colores distintivos según la complejidad, organizar tarjetas o bloques métricos en una disposición clara, y aplicar estilos adaptativos que permitan visualizar correctamente la información en distintos tamaños de ventana.

Aunque CSS no participa en los modos inmersivos XR ni en la representación tridimensional del DOM, sí cumple un rol importante en la experiencia de usuario cuando se elige una visualización 2D. Gracias a su separación respecto a la lógica de análisis (TypeScript) y de comunicación (JavaScript), el uso de CSS ha permitido mantener una arquitectura limpia y modular, en la que el estilo puede modificarse de forma independiente sin afectar al funcionamiento interno del plugin.

3.8. Python y Lizard

Python es un lenguaje de programación de alto nivel, ampliamente utilizado en tareas de automatización, análisis de datos y procesamiento de texto [17]. En el marco de este proyecto, Python ha sido la tecnología seleccionada para realizar el análisis estático del código fuente, es decir, para extraer métricas estructurales que luego son representadas visualmente en los distintos modos de Code-XR.

La elección de Python responde a su sencillez sintáctica, su ecosistema de bibliotecas bien establecido y su capacidad para integrarse fácilmente con otros entornos. En este caso, la ejecución de los scripts de análisis se lanza desde el backend del plugin, escrito en TypeScript,

mediante servidores locales. Estos scripts procesan ficheros o directorios del proyecto analizado y generan como salida un fichero JSON con las métricas calculadas.

Para calcular métricas como la complejidad ciclomática (CCN), el número de líneas de código (LOC) o el número de funciones o métodos, se ha utilizado la herramienta Lizard [18], una biblioteca de análisis estático especializada y compatible con múltiples lenguajes de programación. Lizard permite procesar archivos fuente y devolver una representación estructurada de sus componentes, lo que ha facilitado la integración directa con el sistema de visualización de Code-XR.

Los datos generados por los scripts de Python son consumidos tanto por el modo LivePanel, que los muestra en formato 2D, como por el modo XR, donde se representan como edificios o elementos tridimensionales en una ciudad metafórica. En ambos casos, la estructura JSON permite mapear propiedades del código (como tamaño o complejidad) a atributos visuales como altura, color o volumen.

Además, en el modo VisualizeDOM, Python también cumple una función complementaria: leer el contenido del archivo HTML seleccionado y convertirlo a una cadena de texto. Esta cadena se pasa directamente al componente `babia-html` de BabiaXR, que se encarga de interpretarla y generar la visualización tridimensional correspondiente.

En conjunto, la combinación de Python y Lizard proporciona una base sólida para la obtención de datos estáticos fiables y bien estructurados, esenciales para alimentar las visualizaciones generadas por Code-XR en todos sus modos de funcionamiento.

3.9. Gafas de Realidad Virtual: Meta Quest 3

Para probar y validar las funcionalidades inmersivas del plugin Code-XR, se han utilizado unas gafas de realidad virtual Meta Quest 3, cedidas temporalmente por la Universidad Rey Juan Carlos. Estos dispositivos permiten ejecutar experiencias de realidad virtual de forma autónoma, gracias a su procesador integrado, sensores de movimiento y capacidad de renderizado gráfico sin necesidad de conexión a un ordenador externo.

Durante el desarrollo, las Meta Quest 3 se han empleado para acceder directamente a las visualizaciones XR generadas por Code-XR, conectándose al servidor local lanzado desde el entorno de desarrollo. Esto ha permitido verificar en entorno real el correcto funcionamiento

del modo XR, evaluar la experiencia de usuario y realizar ajustes en la navegación, disposición de los gráficos y rendimiento general de la escena.

El uso de estos dispositivos ha sido clave para validar el comportamiento del sistema en condiciones reales de uso inmersivo, especialmente en lo que respecta a la integración con WebXR y la capacidad de renderizar métricas complejas de código en escenarios tridimensionales navegables.

3.10. Apoyo de herramientas de inteligencia artificial

Durante el desarrollo del proyecto se ha utilizado el apoyo de herramientas basadas en inteligencia artificial, que han resultado útiles como complemento para resolver dudas técnicas, explorar alternativas de implementación y aprender nuevas tecnologías necesarias para llevar a cabo el trabajo.

En particular, ChatGPT [16] ha sido una herramienta de referencia para comprender el funcionamiento de bibliotecas como Lizard (usada para el análisis estático de código) o A-Frame (empleada en la construcción de escenas XR). También ha servido como apoyo a la hora de encontrar enfoques para resolver problemas concretos, mejorar la organización de ciertas secciones del código o redactar documentación técnica con un enfoque más claro y estructurado.

