Universidad
Rey Juan Carlos

ESCUELA DE INGENIERÍA DE FUENLABRADA

INGENIERÍA TELEMÁTICA

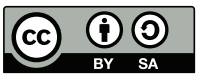
TRABAJO FIN DE GRADO

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



Dedicado a

mis padres

II

Agradecimientos

Quiero dar las gracias en primer lugar a mis padres, por su apoyo incondicional en todos los aspectos de la vida y por la educación y los valores que me han transmitido. A mi hermana, por estar siempre presente y aportar su compañía en los momentos importantes.

También a mis amigos de la universidad, con quienes he compartido años de esfuerzo, aprendizaje y grandes momentos dentro y fuera de las aulas.

A mi tutor, David, por su orientación, sus consejos y su ayuda constante durante el desarrollo de este proyecto.

Y, por último, a todas aquellas personas que, de una manera u otra, han formado parte de este camino y han contribuido a que hoy pueda presentar este trabajo

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 ciclomática 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 an idea proposed to change the way in which developers inspect and understand their program code in real time. It is a plugin for Visual Studio Code that allows you to view advanced code metrics on the fly such as cyclomatic complexity, average number of parameters per function, total number of lines per file, etc.

What makes Code-XR most distinctive is its ability to coexist with extended reality environments. Because of that, the interpretation of complex projects is simplified and becomes more intuitive and accessible for people who are not used to deciphering metrics at a glance.

The plugin's primary logic is developed in TypeScript so that advanced functionality is integrated directly into Visual Studio Code and the full power of its API is leveraged. For immersive visualizations, Code-XR uses A-Frame, an engine for authoring web-based virtual reality scenes, together with BabiaXR, a component for visualizing analytical data as 3D diagrams in the browser. Python computes the code metrics, while HTML, CSS and JavaScript are used to build the user interface.

In short, Code-XR is not just another tool. It provides an alternative means to interact with your code easier to read, easier to visualize, and ultimately an enriched development experience.

Índice general

Índice de figuras

Capítulo 1

Introducción

1.1. Contexto y motivación

Es ampliamente reconocido que, conforme los proyectos de software crecen, resulta cada vez más difícil para los desarrolladores comprender su arquitectura interna y evaluar su calidad [14]. Los recursos habituales de los IDE, como tablas, gráficos de líneas o mapas de calor en 2D, ayudan solo hasta cierto punto: muestran métricas estáticas, pero rara vez revelan con claridad cómo se interconectan los distintos módulos del código. Como consecuencia, localizar con rapidez y precisión problemas clásicos de diseño (incremento de la complejidad, duplicidades de lógica, etc.) se complica [2, 18].

Ante estas limitaciones, la comunidad científica ha explorado enfoques inmersivos como la Realidad Virtual (VR) y la Realidad Aumentada (AR), englobados bajo el paraguas de la Realidad Extendida (XR). Estas tecnologías permiten representar el software en entornos tridimensionales, facilitando una lectura espacial de artefactos y métricas [8]. La evidencia publicada sugiere que los entornos 3D 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 [38] o ExplorViz [12], 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, complejidad ciclomática, etc) en atributos visuales (altura, base, color, etc), 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 primario de Code-XR es explorar y demostrar la capacidad de la Realidad Extendida (XR) para representar en tiempo real métricas de código dentro del propio entorno de desarrollo. Su propósito es ofrecer mayor claridad estructural sobre el software y asistir en la toma de decisiones durante la fase de desarrollo.

A diferencia de otras herramientas afines, que suelen funcionar como sistemas independientes o dependen de dashboards externos, Code-XR apuesta por la integración inmediata en el entorno de desarrollo (en especial en VS Code). De este modo, permite que los desarrolladores visualicen en tiempo real el impacto de sus cambios sin interrumpir ni sacrificar su flujo de trabajo [38, 12].

La propuesta se basa en mapear propiedades estáticas del código como el número de parámetros o la complejidad ciclomática a atributos tridimensionales visibles. Un ejemplo paradigmático es la metáfora de la “ciudad navegable”: la altura de los edificios representa el tamaño, la base refleja la cantidad de parámetros y la coloración indica la complejidad ciclomática [38]. Esta visualización facilita identificar de manera intuitiva patrones como zonas con alta complejidad, progresos irregulares o redundancias lógicas, incluso en proyectos de gran escala.

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 [24].

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 visualizaciones en escenas XR que pueden experimentarse tanto en el navegador o visor integrado de VS Code como mediante gafas de realidad virtual.
- 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 [12].

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, se actualizan automáticamente a medida que el usuario modifica el código.

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 [38, 12].
- **DOM Mode**: pensado para proyectos web, representa la estructura del DOM HTML en 3D, facilitando la exploración jerárquica y la detección de anidamientos complejos [24].

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 [26].

1.4. Estructura de la memoria

- **Capítulo 1. Introducción:** Presenta Code-XR, las motivaciones y objetivos del proyecto, sus aportes a la visualización de métricas y una descripción general de la herramienta.
- **Capítulo 2. Objetivos y planificación:** Resume los objetivos generales y específicos del proyecto, la planificación temporal, los principales hitos alcanzados y la metodología adoptada.
- **Capítulo 3. Estado del arte:** Revisa investigaciones previas en visualización de software, con especial atención a los enfoques inmersivos y a los métodos de visualización de métricas de desarrolladores.
- **Capítulo 4. Diseño e implementación:** Explica el diseño y la implementación de Code-XR, detallando su arquitectura, componentes principales y modos de visualización.
- **Capítulo 5. Validación del prototipo:** Describe los experimentos realizados para validar el prototipo, incluyendo escenarios de prueba, criterios de evaluación y resultados obtenidos en proyectos reales.
- **Capítulo 6. Resultados:** Presenta los hallazgos del proyecto, analiza el grado de cumplimiento de los objetivos y expone los beneficios y limitaciones del enfoque seguido.
- **Capítulo 7. Conclusiones y líneas futuras:** Recoge las conclusiones finales, los aprendizajes derivados del desarrollo del proyecto y posibles mejoras o extensiones para Code-XR.
- **Apéndices y bibliografía:** Incluye el manual de uso de la herramienta y la bibliografía consultada.

Capítulo 2

Objetivos

2.1. Objetivo general

El objetivo de este trabajo es desarrollar un plugin para Visual Studio Code que, mediante tecnologías de realidad extendida (XR), permita visualizar métricas de código en tiempo real para facilitar la comprensión estructural del software. La herramienta busca ser ligera, accesible y adaptable, representando propiedades clave como tamaño, complejidad o jerarquía mediante metáforas tridimensionales, y ofreciendo retroalimentación continua sin interrumpir el flujo de trabajo del programador.

2.2. Objetivos específicos

Además del objetivo general, el desarrollo de Code-XR plantea los siguientes objetivos específicos:

- **Arquitectura modular:** Definir una estructura clara y escalable que facilite la incorporación de nuevos lenguajes, métricas o modos de visualización.
- **Análisis estático de código:** Implementar scripts en Python apoyados en Lizard [33] para extraer métricas en ficheros o directorios.
- **Visualización 3D inmersiva:** Desarrollar escenas con BabiaXR, A-Frame y WebXR que representen el código en ciudades 3D interactivas, accesibles desde navegador o dispositivos XR.

- **Comunicación en tiempo real:** Utilizar un servidor local con Server-Sent Events (SSE) para actualizar visualizaciones dinámicamente sin recargas manuales.
- **Integración con VS Code:** Incorporar comandos, menús y vistas personalizadas mediante la API oficial de VS Code.
- **Modos de visualización:** Ofrecer tres alternativas complementarias:
  - **LivePanel**: panel 2D con métricas detalladas, versión más clásica.
  - **XR Mode**: escenarios XR accesibles vía navegador o dispositivos de realidad virtual.
  - **DOM Mode**: visualización específica de estructuras HTML en escenarios XR.
- **Validación práctica:** Probar la herramienta en proyectos reales de software para comprobar su utilidad, detectar patrones y evaluar la usabilidad.

2.3. Planificación temporal

Este Trabajo Fin de Grado se desarrolló durante seis meses, compaginándolo con cinco asignaturas del segundo cuatrimestre. La mayor parte del trabajo se concentró en fines de semana, festivos y períodos sin clases.

Enero 2025: El 22 de enero, mi profesor David Moreno Lumbreras me propuso realizar el TFG. Tras el contacto inicial el 24 de enero, mantuvimos una primera reunión online para explorar líneas de trabajo.

Febrero 2025: Evalué las propuestas del tutor. El 13 de febrero seleccioné tres ideas de mayor interés, destacando el desarrollo de un plugin para Visual Studio Code por mi familiaridad con el editor y el desafío tecnológico.

Marzo 2025: El 4 de marzo decidí desarrollar el plugin de VS Code. Días previos a la reunión del 12 de marzo, exploré extensiones de VS Code y desarrollé prototipos. El 11 de marzo creé el repositorio Code-XR con una versión básica usando A-Frame. En la reunión presenté esta versión inicial y definimos los cuatro pilares fundamentales: soporte HTTPS para dispositivos VR, integración con BabiaXR, y análisis de métricas (CCN, LOC, número de funciones). La figura 2.1 documenta los puntos clave para el desarrollo del proyecto.

Abril 2025: Del 3 al 10 de abril implementé el análisis de ficheros usando Python y Lizard para extraer métricas, desarrollé el modo LivePanel (interfaz 2D) y publiqué oficialmente Code-XR el día 10. Durante Semana Santa (10-20 abril) implementé el análisis en tiempo real con Server-Sent Events (SSE) para actualización automática de métricas.

Mayo 2025: Pausa por exámenes finales del cuatrimestre.

Junio 2025: Del 20 al 23 de junio añadí VisualizeDOM para representar jerarquías HTML mediante babia-html.

Julio 2025: Desarrollé análisis de directorios completos (profundo/superficial), selección de gráficos BabiaXR, configuraciones avanzadas, perfiles personalizados y soporte completo para lenguajes de Lizard. El 29 de julio publiqué la versión 1.0.0 y creé la página web oficial del proyecto [16] con vídeos demostrativos y documentación completa.

Las tareas se solaparon temporalmente, adaptándose a la disponibilidad académica. Las reuniones con el tutor se programaron según el avance del proyecto.

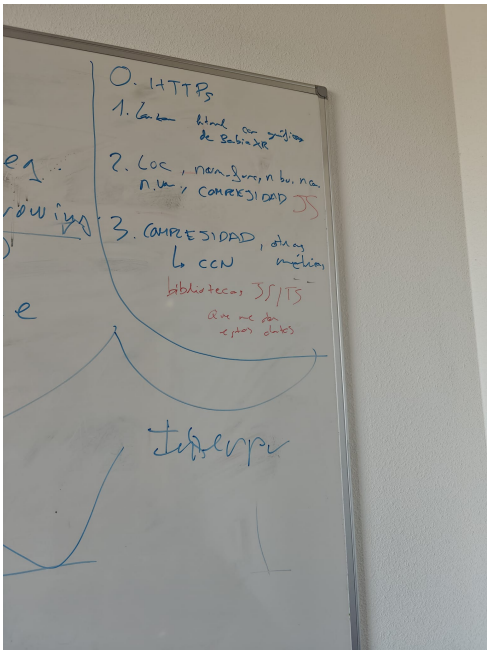


Figura 2.1: Imagen de la reunión del 12 de marzo de 2025, donde se definieron los puntos principales del TFG.

En la figura 2.2 se muestra el diagrama de Gantt con la planificación temporal detallada del proyecto, incluyendo las fases principales, tareas específicas e hitos clave desde enero hasta julio de 2025. Las franjas verdes representan las fases principales, las azules las tareas específicas y los rombos rojos los hitos clave.

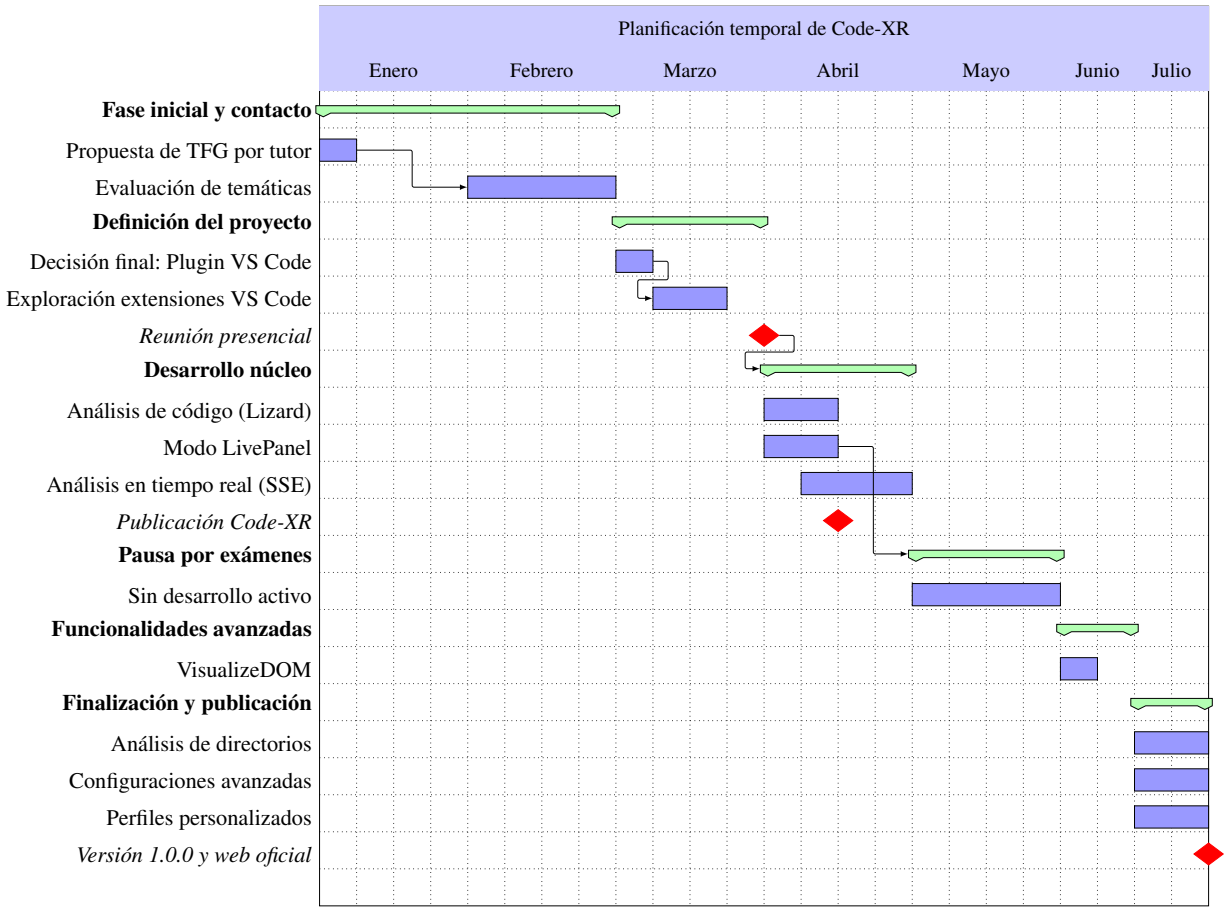


Figura 2.2: Diagrama de Gantt de la planificación temporal de Code-XR (enero-julio 2025).

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 adoptado por su ligereza, posibilidad de extensión y soporte para múltiples lenguajes. Su éxito se debe tanto a funcionalidades integradas (autocompletado, depuración, integración con Git) como a su ecosistema de extensiones.

El sistema de extensiones permite ampliar el editor mediante plugins basados en tecnologías web (TypeScript, JavaScript, HTML y CSS), que interactúan con el entorno a través de la API oficial de VS Code [21]. Esta API ofrece acceso al árbol de archivos, edición de documentos, creación de paneles personalizados, ejecución de comandos y gestión de eventos internos.

En este TFG, VS Code se ha empleado tanto como entorno de desarrollo como plataforma de despliegue de Code-XR. Su elección responde a dos motivos: técnico, al posibilitar la integración de visualizaciones de métricas directamente en el flujo de trabajo del programador sin cambios de contexto; y personal, por ser el editor utilizado a lo largo de mi formación académica.

Gracias a la API de extensiones [21], ha sido posible registrar comandos, interceptar eventos y lanzar servidores locales que conectan el análisis estático con las visualizaciones XR. El modelo modular y orientado a eventos de VS Code lo convierte en un entorno idóneo para herramientas como Code-XR, que requieren sincronización en tiempo real con el proceso de edición.

3.2. BabiaXR

BabiaXR [26] 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.

3.3. A-Frame

A-Frame es un framework de código abierto desarrollado por Mozilla para crear experiencias inmersivas de realidad virtual y aumentada desde el navegador [27]. Ofrece una sintaxis declarativa en HTML que simplifica la definición de escenas 3D, actuando como capa de abstracción sobre Three.js y permitiendo el uso de componentes modulares y extensibles.

En Code-XR, A-Frame constituye el motor principal de visualización XR: cada análisis inmersivo se representa en una escena `<a-scene>` que integra el entorno, los datos analizados, gráficos tridimensionales (bar, city, etc.) y controladores de interacción tanto para escritorio como para dispositivos XR. Extensiones como `aframe-extras`, `aframe-environment-component` y `aframe-geometry-merger-component` mejoran la navegación, los entornos predefinidos y el rendimiento en escenas complejas.

Gracias a esta infraestructura, Code-XR permite explorar métricas de código en entornos tridimensionales interactivos sin necesidad de conocimientos avanzados en gráficos 3D, ofreciendo una experiencia inmersiva directamente integrada en el flujo de trabajo del desarrollador.

3.4. WebXR y Three.js

WebXR es el estándar del W3C que unifica el acceso a dispositivos de realidad virtual y aumentada desde navegadores modernos [36]. Sustituye a WebVR al soportar VR y AR desde una misma API, permitiendo experiencias multiplataforma que responden en tiempo real a la posición y orientación del usuario. Aunque Code-XR no usa WebXR de forma explícita, sus capacidades están presentes a través de BabiaXR y A-Frame, lo que posibilita ejecutar visualizaciones XR inmersivas directamente en el navegador o en dispositivos como Meta Quest sin configuraciones adicionales.

Por su parte, Three.js es la biblioteca de JavaScript que facilita el renderizado 3D en WebGL [4], proporcionando primitivas para cámaras, luces, materiales o geometrías. A-Frame se apoya en ella como motor de bajo nivel, de modo que aunque no se use directamente en Code-XR, resulta esencial para representar métricas en forma de ciudades 3D, jerarquías DOM navegables o escenas con decenas de elementos manteniendo fluidez y compatibilidad en distintos navegadores.

En conjunto, WebXR y Three.js, aun sin estar programadas directamente en este TFG, constituyen la base tecnológica que hace posible la inmersión y el rendimiento gráfico de Code-XR a través de los frameworks que las encapsulan.

3.5. HTML

HTML (HyperText Markup Language) es el lenguaje de marcado esencial para estructurar contenido web [39] y constituye la base de todas las interfaces de Code-XR, ya sea en LivePanel, XR Mode o VisualizeDOM.

Cada vez que se lanza un análisis, el sistema genera dinámicamente una plantilla HTML que incluye scripts, referencias a datos y configuraciones específicas según el modo. Estas plantillas

se sirven desde un servidor local, accesibles tanto en navegador como en un panel de VS Code.

En LivePanel, HTML organiza métricas en tablas o tarjetas interactivas; en XR Mode define escenas 3D con A-Frame mediante elementos como `<a-scene>` y `<a-entity>`; y en VisualizeDOM representa la estructura del DOM como un árbol navegable.

El uso de HTML aporta versatilidad y compatibilidad con el ecosistema web, lo que facilita la incorporación de bibliotecas, componentes personalizados e interacciones, manteniendo una arquitectura visual coherente y extensible en todos los modos de la herramienta.

3.6. TypeScript y JavaScript

TypeScript, desarrollado por Microsoft, extiende JavaScript con tipado estático y características orientadas a objetos [20]. En Code-XR constituye el motor principal del plugin, encargado de la lógica de control, la integración con la API de VS Code, la gestión de análisis de ficheros y directorios, el lanzamiento de servidores locales y la orquestación de las visualizaciones (LivePanel, XR Mode, VisualizeDOM). Su sistema de tipos aporta robustez y mantenibilidad al proyecto.

JavaScript, por su parte, se utiliza en el cliente para la actualización dinámica de las visualizaciones [28]. Implementa la comunicación en tiempo real mediante Server-Sent Events (SSE), de modo que cada cambio en el código provoca la actualización inmediata de los datos en la interfaz. En el caso del LivePanel, se encarga de inyectar y actualizar dinámicamente el HTML con métricas en tablas, tarjetas o gráficos.

La combinación de TypeScript para la lógica central y de JavaScript para la capa de presentación ha permitido construir una herramienta modular, interactiva y eficiente, alineada con las prácticas modernas de desarrollo web.

3.7. CSS

CSS (Cascading Style Sheets) es el lenguaje estándar para definir la presentación visual de documentos HTML [35]. En Code-XR se emplea para dar estilo a las plantillas HTML generadas en el modo LivePanel, donde se muestran métricas como líneas de código, complejidad ciclomática o número de parámetros. Su uso permite resaltar valores según la complejidad,

organizar la información en tarjetas y adaptar la visualización a distintos tamaños de ventana. Aunque no interviene en los modos XR inmersivos ni en la representación tridimensional del DOM, CSS cumple un papel clave en la experiencia de usuario de las visualizaciones 2D, manteniendo una arquitectura modular al separar el estilo de la lógica de análisis y comunicación.

3.8. Python y Lizard

Python es un lenguaje de alto nivel ampliamente utilizado en automatización y análisis de datos [31], y en Code-XR se emplea para realizar el análisis estático del código fuente. Desde el backend del plugin, escrito en TypeScript, se lanzan scripts en Python que procesan ficheros o directorios y generan un fichero JSON con las métricas calculadas.

Para obtener métricas como complejidad ciclomática (CCN), líneas de código (LOC) o número de funciones, se utiliza Lizard [33], una biblioteca de análisis estático compatible con múltiples lenguajes que devuelve una representación estructurada de los componentes del código. Estos datos alimentan tanto al modo LivePanel (visualización 2D) como al modo XR (ciudad 3D), donde se mapean propiedades como tamaño o complejidad a atributos visuales como altura, color o volumen.

Además, en el modo VisualizeDOM, Python se encarga de leer archivos HTML y transformarlos en una cadena que el componente `babia-html` interpreta para generar la visualización tridimensional. En conjunto, la integración de Python y Lizard proporciona datos estáticos fiables y estructurados, fundamentales para las visualizaciones de Code-XR en todos sus modos.

3.9. Gafas de Realidad Virtual: Meta Quest 3

Para validar las funcionalidades inmersivas de Code-XR se utilizaron gafas de realidad virtual Meta Quest 3 [17], cedidas temporalmente por la Universidad Rey Juan Carlos. Estos dispositivos, capaces de ejecutar experiencias VR de forma autónoma gracias a su procesador y sensores integrados, se conectaron al servidor local del plugin para acceder a las visualizaciones XR.

Su uso permitió comprobar en condiciones reales el funcionamiento del modo XR, ajustar la

navegación y la disposición de los gráficos, y evaluar el rendimiento general. Además, fueron fundamentales para verificar la integración con WebXR y la representación de métricas complejas en escenarios tridimensionales navegables.

3.10. Apoyo de herramientas de inteligencia artificial

Durante el desarrollo se emplearon herramientas de inteligencia artificial como apoyo para resolver dudas técnicas, explorar alternativas y aprender nuevas tecnologías.

ChatGPT [29] fue clave para comprender bibliotecas como Lizard (análisis estático de código) o A-Frame (escenas XR), así como para proponer soluciones a problemas concretos, mejorar la organización del código y redactar documentación técnica más clara.

Claude 4 [1] se utilizó principalmente en la gestión y organización de grandes volúmenes de código, especialmente útil en un repositorio con más de 200 ficheros en la carpeta `src/`.

Estas herramientas no sustituyen el conocimiento propio, pero sí aportaron valor al desarrollo al mejorar la eficiencia y la claridad en distintas partes del proyecto.

Capítulo 4

Diseño e implementación

4.1. Resumen cronológico de versiones

El desarrollo de Code-XR siguió un proceso iterativo e incremental a lo largo de seis meses, desde el prototipo inicial hasta la versión estable 1.0.0. La tabla 4.1 presenta un resumen cronológico de las versiones principales del plugin, condensando los hitos de desarrollo más relevantes y permitiendo observar la evolución progresiva de funcionalidades.

Complementariamente, la figura 4.1 ilustra mediante un diagrama temporal simplificado la secuencia de versiones y sus principales características, facilitando la comprensión visual del proceso de desarrollo seguido.

Nota: Detalles extraídos del fichero `CHANGELOG.md`. Registros públicos desde v0.0.3, marketplace desde v0.0.6.

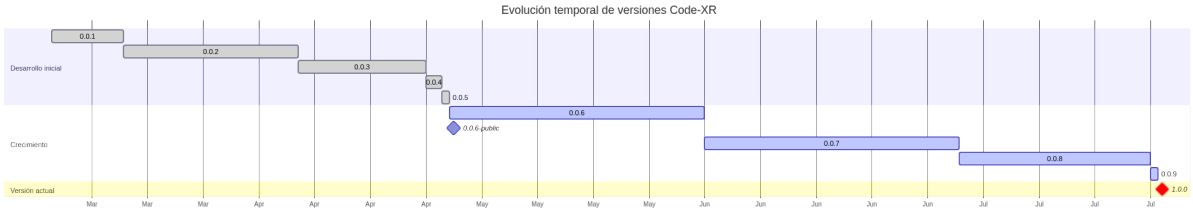


Figura 4.1: Evolución temporal resumida de las versiones principales del proyecto.

En los siguientes subapartados se describirán los sprints/hitos que explican cómo se pasó de la idea inicial a la versión 1.0.0, detallando para cada sprint el objetivo, las tareas realizadas y los artefactos resultantes.

Versión	Fecha	Hito principal
0.0.1	2025-03-11	Prototipo inicial: servidor HTTP con escena A-Frame básica y comandos de lanzamiento.
0.0.2	2025-03-20	Análisis básico LivePanel (JS/TS), infraestructura XR inicial, verificaciones HTTPS.
0.0.3	2025-04-11	Auto-reanalysis, SSE, watchers, análisis estático (LOC, CCN, funciones).
0.0.4	2025-04-27	Soporte multi-lenguaje, debounce configurable, análisis múltiple simultáneo.
0.0.5	2025-04-29	Visualizaciones babia-boats, mapeo parámetros-dimensiones visuales.
0.0.6	2025-04-29	Publicación inicial marketplace, estabilización servidores y UX.
0.0.7	2025-06-01	Live reload mejorado, controles VR/AR, configuración avanzada entornos.
0.0.8	2025-07-03	Motor análisis revisado: Lizard ampliado, VisualizeDOM, bubble charts.
0.0.9	2025-07-27	Re-arquitecturación: análisis directorios completos, soporte deep, optimizaciones.
1.0.0	2025-07-28	Versión estable: auto-análisis configurable, gestión sesiones, web oficial.

Tabla 4.1: Resumen cronológico de versiones principales del desarrollo de Code-XR.

4.2. Primeros pasos

Esta sección presenta las decisiones técnicas iniciales para crear la extensión de VS Code. Se siguió el tutorial oficial [22] y recursos complementarios [5], utilizando el generador "Yo Code" [40] para establecer las bases del proyecto.

4.2.1. Generador *Yo Code*

El generador oficial "Yo Code" configura la plantilla inicial mediante preguntas sobre identificador, descripción, lenguaje de desarrollo (TypeScript/JavaScript), repositorio Git y tipo de empaquetado (*unbundled*, Webpack, Esbuild). Estas decisiones configuran la estructura base y scripts de compilación.

4.2.2. Elección de lenguaje

Se eligió **TypeScript** por su tipado estático, integración directa con la API de VS Code y mejor mantenibilidad en proyectos complejos [20]. Permitió la evolución segura de Code-XR al crecer en ficheros y complejidad.

4.2.3. Bundlers

El empaquetado puede realizarse de distintas formas según las necesidades del proyecto [34]:

- **Unbundled**: sin bundler, prototipado rápido pero distribución manual de dependencias.
- **Webpack**: bundler establecido, genera paquete único y minimiza dependencias [37].
- **Esbuild**: bundler moderno con compilación extremadamente rápida [11].

Code-XR comenzó con *unbundled* para prototipado rápido, migrando posteriormente a **Webpack** para mayor fiabilidad y resolver advertencias de `npm` por tamaño excesivo en el empaquetado `.vsix`.