Por otro lado, se ha recurrido a Claude 4 [1] para tareas relacionadas con la organización de grandes volúmenes de código. Esto ha sido especialmente útil en este proyecto, cuyo repositorio cuenta con más de 200 ficheros solo en la carpeta `src/`, sin contar plantillas HTML, archivos CSS o scripts JavaScript asociados.

Estas herramientas no sustituyen el conocimiento técnico ni el razonamiento propio, pero sí han aportado valor como apoyo al desarrollo, ayudando a mejorar la eficiencia y la claridad en distintas partes del proyecto.

Capítulo 4

Diseño e implementación

Aquí viene todo lo que has hecho tú (tecnológicamente). Puedes entrar hasta el detalle. Es la parte más importante de la memoria, porque describe lo que has hecho tú. Eso sí, normalmente aconsejo no poner código, sino diagramas.

4.1. Arquitectura general

Si tu proyecto es un software, siempre es bueno poner la arquitectura (que es cómo se estructura tu programa a “vista de pájaro”).

Por ejemplo, puedes verlo en la figura 4.1. \LaTeX pone las figuras donde mejor cuadran. Y eso quiere decir que quizás no lo haga donde lo hemos puesto... Eso no es malo. A veces queda un poco raro, pero es la filosofía de \LaTeX : tú al contenido, que yo me encargo de la maquetación.

Recuerda que toda figura que añadas a tu memoria debe ser explicada. Sí, aunque te parezca evidente lo que se ve en la figura 4.1, la figura en sí solamente es un apoyo a tu texto. Así que explica lo que se ve en la figura, haciendo referencia a la misma tal y como ves aquí. Por ejemplo: En la figura 4.1 se puede ver que la estructura del *parser* básico, que consta de seis componentes diferentes: los datos se obtienen de la red, y según el tipo de dato, se pasará a un *parser* específico y bla, bla, bla. . .

Si utilizas una base de datos, no te olvides de incluir también un diagrama de entidad-relación.

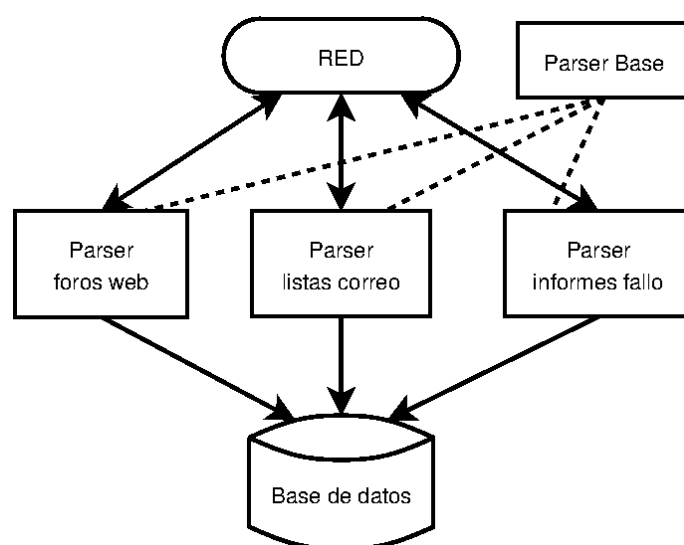


Figura 4.1: Estructura del parser básico

Capítulo 5

Experimentos y validación

Este capítulo se introdujo como requisito en 2019. Describe los experimentos y casos de test que tuviste que implementar para validar tus resultados. Incluye también los resultados de validación que permiten afirmar que tus resultados son correctos.

Capítulo 6

Resultados

En este capítulo se incluyen los resultados de tu trabajo fin de grado.

Si es una herramienta de análisis lo que has realizado, aquí puedes poner ejemplos de haberla utilizado para que se vea su utilidad.

Capítulo 7

Conclusiones

7.1. Consecución de objetivos

Esta sección es la sección espejo de las dos primeras del capítulo de objetivos, donde se planteaba el objetivo general y se elaboraban los específicos.

Es aquí donde hay que debatir qué se ha conseguido y qué no. Cuando algo no se ha conseguido, se ha de justificar, en términos de qué problemas se han encontrado y qué medidas se han tomado para mitigar esos problemas.

Y si has llegado hasta aquí, siempre es bueno pasarle el corrector ortográfico, que las erratas quedan fatal en la memoria final. Para eso, en Linux tenemos *aspell*, que se ejecuta de la siguiente manera desde la línea de *shell*:

```
aspell --lang=es_ES -c memoria.tex
```

7.2. Aplicación de lo aprendido

Aquí viene lo que has aprendido durante el Grado/Máster y que has aplicado en el TFG/TFM. Una buena idea es poner las asignaturas más relacionadas y comentar en un párrafo los conocimientos y habilidades puestos en práctica.