4.2.4. Archivo `package.json` (manifiesto de la extensión)

El `package.json` es el manifiesto central con metadatos del Marketplace, compatibilidad con VS Code y configuración de compilación. La sección `contributes` define los elementos de interfaz: +40 comandos, vista `codexrTree` y menús contextuales [34].

Code-XR utiliza activación ansiosa (lista vacía en `activationEvents`) para garantizar disponibilidad inmediata de menús y vistas. La iconografía combina *Codicons* [23] oficiales, iconos *Devicon* [9] para representación de ficheros, y logotipo personalizado generado con IA.

4.2.5. Archivo `extension.ts`

Implementa la lógica del manifiesto como punto de entrada con dos funciones principales:

- `activate`: inicializa servicios, registra vistas y comandos, sincroniza estado.
- `deactivate`: libera recursos (servidores, gestores SSE, mapeos).

Garantiza arranque limpio y cierre ordenado sin procesos residuales.

4.2.6. Registro de comandos e interfaz de usuario

El sistema de comandos de Code-XR establece la conexión entre las acciones del usuario y la lógica interna del plugin mediante un proceso de dos fases: declaración en el manifiesto `pa`

`ckage.json` y enlace programático en `extension.ts`. La figura 4.2 ilustra la arquitectura general de estos componentes y sus interrelaciones dentro del ecosistema de VS Code.

La interfaz de usuario se construye utilizando las primitivas nativas proporcionadas por la API de VS Code, garantizando integración consistente con el entorno del editor:

- **TreeView / TreeDataProvider**: estructuras jerárquicas para organización en vista lateral, utilizadas para mostrar servidores activos, ejemplos de BabiaXR y configuraciones del proyecto.
- **WebviewPanel / WebviewView**: paneles web integrados que permiten visualizaciones 2D/XR interactivas, manteniendo comunicación bidireccional con la extensión.
- **StatusBarItem**: elementos de la barra de estado para notificaciones en tiempo real y accesos directos a funcionalidades principales.

Como se observa en la figura 4.2, estos componentes forman un ecosistema cohesivo donde el archivo `package.json` define la estructura declarativa, el módulo `extension.ts` orquesta la lógica operacional, y las primitivas de UI proporcionan la interfaz de usuario final.

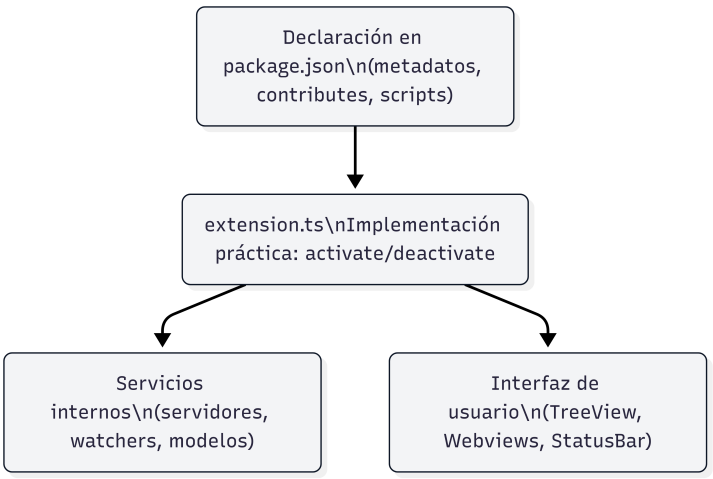


Figura 4.2: Relación entre los componentes clave de una extensión VS Code en Code-XR.

4.3. Sprint 1 — Lanzamiento de servidores locales

Este sprint estableció la capacidad fundamental de Code-XR para lanzar servidores locales, permitiendo servir visualizaciones XR desde el propio editor. La solución se inspiró en

extensiones como *Live Server* [10], eliminando la necesidad de configurar servidores externos manualmente.

El subsistema de servidores proporciona **arquitectura modular**, **persistencia de preferencias**, **multi-servidor concurrente**, **resolución automática de puertos**, **soporte HTTP/HTTPS completo** y **gestión centralizada de estado**. El diseño sigue patrones escalables con separación de responsabilidades y gestión automatizada del ciclo de vida [15].

4.3.1. Arquitectura modular del subsistema

El código se estructura en capas especializadas bajo `src/servers/` y `src/active_servers/`, implementando una arquitectura de responsabilidades distribuidas:

- **runtime**: Núcleo de ejecución con servidores HTTP/HTTPS, gestión de puertos y subsistema SSE.
- **storage**: Persistencia con validación atómica y migraciones de esquema.
- **registry**: Estado centralizado con eventos síncronos para UI.
- **services**: Validación de registros y control de WebviewPanels.
- **views**: Integración con TreeView de VS Code.

4.3.2. Configuración y persistencia

El `ServerSettingsManager` gestiona preferencias mediante `server-settings.json` en `globalStorage`, garantizando persistencia cross-workspace. El panel de configuración (Figura 4.3) ofrece controles granulares: selección de protocolo HTTP/HTTPS con validación automática de certificados, configuración de puerto por defecto con detección de colisiones, apertura automática configurable y selección entre modo navegador o panel lateral. Los botones de acción directa permiten lanzar servidores con la configuración activa o restaurar valores por defecto.

```
{
  "mode": "HTTPS",
  "https": { "certSource": "default",
             "certPath": "certs/babia_cert.pem",
             "keyPath":  "certs/babia_key.pem" },
  "defaultPort": 3000,
  "launch": { "autoOpen": true, "openMode": "browser" },
  "configNonce": "37a3047821ba23de99348bcba2d155c3",
  "version": "1.0.0"
}
```

Listing 4.1: Configuración de servidores en Code-XR.

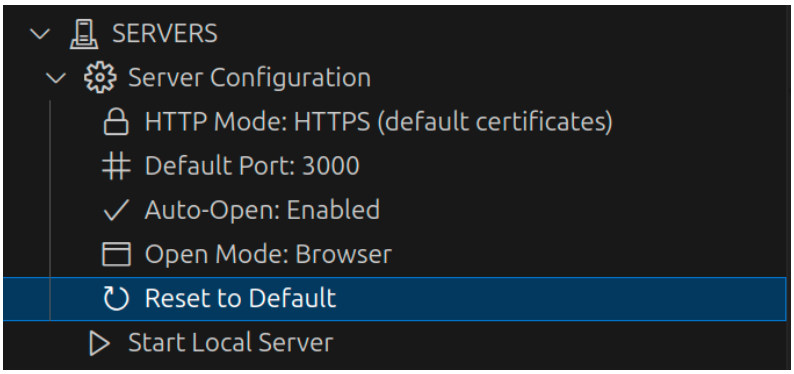


Figura 4.3: Panel de configuración avanzada con controles HTTP/HTTPS, puertos y modo de lanzamiento.

4.3.3. Gestión inteligente de puertos y HTTP/HTTPS

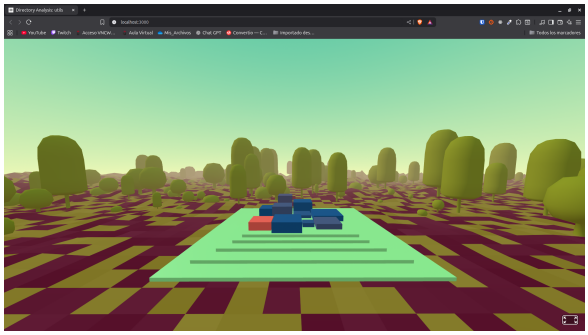
El `PortManager` implementa descubrimiento automático mediante `get-port` [32], realizando búsqueda secuencial desde el puerto configurado hasta encontrar uno disponible. Ante colisiones, notifica de manera transparente la reasignación. Se soporta HTTP y HTTPS (obligatorio para XR); en caso de intentar usar HTTPS con panel lateral, el plugin avisa y fuerza el uso de HTTP por limitaciones de VS Code.

El soporte HTTPS ofrece dos modalidades:

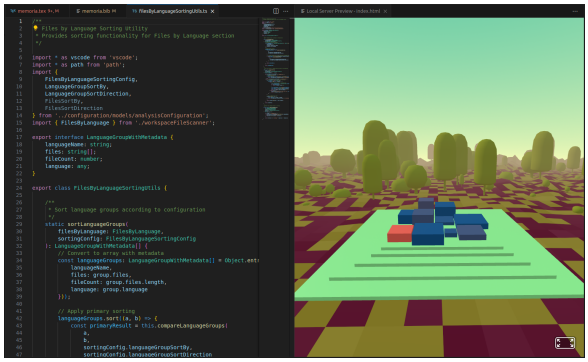
- **Certificados por defecto**: Par `cert/key` incluido, generados con `OpenSSL` [30].
- **Certificados personalizados**: Selector con validación de compatibilidad.

4.3.4. Modos de visualización

Code-XR ofrece dos modos de visualización de los servidores lanzados, configurables desde el panel de settings. En la figura 4.4 se comparan ambos modos, en la izquierda el modo navegador (pestaña externa) y en la derecha el modo panel lateral (integrado en VS Code).



(a) Modo navegador: visualización completa en navegador web externo



(b) Modo panel lateral: integración directa en VS Code

Figura 4.4: Comparación entre los dos modos de visualización: navegador web (izquierda) y panel lateral integrado (derecha).

Nota: Ambas imágenes muestran el mismo análisis XR del directorio `src/utils` de Code-XR.

4.3.5. Active Servers y observabilidad

La vista *Active Servers* constituye el sistema de observabilidad y control del subsistema de servidores, exponiendo un registro centralizado que rastrea instancias activas con estado completo, metadatos contextuales y acciones específicas. El registro mantiene naturaleza efímera para prevenir servidores zombie y garantizar coherencia entre sesiones de VS Code.

Como se aprecia en la figura 4.5, esta vista presenta un listado estructurado en tiempo real donde cada servidor activo se identifica mediante iconografía específica: un escudo con llave para instancias HTTPS y un globo terráqueo para servidores HTTP. El nombre descriptivo del servidor incluye información contextual relevante: en caso de análisis de código se especifica el fichero o directorio analizado, mientras que para otros tipos se indica la funcionalidad correspondiente.

Adicionalmente, cada entrada muestra el modo de visualización con el que se lanzó el

servidor mediante indicadores que distinguen entre lanzamiento en navegador web externo o visualización mediante panel lateral integrado en VS Code. Las acciones disponibles mediante click izquierdo sobre cada servidor incluyen: apertura directa en navegador o panel lateral, copia de URL al portapapeles, visualización de información detallada del servidor y cierre individual del servidor seleccionado. Para la gestión masiva, el sistema proporciona la opción *Stop All Servers* que permite cerrar todas las instancias activas simultáneamente.

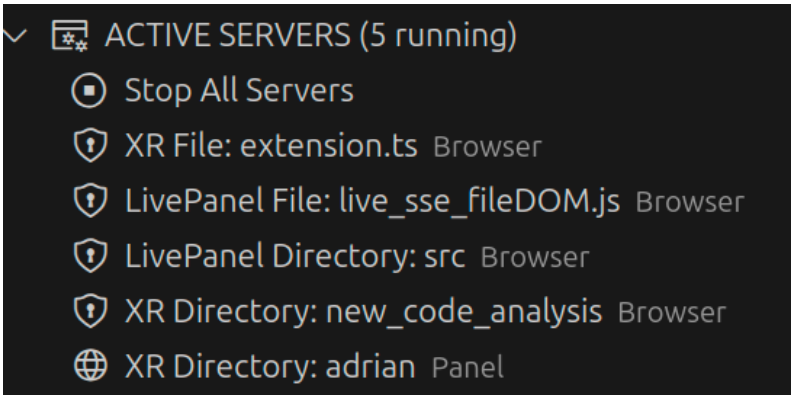


Figura 4.5: Vista *Active Servers* con registro centralizado y acciones contextuales.

4.3.6. Comunicación en tiempo real (SSE)

Los servidores exponen endpoints `/events` con *Server-Sent Events* para propagación automática de cambios de análisis, gestión de clientes y mapeo archivo-servidor. El sistema actúa como bus de comunicación entre extensión y visualizaciones web.

4.3.7. Compatibilidad VR

`NetworkUtils` detecta automáticamente la IP local y genera URLs de acceso externo, facilitando uso desde dispositivos VR en la misma red mediante bind a `0.0.0.0`.

4.4. Sprint 2 — Integración con BabiaXR y ejemplos de visualización

Este sprint estableció las capacidades de visualización XR completas de Code-XR mediante la integración con **BabiaXR**, motor de visualización 3D para todas las representaciones

WebXR/AR/VR del plugin. El desarrollo siguió tres fases consecutivas para dominar la integración entre **BabiaXR**, **datos JSON** y el **plugin VS Code**, preparando el Sprint 3 de análisis en tiempo real.

Fase 1: Comprensión mediante ejemplos — Galería de ejemplos predefinidos basados en BabiaXR [25] para familiarización con tipos de gráficos, configuración y estructuras de datos.

Fase 2: Pipeline JSON→XR — Transformación automática de ficheros JSON en visualizaciones XR mediante análisis de estructura, mapeo de dimensiones y generación de plantillas HTML/A-Frame.

Fase 3: Personalización visual — Control granular de aspectos estéticos: ambientes XR, colores de fondo/suelo y paletas cromáticas aplicables tanto a datos JSON como análisis de código.

Módulos resultantes

1. **Babia Examples**: galería interactiva de ejemplos BabiaXR con lanzamiento directo y gestión centralizada.

2. **Visualize Data**: pipeline JSON→XR con análisis automático, mapeo inteligente y validación contextual.

3. **Visualization Settings**: configuración visual global para entornos, colores y paletas aplicada a todas las visualizaciones.

Todos reutilizan la infraestructura de servidores del Sprint 1, garantizando coherencia en comportamiento, configuración y experiencia de usuario.

4.4.1. Babia Examples

El subsistema *Babia Examples* proporciona una galería interactiva de ejemplos predefinidos BabiaXR dentro de VS Code, facilitando el acceso inmediato a visualizaciones WebXR/AR/VR mediante un flujo de un clic. Permite explorar capacidades de BabiaXR, experimentar con diferentes gráficos y usar ejemplos como plantillas de referencia [25].

Reutiliza la infraestructura de servidores del Sprint 4.3, lanzando ejemplos según la configuración activa y registrándolos en *Active Servers* para gestión centralizada.

Arquitectura y descubrimiento Implementa arquitectura modular bajo `src/babia_examples/` con capas especializadas: `model/`, `runtime/`, `views/` y `commands/`. El `ExampleLauncher` busca automáticamente ejemplos disponibles recorriendo `examples/charts/` y detectando archivos HTML válidos.

Catálogo de ejemplos La colección incluye gráficos diversos que ilustran técnicas BabiaXR:

- **Bar Chart**: gráfico de barras clásico en dos ejes.
- **Barsmap**: variante espacial con tres ejes.
- **Bubble Chart**: dispersión 3D con tamaño como cuarto eje.
- **Cylinder Chart**: bar chart con cilindros tridimensionales.
- **Cylindermap Chart**: barsmap con cilindros.
- **Pie**: gráfico circular clásico.
- **Mix**: escenario combinando cuatro tipos de gráficos.

Como puede observarse en la figura 4.6, la interfaz de *Babia Examples* presenta un listado visual de cada ejemplo, su nombre descriptivo y una breve descripción. Cada entrada es interactiva, permitiendo lanzar el ejemplo correspondiente con un solo clic izquierdo.

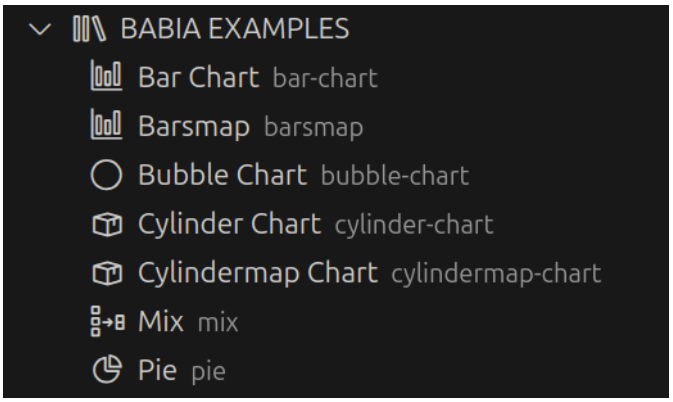


Figura 4.6: Vista del apartado *Babia Examples* mostrando la galería de ejemplos BabiaXR disponibles, cada uno desplegable con un clic y con integración completa en el ecosistema de servidores de Code-XR.

Integración ecosistémica Sigue principios de delegación inteligente, aprovechando la infraestructura existente sin duplicar funcionalidad. Delega al `MultiServerLauncher` para garantizar comportamiento consistente en puertos, certificados SSL y registro en *Active Servers*. Los ejemplos aparecen con nombres descriptivos y heredan preferencias de configuración.

Estructura de archivos El directorio `examples/` separa recursos y datos:

- `charts/`: estructura categórica con ejemplos HTML completos.
- `data/`: datasets centralizados (`gdp.json`, `population.json`, etc.) para demostraciones.

Esta organización facilita reutilización, mantenimiento y escalabilidad para nuevos ejemplos.

4.4.2. Visualize Data — del JSON a una visualización XR

El subsistema *Visualize Data* constituye el núcleo de transformación de datos, proporcionando un pipeline que convierte ficheros `JSON` arbitrarios en visualizaciones inmersivas BabiaXR mediante un flujo guiado de cuatro etapas: selección de gráfico, análisis JSON, mapeo de dimensiones y lanzamiento. Se integra de forma transparente con la infraestructura del Sprint 4.3 y hereda automáticamente preferencias de *Visualization Settings*.

Arquitectura modular Implementa arquitectura distribuida bajo `src/visualize_data/` con capas especializadas: `state/`, `model/`, `runtime/`, `commands/` y `views/`. El estado se persiste a nivel workspace, garantizando configuraciones persistentes entre sesiones.

Catálogo de gráficos Incluye tipos BabiaXR: *Bar*, *Pie*, *Donut*, *Bubbles*, *Cylinders*, *Barsmap*, *Cylindermap*, *Mix* y *Boats*. Cada tipo define dimensiones específicas y plantilla HTML/BabiaXR asociada, estableciendo las bases para el análisis de código. En la figura 4.7 se muestra el selector de gráficos disponibles, ilustrando la diversidad de visualizaciones soportadas.

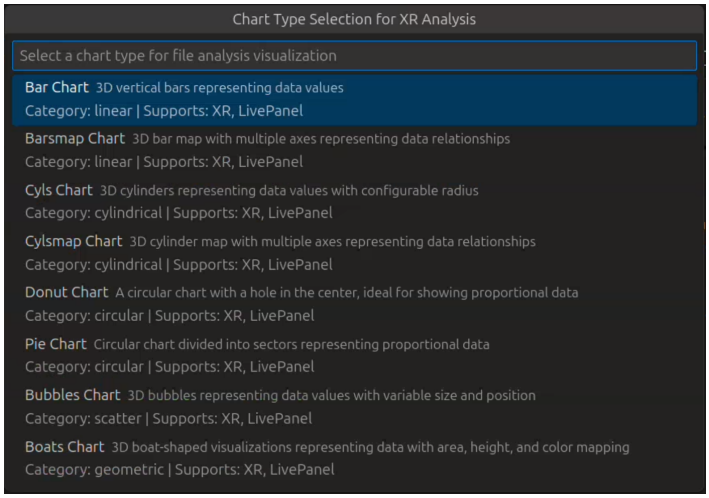


Figura 4.7: Selector de gráficos disponibles en *Visualize Data*, mostrando la diversidad de tipos de visualización BabiaXR soportados.

Análisis inteligente JSON El analizador identifica estructura y clasifica valores entre numéricos y no numéricos (*string*, *boolean*, *object*, *array*), considerando que ciertos ejes BabiaXR requieren datos numéricos exclusivamente. Genera resumen estructurado con entradas totales, tipos de campo y ejemplos de valores.

Mapeo de dimensiones contextual Permite asignar campos JSON a dimensiones gráficas con validación inteligente: dimensiones numéricas (`size`, `height`) filtran automáticamente campos disponibles. Proporciona retroalimentación en tiempo real, validación de coherencia y prevención de configuraciones incompletas. En la figura 4.8 se observa la interfaz de mapeo de dimensiones, mostrando la asignación de campos JSON a dimensiones del gráfico con filtrado por tipo de dato.

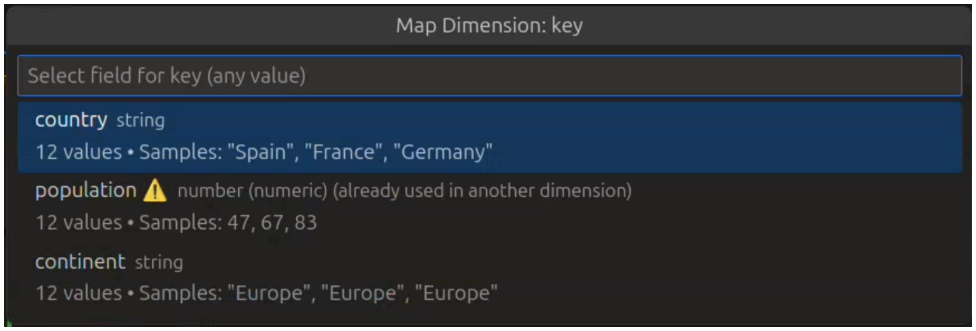


Figura 4.8: Interfaz de mapeo de dimensiones con validación contextual, mostrando la asignación de campos JSON a dimensiones del gráfico con filtrado por tipo de dato.

Generación y persistencia Combina plantilla HTML del gráfico con datos mapeados y preferencias globales de *Visualization Settings*, inyectando automáticamente paleta cromática, ambiente XR y configuración de suelo. Almacena visualizaciones en `globalStorage` con directorios únicos (`Identificador_{nonce}`) conteniendo `index.html` y `data.json`.

Browse Visualizations Gestiona biblioteca personal de visualizaciones: escanea `globalStorage`, valida integridad de archivos y ofrece relanzamiento individual o limpieza masiva (*Reset All*). Transforma el subsistema en herramienta reutilizable con integración completa al sistema de servidores.

Interfaz Visualize Data La figura 4.9 muestra la interfaz completa de *Visualize Data*, ilustrando el flujo de trabajo paso a paso desde la selección de gráfico, de fichero JSON, mapeo de dimensiones y lanzamiento, hasta la gestión de la biblioteca de visualizaciones almacenadas. Cada etapa está claramente diferenciada, proporcionando una experiencia de usuario intuitiva y guiada.

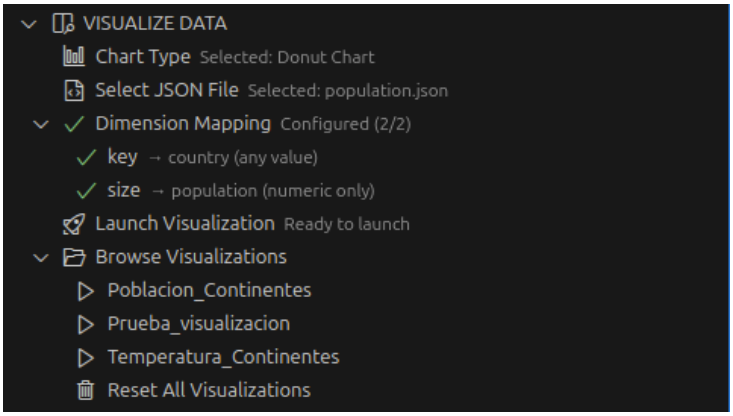


Figura 4.9: Interfaz completa de *Visualize Data* mostrando el flujo de trabajo paso a paso y la gestión de biblioteca de visualizaciones almacenadas.

Integración ecosistémica Delega lanzamiento al `MultiServerLauncher` siguiendo la arquitectura del Sprint 4.3, garantizando consistencia en HTTP/HTTPS, puertos automáticos y registro en *Active Servers*. La herencia automática desde *Visualization Settings* aplica configuración estética consistente sin intervención del usuario.

4.4.3. Visualization Settings

El apartado *Visualization Settings* constituye el sistema de configuración visual global, permitiendo personalizar cuatro aspectos fundamentales aplicados automáticamente a todas las visualizaciones: **color de fondo**, **color del suelo**, **preset del entorno** y **paleta de gráficos**.

Arquitectura y persistencia Sigue la arquitectura modular establecida bajo `src/visualization_settings/` con separación entre `model/`, `storage/`, `commands/`, `views/` y `utils/`.

Implementa persistencia dual: `visualization-settings.json` en `globalStorage` (primario) y `globalState` (respaldo), garantizando configuraciones cross-workspace con migración automática.

```json
{
  "backgroundColor": "#FFFFFF",
  "groundColor": "#000000",
  "environment": "forest",
  "palette": "ubuntu"
}
```

Listing 4.2: Estructura del archivo `visualization-settings.json` que persiste las preferencias visuales globales.

Interfaz visual integrada Como se muestra en la figura 4.10, el apartado *Visualization Settings* se presenta como colapsable en el árbol principal con iconografía contextual: *Background/Ground Color* muestran iconos SVG dinámicos de 16×16 píxeles reflejando el color seleccionado, mientras *Environment Preset* y *Chart Palette* muestran texto de la opción activa.

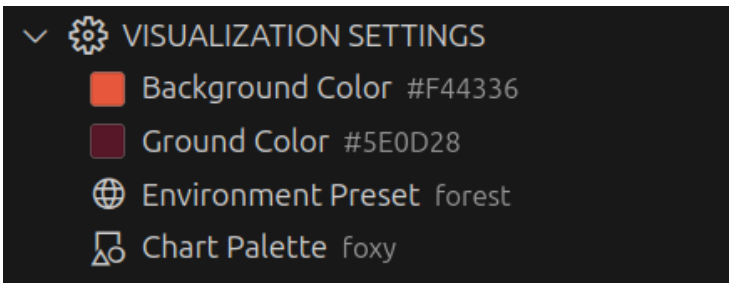


Figura 4.10: Vista del apartado *Visualization Settings* integrado en el árbol principal, mostrando iconos dinámicos de color para los parámetros visuales.

Configuración de colores Implementa flujo híbrido con webview personalizado integrando selector HTML5 nativo, proporcionando preview en tiempo real y validación automática hexadecimal (`#RRGGBB`). Se puede observar en la figura 4.11 el panel de selección de colores, mostrando el selector HTML5 con el color actual seleccionado y controles para aplicar o cancelar cambios.

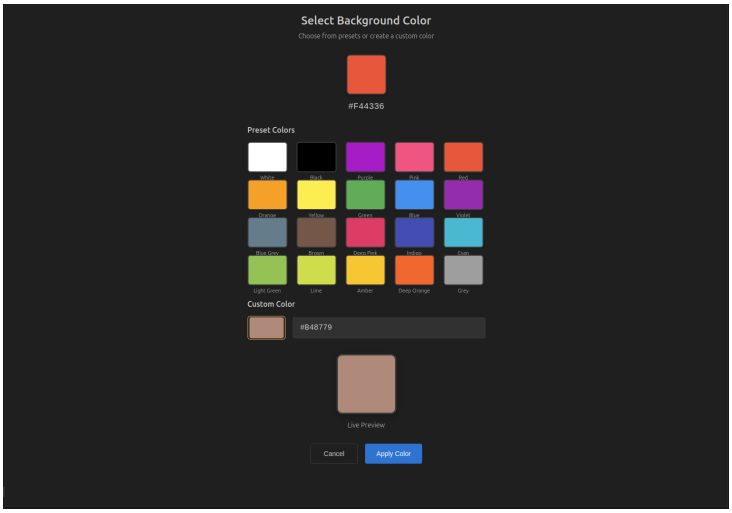
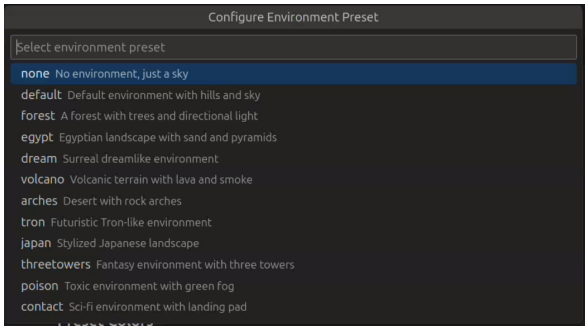
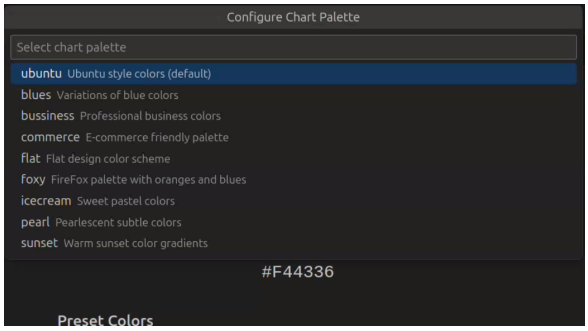


Figura 4.11: Panel de selección de colores mediante webview integrado, utilizando el selector HTML5 nativo con validación en tiempo real.