1. a

2. b

7.3. Lecciones aprendidas

Aquí viene lo que has aprendido en el Trabajo Fin de Grado/Máster.

1. Aquí viene uno.
2. Aquí viene otro.

7.4. Trabajos futuros

Ningún proyecto ni software se termina, así que aquí vienen ideas y funcionalidades que estaría bien tener implementadas en el futuro.

Es un apartado que sirve para dar ideas de cara a futuros TFGs/TFMs.

Apéndice A

Manual de usuario

Esto es un apéndice. Si has creado una aplicación, siempre viene bien tener un manual de usuario. Pues ponlo aquí.

Bibliografía

- [1] Anthropic. Claude 4. <https://claude.ai/>, 2024. Accedido el 3 de agosto de 2025.
- [2] T. Ball and S. G. Eick. Software visualization in the large. *Computer*, 29(4):33–43, 1996. <https://doi.org/10.1109/2.488299>.
- [3] A. Batch et al. There is no spoon: Evaluating performance, space use, and presence with expert domain users in immersive analytics. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):536–546, 2019.
- [4] R. y. c. Cabello. Three.js: Javascript 3d library. <https://threejs.org/>, 2024. Accedido el 3 de agosto de 2025.
- [5] C. Demiralp et al. Cave and fishtank virtual-reality displays: A qualitative and quantitative comparison. *IEEE Transactions on Visualization and Computer Graphics*, 12:323–330, 2006.
- [6] F. Fittkau, A. Krause, and W. Hasselbring. Exploring software cities in virtual reality. In *IEEE Working Conference on Software Visualization (VISsOFT)*, pages 130–134, 2015.
- [7] R. Koschke. Software visualization in software maintenance, reverse engineering, and re-engineering: a research survey. *Journal of Software Maintenance and Evolution: Research and Practice*, 15(2):87–109, 2003.
- [8] A. M. Linares. Oficial website of code-xr. <https://amontesl.github.io/code-xr-docs/>, 2025. Accedido el 3 de agosto de 2025.
- [9] B. Meyer. Seven principles of software testing. *Computer*, 41(8):99–101, 2008.

- [10] Microsoft. Typescript: Javascript with syntax for types. <https://www.typescriptlang.org/>, 2024. Accedido el 3 de agosto de 2025.
- [11] Microsoft. Visual studio code api. <https://code.visualstudio.com/api>, 2024. Disponible en <https://code.visualstudio.com/api>.
- [12] D. Moreno-Lumbreras. Enhancing html structure comprehension: Real-time 3d/xr visualization of the dom. In *2024 IEEE Working Conference on Software Visualization (VIS-SOFT)*, pages 127–132, 2024. <https://doi.ieeecomputersociety.org/10.1109/VISSOFT64034.2024.00025>.
- [13] D. Moreno-Lumbreras, J. M. Gonzalez-Barahona, and A. Villaverde. Babiaxr: Virtual reality software data visualizations for the web. In *2022 IEEE Conference on Virtual Reality and 3D User Interfaces Workshops (VRW)*, pages 71–74, 2022.
- [14] Mozilla. A-frame: A web framework for building virtual reality experiences. <https://aframe.io/>, 2024. Accedido el 3 de agosto de 2025.
- [15] Mozilla Developer Network. Javascript — mdn web docs. <https://developer.mozilla.org/docs/Web/JavaScript>, 2024. Accedido el 3 de agosto de 2025.
- [16] OpenAI. Chatgpt. <https://chat.openai.com/>, 2024. Accedido el 3 de agosto de 2025.
- [17] Python Software Foundation. Python programming language. <https://www.python.org/>, 2024. Accedido el 3 de agosto de 2025.
- [18] T. Tian. Lizard - an extensible cyclomatic complexity analyzer for many programming languages. <https://github.com/terryyin/lizard>, 2024. Accedido el 3 de agosto de 2025.
- [19] W3C. Css: Cascading style sheets. <https://www.w3.org/Style/CSS/>, 2024. Accedido el 3 de agosto de 2025.
- [20] W3C. Webxr device api. <https://www.w3.org/TR/webxr/>, 2024. Accedido el 3 de agosto de 2025.

- [21] R. Wettel and M. Lanza. Visualizing software systems as cities. In *IEEE International Workshop on Visualizing Software for Understanding and Analysis*, pages 92–99, 2007.
- [22] WHATWG. Html living standard. <https://html.spec.whatwg.org/>, 2024. Accedido el 3 de agosto de 2025.