Entornos y paletas Utiliza interfaz *QuickPick* para selección contextual. Entornos: 12 presets A-Frame (`forest`, `tron`, `egypt`, `dream`, `volcano`, `arches`, etc.) redefiniendo ambientación XR completa. Paletas: 9 esquemas cromáticos BabiaXR (`ubuntu`, `blues`, `sunset`, `foxy`, `icecream`, etc.) optimizados por contexto. Se puede observar en la figura 4.12 los selectores *QuickPick* para ambos parámetros, mostrando la lista de opciones disponibles con descripciones contextuales e iconografía visual.



(a) Selector *QuickPick* para *Environment Preset*



(b) Selector *QuickPick* para *Chart Palette*

Figura 4.12: Selectores *QuickPick* para configuración visual: entornos A-Frame disponibles con descripciones contextuales (izquierda) y paletas de colores BabiaXR con vistas previas (derecha).

Aplicación automática Integración transparente con motor de plantillas: `getVisualizationConfiguration()` recupera preferencias e inyecta en plantillas HTML/A-Frame. Mapeo

directo:

- **Background Color** → atributo `background="color:#FFFFFF"` de la escena A-Frame
- **Environment Preset** → componente `environment="preset:forest"`
- **Ground Color** → parámetro `groundColor:#000000` del entorno
- **Chart Palette** → atributo `palette:ubuntu` en componentes BabiaXR

Iconografía dinámica Generación automática de iconos SVG como indicadores visuales: cuadrados 16×16 píxeles para colores, efectos de borde para legibilidad en temas VS Code, nomenclatura `\{settingKey\}_\{hexcolor\}.svg` con caché eficiente y limpieza automática de obsoletos.

4.4.4. Integración de fases y transición al Sprint 3

Esta aproximación incremental estableció una base arquitectónica sólida y extensible que permitió avanzar hacia el analizador y visualizador de análisis en tiempo real del siguiente sprint. Cada fase aportó componentes modulares reutilizables, garantizando que Code-XR cumpliera objetivos funcionales con experiencia de usuario fluida.

4.5. Sprint 3 — Análisis y visualización de métricas de código

4.5.1. Introducción al análisis de código

El subsistema de análisis de código constituye el núcleo de Code-XR, transformando código fuente en visualizaciones interactivas 2D y XR. Integra los componentes desarrollados en sprints anteriores para ofrecer comprensión intuitiva de la estructura y complejidad del software.

Contexto y evolución desde sprints anteriores Aprovecha componentes clave de sprints anteriores:

- **Sprint 1 — Infraestructura de servidores**: arquitectura modular HTTP/HTTPS para alojar visualizaciones dinámicas y gestionar múltiples sesiones.

- **Sprint 2 — Visualización con BabiaXR**: plantillas BabiaXR y sistema de configuración para visualizaciones personalizadas.

Capacidades principales del subsistema Dos ejes fundamentales: *alcance* y *modo de visualización*.

1. **Alcance del análisis**:
   - **Análisis de fichero individual**: métricas detalladas (complejidad, distribución código/comentarios, estructura de clases).
   - **Análisis de directorio**: agregación multi-archivo, distribución de lenguajes y complejidad global.
   - **Análisis de proyecto**: visión holística incluyendo dependencias y estructura organizativa.
2. **Modos de visualización**:
   - **LivePanel**: visualización 2D integrada en VS Code mediante webviews.
   - **XR Mode**: representación tridimensional inmersiva para entornos VR/AR.
   - **DOM Mode**: visualización especializada de estructura DOM como árbol 3D interactivo.

Unión de alcance y modos de visualización Como se puede ver en la figura 4.13, el subsistema de análisis se define por la combinación de estas dos dimensiones: el alcance determina el contexto y profundidad del análisis, mientras que el modo de visualización define cómo se presenta esta información al usuario (Importante no confundir con la forma en la que se visualiza el servidor, recuerda figura 4.4). Esta matriz de posibilidades permite a los usuarios adaptar el análisis a sus necesidades específicas, ya sea para inspección detallada de un archivo o comprensión global de un proyecto completo, utilizando la representación visual que mejor se adapte a su flujo de trabajo.

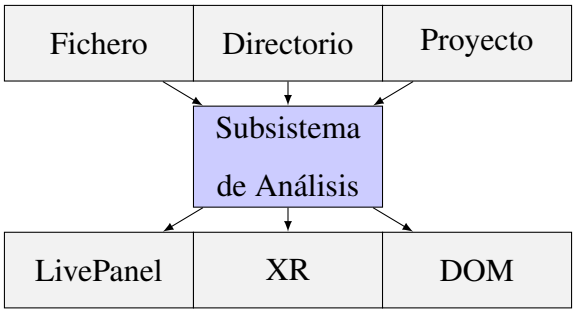


Figura 4.13: Dimensiones fundamentales del subsistema de análisis: alcance del análisis (entrada) y modos de visualización (salida).

Objetivos técnicos y de experiencia de usuario Objetivos clave del subsistema:

- **Análisis en tiempo real**: retroalimentación instantánea sobre cambios en el código.
- **Representación intuitiva**: transformar conceptos abstractos en metáforas visuales comprensibles.
- **Escalabilidad**: rendimiento óptimo con proyectos grandes mediante análisis incremental.
- **Integración fluida**: fusión con el flujo natural de trabajo en VS Code.
- **Personalización**: adaptación según preferencias específicas del usuario.
- **Extensibilidad**: facilitar adición de nuevos analizadores y visualizaciones.

En las siguientes secciones se detalla la arquitectura técnica, el flujo de datos, los motores de análisis y los mecanismos de visualización que hacen posible esta funcionalidad.

4.5.2. Arquitectura del subsistema de análisis

El subsistema implementa una arquitectura modular combinando patrones de diseño para flexibilidad, mantenibilidad y rendimiento en análisis múltiples, sesiones simultáneas y actualizaciones en tiempo real.

Patrones de diseño aplicados Patrones clave para organización y extensibilidad:

- **Patrón "Nested Dolls"**: registro de comandos donde cada subsistema contribuye sin conocimiento de otros componentes (principio Matryoshka).

- **Patrón Registry**: `UnifiedSessionRegistry` como punto central para gestión del ciclo de vida de sesiones.
- **Patrón Observer**: sistema de *watchers* que monitoriza cambios y emite eventos SSE para actualizaciones en tiempo real.
- **Patrón Factory**: fábricas especializadas como `NewCodeAnalysisItemFactory` para generar implementaciones concretas.

Componentes principales La arquitectura está organizada en componentes especializados con responsabilidades bien definidas. Como se muestra en la figura 4.14, el sistema se estructura en ocho módulos principales que trabajan de forma coordinada:

1. **Analysis Orchestrator**: coordina proceso completo desde solicitud hasta visualización.
2. **Session Registry**: registro centralizado de sesiones activas con estado y configuración.
3. **Python Engine**: extrae métricas del código fuente mediante bibliotecas especializadas.
4. **File Requirement Processor**: prepara archivos necesarios (plantillas HTML, scripts JS/CSS, datos).
5. **Template Engine**: combina plantillas con datos de análisis y configuraciones visuales.
6. **Session Watcher Manager**: gestiona observadores de archivos para re-análisis automáticos.
7. **Session Server Manager**: controla servidores HTTP y canales SSE.
8. **Configuration Storage**: almacena configuraciones, preferencias y perfiles de usuario.

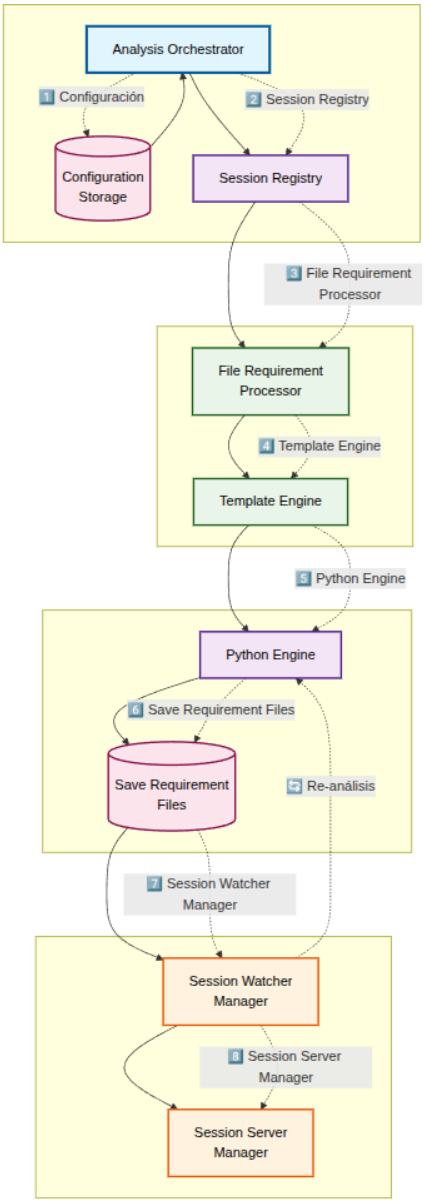


Figura 4.14: Arquitectura del subsistema de análisis mostrando los componentes principales y sus interacciones

Integración con VS Code y Code-XR El subsistema se integra con VS Code a través de varios puntos de extensión:

- **Comandos**: Registra comandos para iniciar análisis desde la paleta de comandos, menús contextuales y la interfaz del plugin.
- **TreeView**: Proporciona vistas de árbol personalizadas para explorar sesiones activas, resultados de análisis y configuraciones disponibles.

- **WebViews**: Utiliza WebViews de VS Code para visualizaciones LivePanel integradas directamente en el editor.
- **File System Watcher API**: Aprovecha la API de observación de archivos para detectar cambios y activar re-análisis automáticos.

Reutilización de componentes Code-XR:

- **MultiServerLauncher**: gestión de servidores HTTP/HTTPS para visualizaciones.
- **Visualization Settings**: configuraciones estéticas (temas, colores, paletas).
- **Storage Utilities**: persistencia de datos, configuraciones y resultados.

Esta integración garantiza comportamiento coherente y aprovechamiento completo de la infraestructura existente.

4.5.3. Flujo de datos / Secuencia de ejecución

El análisis en Code-XR sigue una secuencia básica de siete etapas bien definidas, desde la petición inicial del usuario hasta la actualización automática de la visualización. La figura 4.15 ilustra esquemáticamente este proceso, mostrando la progresión lineal de cada fase y las dependencias entre los componentes del sistema:

1. **Petición del análisis.** Usuario solicita análisis mediante interfaz gráfica (clic derecho), menú rápido del plugin (*Files by Language*/*Project Structure*) o paleta de comandos (F1). Especifica tipo de análisis y objetivo (ruta al fichero/directorio).
2. **Registro de la sesión.** Se crea sesión con identificador único (nonce) registrando: ruta `workspaceStorage` [19], ficheros involucrados, tipo de análisis, estado y configuración.
3. **Obtención de ficheros.** Motor determina ficheros necesarios: plantillas HTML/CSS/JS para vista y argumentos para `main.py` que genera `data.json` con métricas. Se prepara paquete completo en `workspaceStorage`.
4. **Persistencia temporal.** Paquete se guarda en ruta de sesión facilitando re-análisis y arranque del servidor. Ficheros se eliminan al cerrar VS Code evitando *analysis zombies*.

5. **Activación de watchers.** Se registran observadores con *debounce time* configurable para agrupar cambios rápidos. Sistema espera intervalo sin cambios antes de ejecutar análisis, evitando sobrecarga.

6. **Lanzamiento servidor y SSE.** Servidor local sirve ficheros como `localhost` (HTTP/HTTPS según configuración) exponiendo canal Server-Sent Events para notificar actualizaciones en tiempo real.

7. **Re-análisis incremental.** Cuando watcher detecta cambio: identifica sesión asociada, re-analiza ficheros afectados, actualiza JSON/metadata, persiste resultados y emite evento SSE (`analysis-updated`) para actualización automática de vistas.

   a) Persistir los resultados en la ruta de la sesión para que el servidor los sirva inmediatamente.

   b) Emitir un evento por el canal SSE (`analysis-updated`); las vistas suscritas reciben la notificación y actualizan la visualización en tiempo real.

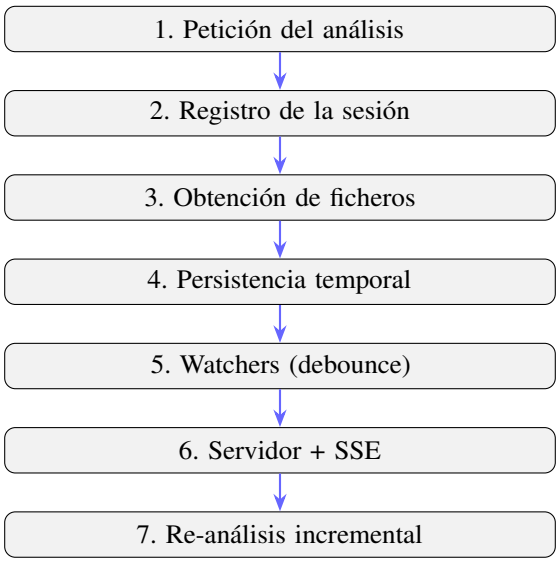


Figura 4.15: Secuencia vertical simplificada del flujo de un análisis.

El sistema gestiona múltiples sesiones simultáneas mediante registro centralizado, asegurando que los cambios se propagan correctamente a todas las vistas afectadas sin conflictos entre análisis concurrentes.

4.5.4. Motor de análisis en Python

Code-XR integra un motor Python que extrae métricas del código fuente, aprovechando las bibliotecas de análisis estático de Python dentro de la arquitectura TypeScript de VS Code.

Arquitectura de integración TypeScript ↔ Python La comunicación utiliza procesos hijo (`child_process`) y protocolo JSON:

1. **Inicialización**: TypeScript determina parámetros de análisis
2. **Ejecución Python**: Invocación de `main.py` como proceso hijo
3. **Transmisión**: Resultados JSON con marcadores `===JSON_START/END===`
4. **Procesamiento**: `ExecutePython` extrae y transforma los datos

Gestión del entorno Python `VenvManager` gestiona automáticamente entornos virtuales Python, detectando instalaciones, creando entornos aislados, instalando dependencias (Lizard) y manteniendo persistencia entre sesiones.

Lenguajes de programación soportados El motor de análisis utiliza la biblioteca Lizard [33] como analizador principal, proporcionando soporte nativo para una amplia gama de lenguajes de programación modernos:

- **Lenguajes web**: JavaScript, TypeScript, PHP, Vue
- **Lenguajes compilados**: C, C++, C#, Java, Go, Swift, Kotlin, Rust, Zig
- **Lenguajes dinámicos**: Python, Ruby, Lua, Perl
- **Lenguajes especializados**: Objective-C, Scala, Erlang, Fortran, GDScript, Solidity, TTCN-3

Caso especial: Los archivos HTML utilizan el analizador DOM especializado que extrae la estructura jerárquica del Document Object Model, proporcionando métricas específicas de complejidad estructural y anidamiento de elementos.

Esta amplia compatibilidad permite analizar proyectos políglotas y ofrece métricas consistentes independientemente del stack tecnológico utilizado.

Pipeline de análisis y analizadores especializados Pipeline modular con analizadores especializados:

1. **Analizador Lizard**: Métricas de complejidad ciclomática (CCN), densidad, profundidad de anidamiento y funciones de alta/crítica complejidad.
2. **Contador de clases**: Detecta clases, ubicaciones, nombres, métodos y relaciones de herencia.
3. **Analizador de comentarios**: Cuenta el número de comentarios de cada fichero.

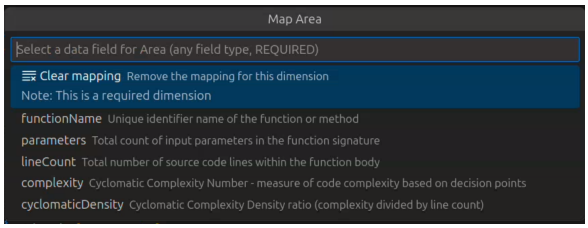
Estructura del `data.json` resultante El motor genera un documento JSON estructurado que contiene métricas y metadatos necesarios para la visualización. La estructura específica varía según el tipo de análisis (archivo individual vs. directorio) y el alcance del análisis, adaptándose dinámicamente a las características del código analizado. Este formato actúa como puente canónico entre el análisis técnico y la visualización, diseñado para ser extensible.

Las métricas disponibles para visualización dependen del alcance del análisis y permiten diferentes grados de personalización según el modo de visualización:

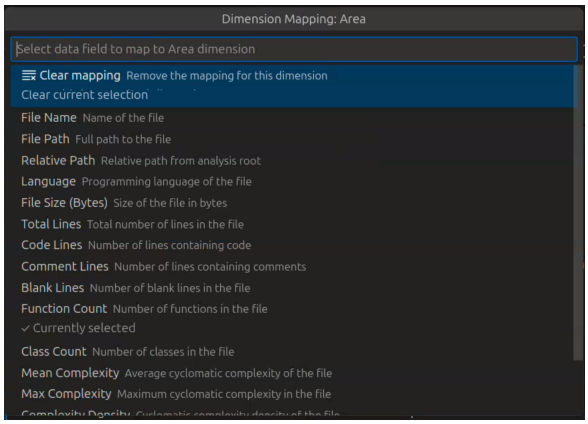
- **Modo XR**: Permite selección específica de métricas a representar mediante interfaz de mapeo de dimensiones, ofreciendo control granular sobre qué datos visualizar y cómo mapearlos a propiedades visuales (altura, tamaño, color...). Las métricas específicas disponibles varían según el alcance:
  - **Análisis de fichero**: nombre de función, parámetros, número de líneas, complejidad, densidad ciclomática.
  - **Análisis de directorio**: nombre de archivo, ruta del archivo, ruta relativa, lenguaje, tamaño del archivo (bytes), líneas totales, líneas de código, líneas de comentario, líneas en blanco, número de funciones, número de clases, complejidad media, complejidad máxima, densidad de complejidad, máximo de parámetros por función, promedio de parámetros por función.
- **Modo LivePanel**: Representa automáticamente todas las métricas disponibles en formato tabular integrado, sin posibilidad de selección específica de datos.

- **Modo DOM**: Visualiza la estructura jerárquica del DOM, no es posible configurar qué métricas visualizar.

En la figura 4.16 se muestran las interfaces de selección de métricas específicas en modo XR, ilustrando las diferentes opciones disponibles según el alcance del análisis (fichero vs. directorio). La imagen de directorios muestra una buena muestra del total de métricas disponibles, aunque no se visualizan todas debido a limitaciones de espacio en la interfaz.



(a) Métricas disponibles para análisis XR de ficheros individuales



(b) Métricas disponibles para análisis XR de directorios (muestra representativa)

Figura 4.16: Interfaces que permiten la selección de métricas específicas en modo XR, en función del alcance del análisis (fichero frente a directorio).

Optimización y análisis incremental Estrategias de optimización para rendimiento ágil:

- **Análisis incremental**: Re-analiza solo archivos modificados
- **Paralelización**: Distribuye análisis entre múltiples procesos Python
- **Caching**: Almacena resultados intermedios para acelerar análisis futuros
- **Análisis progresivo**: Retorna métricas esenciales rápidamente, completa detalles en segundo plano

4.5.5. Gestión de sesiones de análisis

La gestión de sesiones constituye el mecanismo central para coordinar análisis simultáneos, persistir configuraciones y gestionar el ciclo de vida completo de cada análisis, fundamental para

la naturaleza en tiempo real de Code-XR.

Registro centralizado de sesiones El sistema implementa el patrón Registry a través de `UnifiedSessionRegistry`, proporcionando:

- Repositorio centralizado con identificadores únicos (UUID v4)
- Control de transiciones de estado (creada → analizando → monitorizando/error)
- Gestión de metadatos, progreso y recursos asociados
- Prevención de duplicación de análisis idénticos
- APIs para consultas por ID, ruta o estado

Modelo de estado de sesión Ciclo de vida con estados discretos y transiciones estrictas:

1. **created**: Sesión inicializada, análisis pendiente
2. **analyzing**: Análisis en progreso con porcentaje opcional
3. **monitoring**: Análisis completado, observando cambios
4. **error**: Error que impide continuar
5. **closed**: Sesión terminada, recursos liberándose

Estructura de una sesión de análisis Cada sesión `UnifiedAnalysisSession` captura información completa para gestión autónoma: identificación (ID, ruta, tipo), estado y progreso, recursos asociados (hash, directorio, watcher, puerto, URL) y configuración (archivos requeridos, archivos a monitorizar).

Persistencia de configuraciones y resultados Esquema sofisticado de persistencia multinivel:

- **Archivos de análisis**: Directorio único por sesión en `workspaceStorage` con HTML/CSS/JS, `data.json`, metadatos y recursos auxiliares
- **Persistencia temporal**: Archivos mantenidos durante sesión VS Code para reanudación rápida
- **Configuraciones personalizadas**: Almacenadas en `globalState` para reutilización

Gestión del ciclo de vida y liberación de recursos Control de recursos para prevenir fugas de memoria y bloqueos de puertos:

- **Registro de recursos**: Referencias explícitas a watchers, servidores y archivos temporales
- **Liberación coordinada**: Detención ordenada de watchers, cierre de servidores HTTP/SSE, liberación de puertos y marcado de archivos para eliminación
- **Limpieza automática**: Al desactivar extensión, limpieza periódica de sesiones inactivas y al cerrar VS Code
- **Recuperación ante fallos**: Detección de puertos abandonados, watchers huérfanos y regeneración de archivos corruptos

Arquitectura de eventos y notificaciones `SessionRegistry` implementa sistema de eventos para arquitectura desacoplada:

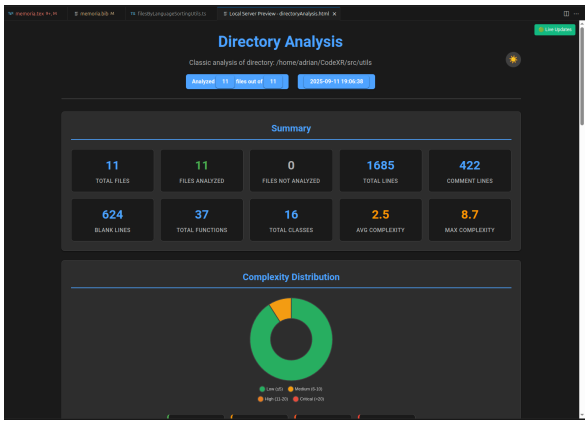
- **Eventos de ciclo de vida**: Notifican creación, cambio de estado y cierre
- **Eventos de progreso**: Indican avances en análisis
- **Eventos de error**: Comunican problemas durante análisis

Integración con watchers y servidores Coordinación con subsistemas asociados: cada sesión mantiene watchers específicos, servidores activos con canales SSE, asignación de puertos únicos y configuración de debounce a nivel sesión, garantizando operación autónoma sin interferencias entre sesiones.

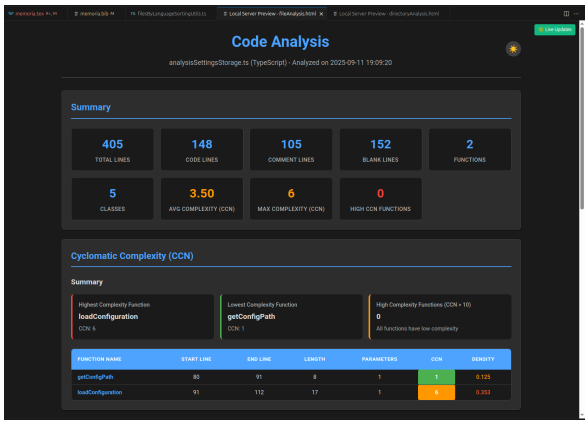
4.5.6. Tipos de análisis

El subsistema de análisis de Code-XR ofrece tres tipos de análisis distintos, cada uno especializado en extraer diferentes aspectos del código fuente. Estos tipos de análisis determinan qué métricas se calculan y cómo se estructuran los datos, pudiendo visualizarse posteriormente en panel lateral, navegador web o dispositivos VR/AR según la configuración del servidor.

LivePanel: análisis tradicional de métricas de código El análisis LivePanel extrae métricas completas de código fuente proporcionando información estructurada sobre complejidad, calidad y organización del código. El la figura 4.17 se muestran ejemplos de análisis tanto para directorios completos como para ficheros individuales, destacando métricas clave como complejidad ciclomática, distribución de líneas de código y comentarios, y estructura de clases.



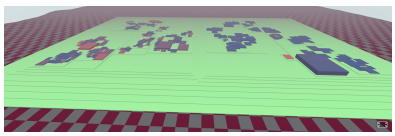
(a) Análisis LivePanel de directorio



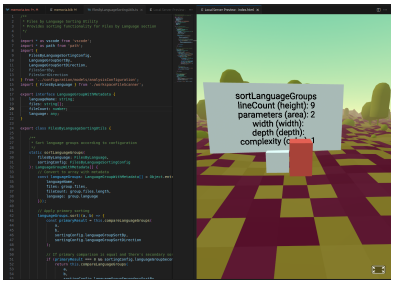
(b) Análisis LivePanel de fichero individual

Figura 4.17: Análisis LivePanel mostrando métricas tradicionales de código tanto para directorios completos como para ficheros individuales.

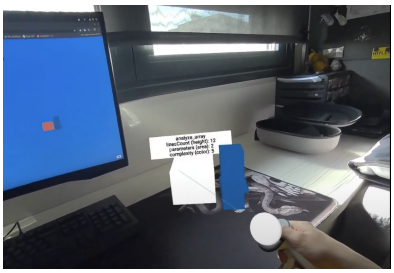
XR: análisis optimizado para visualización tridimensional El análisis XR procesa código fuente para generar representaciones espaciales tridimensionales, transformando métricas abstractas en elementos visuales navegables mediante A-Frame y BabiaXR. A continuación en la figura 4.18 se muestran ejemplos de análisis XR para diferentes alcances (directorio y fichero) y su experiencia inmersiva en realidad aumentada.



(a) Análisis XR de directorio



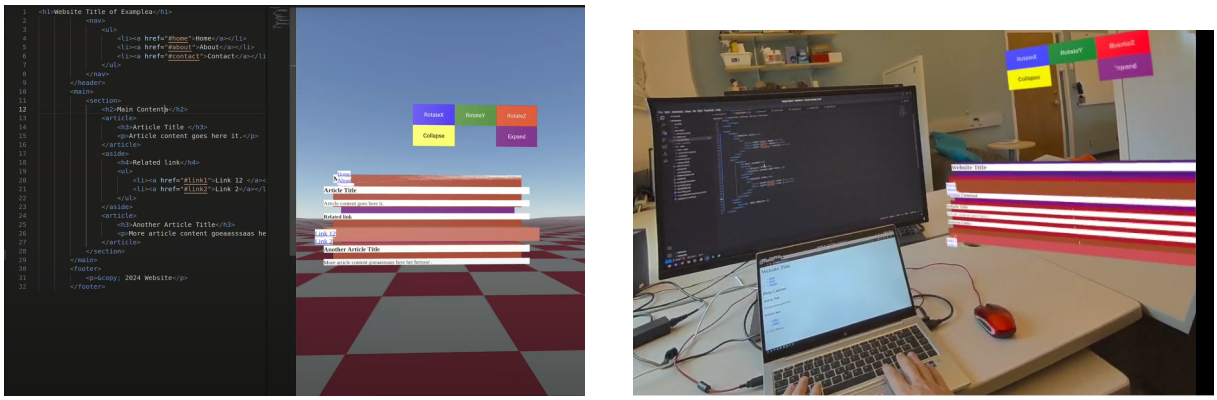
(b) Análisis XR de fichero



(c) Experiencia AR/VR

Figura 4.18: Análisis XR mostrando visualizaciones tridimensionales para diferentes alcances y su experiencia inmersiva en realidad aumentada.

DOM: análisis especializado para documentos HTML El análisis DOM está específicamente diseñado para extraer y procesar la estructura jerárquica de archivos HTML, transformando el Document Object Model en datos navegables para visualización tridimensional. El resultado de este tipo de análisis se puede ver en la figura 4.19, donde se muestran ejemplos del análisis DOM de un documento HTML y su experiencia inmersiva en realidad aumentada.



(a) Análisis DOM de documento HTML

(b) Experiencia AR del árbol DOM

Figura 4.19: Análisis DOM mostrando la estructura jerárquica de documentos HTML y su representación inmersiva en realidad aumentada.

Modos de visualización universales Independientemente del tipo de análisis seleccionado (LivePanel, XR o DOM), todos pueden visualizarse según la configuración de lanzamiento del servidor establecida en el Sprint 4.3:

- **Panel lateral integrado**: Visualización directa dentro de VS Code mediante WebViews, manteniendo el contexto de desarrollo
- **Navegador web**: Apertura en navegador externo para visualización de pantalla completa con mejor rendimiento
- **Dispositivos VR/AR**: Acceso desde dispositivos de realidad virtual o aumentada conectados a la misma red

4.5.7. Análisis en tiempo real

Code-XR proporciona análisis en tiempo real mediante actualización automática de visualizaciones durante la edición de código, ofreciendo retroalimentación inmediata sobre

calidad y estructura del software.

Sistema de watchers para detección de cambios Sistema sofisticado de observadores de archivos mediante `FileSystemWatcher` de VS Code:

- **Watchers granulares**: Observadores específicos por archivo/directorio con creación/destrucción dinámica
- **Gestión por sesión**: Cada sesión mantiene watchers independientes para análisis paralelos
- **Filtros inteligentes**: Exclusión automática de archivos irrelevantes (configuración, binarios)
- **Eventos detectados**: `create`, `change`, `delete`, `rename`

Estrategia de debounce para optimización Agrupación de eventos cercanos temporalmente para evitar análisis excesivos:

- **Temporizador configurable**: Intervalo ajustable (predeterminado: 1000ms)
- **Agrupación inteligente**: Múltiples cambios procesados como evento único
- **Reconfiguración dinámica**: Actualización sin reinicio de análisis

Análisis incremental y diferencial Optimización mediante procesamiento selectivo de modificaciones:

- **Detección por hash**: Identificación de modificaciones reales mediante hash de contenido

Sincronización de cambios con visualizaciones Mecanismo robusto de propagación desde detección hasta actualización visual:

1. **Detección**: Watcher identifica modificación
2. **Debounce**: Espera intervalo sin cambios adicionales
3. **Re-análisis**: Ejecución del motor Python solo en archivos modificados

4. **Actualización datos**: Generación de nuevo `data.json`

5. **Evento SSE**: Emisión de `analysis-updated`

6. **Actualización cliente**: Solicitud y renderizado incremental

En la figura 4.20 se muestra un diagrama del ciclo de re-análisis y actualización en tiempo real, ilustrando las etapas desde la detección de cambios hasta la actualización de la visualización.

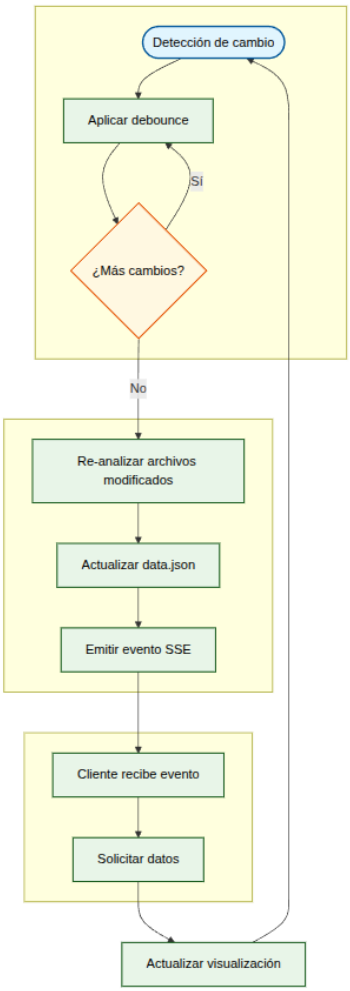


Figura 4.20: Diagrama del ciclo de re-análisis y actualización en tiempo real.

Rendimiento y optimización Optimizaciones para mantener capacidad de respuesta en proyectos grandes:

- **Análisis por lotes**: Procesamiento de archivos en lotes para evitar bloqueo de UI
- **Procesamiento asíncrono**: Operaciones en hilos separados

- **Limitación adaptativa**: Restricciones dinámicas según carga del sistema
- **Cancelación inteligente**: Cancelación automática de análisis obsoletos

4.5.8. Comunicación en tiempo real (SSE)

Code-XR reutiliza la infraestructura de Server-Sent Events (SSE) desarrollada en el Sprint 4.3 para proporcionar actualizaciones asíncronas en tiempo real de las visualizaciones durante el análisis de código, eliminando la necesidad de recargar manualmente y ofreciendo retroalimentación inmediata.

Reutilización de la arquitectura SSE del Sprint 1 Cada sesión de análisis aprovecha el sistema de servidores HTTP/HTTPS establecido previamente, manteniendo canales SSE dedicados para comunicar cambios entre el motor de análisis y las visualizaciones activas.

Flujo de comunicación entre backend y frontend Patrón secuencial optimizado que conecta análisis con visualización:

1. **Conexión**: cliente conecta automáticamente a endpoint SSE
2. **Estado inicial**: servidor envía estado actual inmediatamente
3. **Eventos**: listeners configurados para cada tipo, actualización UI correspondiente
4. **Datos**: evento `analysis-updated` → petición HTTP a `/data.json`
5. **Actualización parcial**: solo componentes afectados, preservando navegación
6. **Reconexión**: automática con backoff exponencial

A continuación, en la figura 4.21 se muestra un diagrama detallado del flujo de comunicación SSE entre el backend (análisis) y el frontend (visualización), ilustrando las interacciones clave y la secuencia de eventos que permiten la actualización en tiempo real de las visualizaciones.

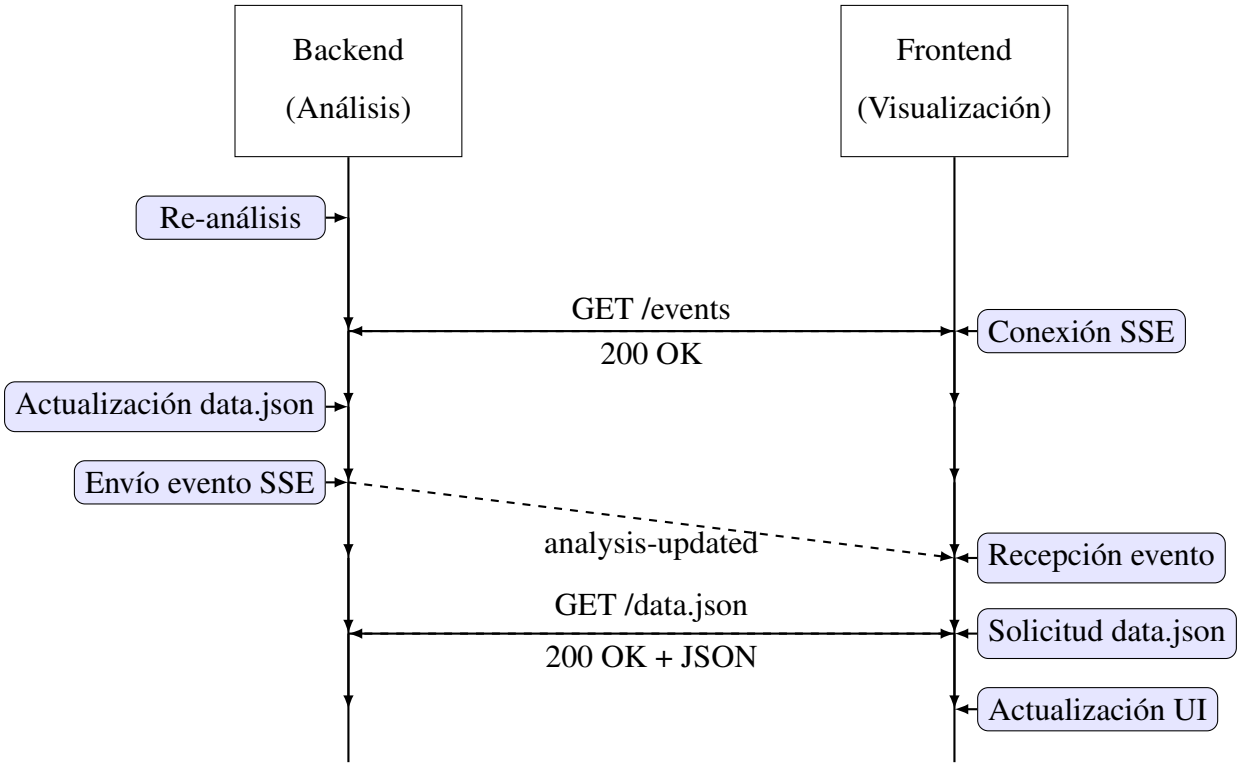


Figura 4.21: Diagrama del flujo de comunicación SSE entre el backend (análisis) y el frontend (visualización).

Implementación en diferentes modos de visualización Cada modo de visualización implementa el protocolo SSE de manera específica:

- **LivePanel**: utiliza la API WebView de VS Code para cargar directamente el código JavaScript que establece la conexión SSE y actualiza el DOM según los eventos recibidos.
- **XR**: implementa la lógica de conexión SSE en el código de A-Frame/BabiaXR, actualizando entidades y componentes 3D en respuesta a los eventos.
- **DOM Mode**: establece una conexión SSE dedicada para monitorizar cambios en la estructura DOM del archivo HTML analizado.

4.5.9. Configuración y personalización

Code-XR implementa un sistema integral de configuración mediante interfaz dedicada en el panel lateral, permitiendo personalización completa del comportamiento de análisis y apariencia visual según necesidades específicas del usuario.

Interfaz de configuración unificada Panel centralizado organizando todas las opciones de personalización:

- **Selección de análisis**: fichero individual o directorio completo, que puede visualizarse en LivePanel o XR, con opción `deep` para directorios que activa análisis exhaustivo de dependencias y relaciones
- **Tema visual LivePanel**: modo claro/oscuro optimizado para diferentes condiciones de iluminación, afectando únicamente visualizaciones LivePanel para mantener legibilidad
- **Gráficos XR**: selección entre tipos BabiaXR especializados (Bar, Barsmap, Bubble, Cylinder, Boats) según Figura 4.7, cada uno optimizado para diferentes tipos de datos y densidades
- **Mapeo de dimensiones XR**: asignación flexible de campos de datos a propiedades visuales XR (key, size, color, height, área) con validación automática de tipos numéricos y compatibilidad de datos, referenciado en Figura 4.16
- **Parámetros de ejecución**: debounce time configurable para análisis consecutivos evitando sobrecarga durante edición continua, auto-análisis automático susceptible de activación post-guardado para retroalimentación inmediata
- **Ordenación Files by Language**: criterios jerárquicos primario/secundario (alfabético/tamaño/fecha), dirección ascendente/descendente, orden interno que puede personalizarse por fecha de modificación o alfabético para organización óptima del workspace

Valores por defecto del sistema La configuración inicial incorpora valores optimizados basados en análisis de usabilidad y rendimiento:

- **Fichero de análisis**: XR - análisis en fichero individual por defecto minimizando sobrecarga computacional
- **Directorio de análisis**: XR (sin deep) - análisis superficial preservando rendimiento, evitando análisis recursivo exhaustivo de dependencias

- **Tema visual**: Dark - optimizado para sesiones prolongadas de desarrollo, reduciendo fatiga visual en entornos de trabajo típicos de IDE
- **Tipo de gráfico**: Boats - visualización equilibrada entre densidad de información y navegabilidad XR para ambos contextos (ficheros y directorios)
- **Mapeo de dimensiones ficheros**: Área=parameters, Height=lineCount, Color=Complexity - correlación directa entre complejidad estructural y propiedades visuales XR
- **Mapeo de dimensiones directorios**: Área=Function Count, Height=total Lines, Color=Mean Complexity - métricas agregadas representando características arquitecturales del módulo
- **Debounce time**: 3 segundos - equilibrio entre responsividad y eficiencia computacional durante edición activa de código
- **Auto-análisis**: Habilitado - retroalimentación automática post-guardado facilitando desarrollo iterativo y detección temprana de issues
- **Ordenación Files by Language**: Primaria alfabética ascendente, secundaria desactivada - organización intuitiva del workspace por nomenclatura estándar de ficheros
- **Ordenación Files**: alfabética ascendente - consistencia en presentación de resultados independiente del contexto de análisis

Como se muestra en la figura 4.22, el panel de configuración completo de Code-XR consolida todas estas opciones de personalización en una interfaz unificada e intuitiva, proporcionando al usuario control granular sobre el comportamiento del análisis, la apariencia visual y los parámetros de ejecución desde un único punto de acceso.

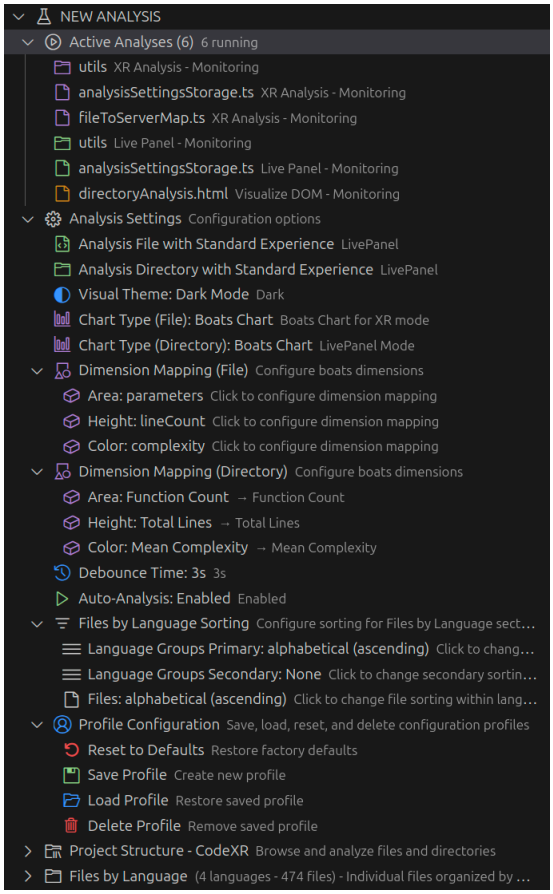


Figura 4.22: Panel de configuración completo de Code-XR mostrando todas las opciones de personalización: selección de análisis, tema visual, gráficos XR, mapeo de dimensiones, parámetros de ejecución y ordenación.

Integración con Visualization Settings Herencia completa de configuraciones estéticas del Sprint 2: Background Color (fondo visualizaciones), Ground Color (suelo XR), Environment Preset (ambientes virtuales forest/tron/dream), Chart Palette (esquemas cromáticos). Garantiza coherencia visual entre ejemplos BabiaXR, visualizaciones JSON y análisis de código.

Interfaz de configuración Code-XR proporciona múltiples puntos de acceso para gestionar la configuración:

- **Panel de configuración de VS Code**: integrado con la configuración estándar del editor bajo la sección `codeXR`.
- **Comandos rápidos**: accesibles mediante la paleta de comandos (F1) para cambios comunes.

- **Menú contextual**: opciones específicas disponibles con clic derecho en archivos y carpetas.
- **Árbol de extensión**: opciones accesibles desde los nodos del árbol de Code-XR en el panel lateral.
- **Configuración por sesión**: opciones disponibles en la propia interfaz de visualización para ajustes específicos.

Gestión de perfiles de usuario Sistema de perfiles para persistencia y reutilización de configuraciones personalizadas:

- **Operaciones básicas**: guardar, cargar, reiniciar y eliminar perfiles de configuración
- **Reutilización**: aplicación de ajustes específicos en diferentes sesiones de trabajo
- **Persistencia**: almacenamiento en `globalState` para disponibilidad entre sesiones VS Code

Gestión de análisis activos Sección `ActiveAnalyses` para monitorización y control de sesiones en ejecución:

- **Acceso rápido**: lista completa de análisis lanzados con información de estado
- **Menú contextual**: opciones específicas mediante clic derecho
  - `ViewDetails`: visualización completa de metadatos de sesión (Session ID, ruta objetivo, modo, timestamps, configuración, progreso)
  - `OpeninBrowser`: apertura directa del servidor de análisis en navegador externo
  - `StopAnalysis`: detención del análisis manteniendo servidor activo para visualización del estado final
- **Coordinación con Active Servers**: cierre de servidor desde Active Servers detiene análisis correspondiente automáticamente

Estructura de perfil encapsulando configuración completa:

```json
{
  "metadata": {
    "version": "1.0.0",
    "createdBy": "CodeXR New Code Analysis"
  },
  "configuration": {
    "analysisFileMode": "XR",
    "analysisDirectoryMode": "XR",
    "viewTheme": "Dark",
    "autoAnalysisDelay": {
      "type": "3s",
      "autoAnalysisEnabled": true
    },
    "chartTypeFile": "boats",
    "chartTypeDirectory": "boats",
    "dimensionMappingFile": {
      "area": "parameters",
      "height": "lineCount",
      "color": "complexity"
    },
    "dimensionMappingDirectory": {
      "area": "functionCount",
      "height": "totalLines",
      "color": "cyclomaticComplexityNumber"
    },
    "filesByLanguageSorting": {
      "languageGroupSortBy": "alphabetical",
      "languageGroupSortDirection": "ascending",
      "filesSortBy": "alphabetical",
      "filesSortDirection": "ascending"
    }
  }
}
```

Listing 4.3: Estructura de un perfil de análisis en Code-XR mostrando el perfil por defecto con todas las configuraciones del sistema.

Navegación de proyectos integrada Interfaces especializadas para lanzamiento directo de análisis:

- **Project Structure**: árbol jerárquico de carpetas y archivos con análisis contextual directo
- **Files by Language**: agrupación automática por lenguaje detectado con análisis de fichero individual

4.5.10. Optimización y extensibilidad

El subsistema de análisis de Code-XR está diseñado con un enfoque en la optimización del rendimiento y la extensibilidad, permitiendo tanto un funcionamiento eficiente como la adición futura de nuevos tipos de análisis y visualizaciones.

Estrategias de optimización de rendimiento Optimizaciones para mantener rendimiento ágil en proyectos complejos:

- **Análisis incremental**: Re-análisis exclusivo de archivos modificados preservando resultados previos
- **Paralelización**: Distribución entre múltiples procesos Python aprovechando núcleos disponibles
- **Procesamiento por lotes**: Archivos procesados en lotes configurables optimizando memoria y permitiendo visualizaciones parciales tempranas
- **Caché multinivel**: Memoria para resultados recientes, disco para persistencia entre sesiones, invalidación inteligente por hash de contenido

Arquitectura modular extensible Sistema de plugins facilitando adición de componentes sin modificar núcleo:

- **Analizadores pluggable**: Interfaz común `CodeAnalyzer` con métodos `analyze()`, `canAnalyze()`, `supportedLanguages`
- **Visualizaciones extensibles**: Sistema templates y adaptadores mediante `VisualizationProvider` con `getTemplateFiles()`, `transformData()`
- **Motor plantillas**: Archivos HTML/CSS/JS con marcadores, sistema transformación, motor sustitución y biblioteca componentes reutilizables

4.6. Nota sobre la documentación técnica del proyecto

La documentación técnica presentada a lo largo de este documento (Primeros pasos, Sprint 1, Sprint 2 y Sprint 3) constituye una síntesis significativa de la implementación completa

de Code-XR. Debido a las limitaciones de espacio inherentes a una memoria de TFG, se ha condensado considerablemente la información técnica para destacar únicamente los aspectos arquitectónicos y funcionales más relevantes del sistema.

Para una comprensión completa y exhaustiva de todos los aspectos técnicos de la implementación, se recomienda encarecidamente consultar el repositorio público del proyecto: `https://github.com/aMonteSl/CodeXR` [15]

Todo el código fuente está disponible bajo licencia MIT, incluyendo documentación técnica completa, guías de desarrollo, ejemplos prácticos y recursos para facilitar la comprensión, extensión y contribución al proyecto por parte de desarrolladores e investigadores interesados.

Capítulo 5

Experimentos y validación

El objetivo fundamental de este TFG era la extracción y posteriormente la representación visual de métricas en experiencias XR gracias a A-Frame y BabiaXR. Además, el plugin ofrece la posibilidad de ver esas métricas en un formato más clásico gracias al análisis tipo LivePanel, y también permite visualizar el árbol DOM de un HTML en experiencia XR.

5.1. Metodología de pruebas y depuración

Para verificar el correcto funcionamiento de todos los componentes del plugin, se implementó una estrategia de depuración completa:

- **Depuración interna**: El plugin incluye numerosas trazas con `console.log`. Para depurarlo, lanzar el debugger con F5 en VS Code y abrir las *Developer Tools* (Ctrl+Shift+I): allí se visualizan todos los logs y se puede seguir el flujo interno del plugin sin depender solo de su comportamiento visual. Se puede ver un ejemplo de como se visualiza la depuración interna en la figura 5.1, donde se ve la herramienta de depuración de VS Code mostrando los logs del plugin.

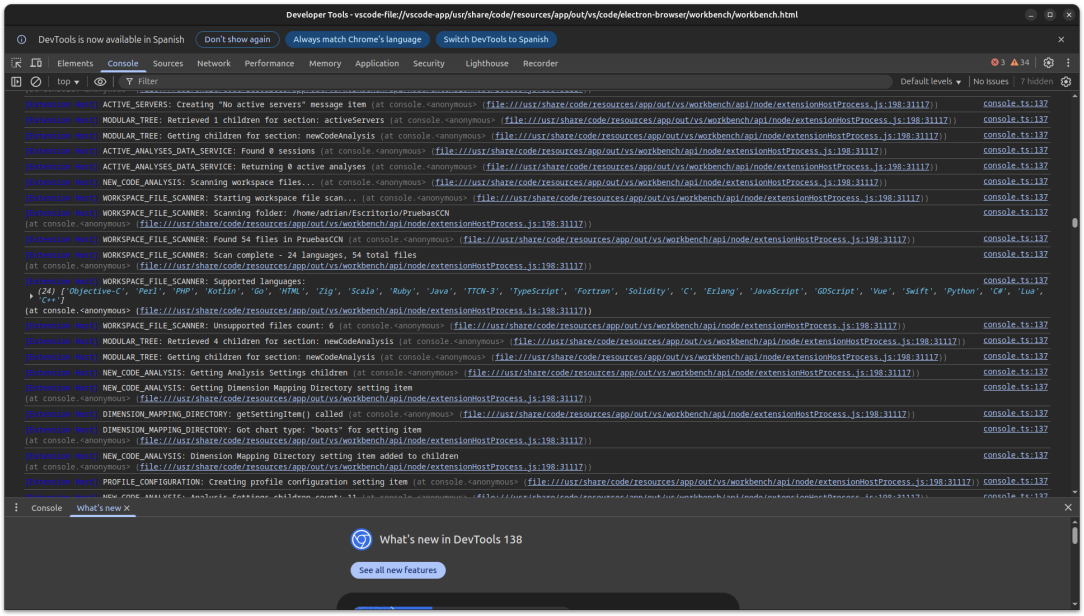


Figura 5.1: Developer Tools de VS Code mostrando los console.logs del plugin durante la ejecución.

5.2. Validación de la comunicación en tiempo real

Para comprobar que la conexión SSE (Server-Sent Events) en los diferentes modos de análisis estaba funcionando correctamente, se utilizaron:

- La consola de Developer Tools de VS Code, como se muestra en la figura 5.1, donde se pueden ver los logs del plugin y verificar que se reciben los eventos SSE correctamente.
- La consola del navegador donde estaba abierto el análisis (de cualquier tipo), como se muestra en la figura 5.2, donde se puede ver la recepción de eventos SSE en tiempo real.

Estas herramientas permitieron verificar que los cambios y el envío de datos mediante SSE se realizaban correctamente, confirmando que la conexión bidireccional estaba establecida y funcionando.

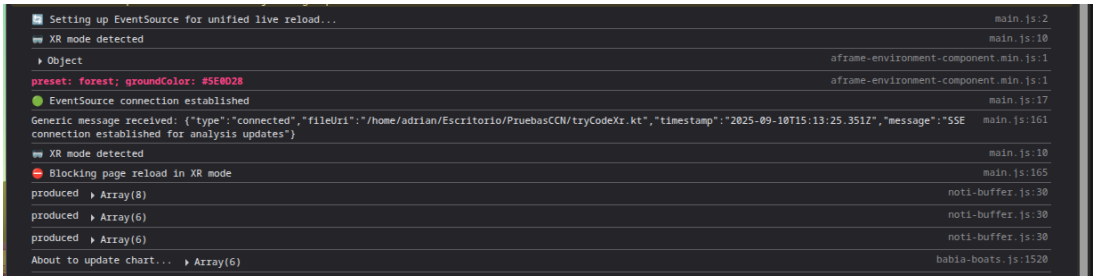


Figura 5.2: Consola del navegador mostrando la recepción de una actualización SSE.

5.3. Pruebas en entornos de realidad virtual

La comprobación del funcionamiento de las experiencias XR se realizó gracias al uso de las gafas de realidad virtual Meta Oculus Quest 3 proporcionadas por la universidad. Estas pruebas permitieron:

- Probar, observar y confirmar que todo en cuanto a experiencias de realidad virtual estaba funcionando correctamente
- Validar la usabilidad de la interfaz en un entorno inmersivo real
- Verificar la correcta actualización de la visualización en tiempo real

El procedimiento para conectarse al servidor desde las gafas es sencillo: el plugin informa de la IP interna con la que se ha lanzado el servidor (como se ve en la figura 5.3), y en el navegador de las gafas, estando en la misma red WiFi, se introduce esa dirección IP y puerto para disfrutar de la experiencia VR/AR.

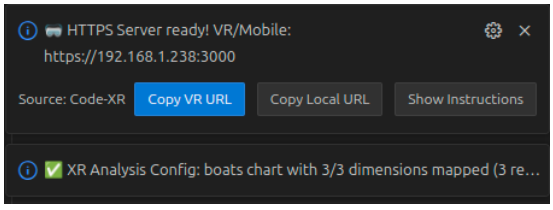


Figura 5.3: Información proporcionada por el plugin sobre la dirección IP para la conexión desde dispositivos externos.

Capítulo 6

Resultados

6.1. Code-XR

El inicio de este TFG empezó con la intención de mostrar las métricas de análisis de ficheros únicos, es decir, solo desarrollar el análisis de fichero en formato XR. Sin embargo, el alcance del proyecto fue ampliándose progresivamente: no solo se desarrolló el modo LivePanel para análisis de ficheros, sino que también se tuvo la ambición de lograr representar el árbol DOM en experiencia XR con actualizaciones en vivo y, además, se dio el salto cualitativo de añadir análisis de directorios completos.

En todo este proceso se logró implementar las siguientes funcionalidades:

- **Análisis de ficheros individuales**: Extracción de métricas como complejidad ciclomática, número de líneas y número de parámetros.
- **Visualización en modo LivePanel**: Representación bidimensional integrada en VS Code.
- **Visualización en modo XR**: Experiencia inmersiva para explorar métricas de código en entornos de realidad virtual/aumentada.
- **Visualización del árbol DOM**: Representación tridimensional de la estructura DOM de archivos HTML.
- **Análisis de directorios completos**: Procesamiento recursivo de todos los archivos en un directorio.

- **Actualizaciones en tiempo real**: Sistema de Server-Sent Events (SSE) para mantener sincronizadas las visualizaciones con cambios en el código.
- **Sistema de configuración extensible**: Opciones para personalizar el comportamiento y la apariencia de las visualizaciones.
- **Creación de escenas A-Frame/BabiaXR**: Generación dinámica de escenas XR con metáforas visuales adaptadas a diferentes tipos de datos y métricas.
- **Integración completa con la API de VS Code**: Implementación de comandos, vistas personalizadas, webviews y extensión del sistema de tareas.
- **Gestión de entornos Python**: Detección y configuración automática de entornos virtuales Python para el análisis de código sin configuración manual.
- **Sistema de servidores multiproceso**: Arquitectura que permite mantener múltiples servidores de análisis simultáneos sin bloquear la interfaz de usuario.
- **Compatibilidad multiplataforma**: Funcionamiento en Windows, macOS y Linux con adaptación automática a particularidades de cada sistema.
- **Manejo Marketplace VS Code**: Publicación y mantenimiento del plugin en el Marketplace oficial de Visual Studio Code.
- **Documentación y recursos públicos**: Creación de una página web con guías prácticas, guías de instalación, ejemplos y recursos académicos.

El plugin además cuenta con la posibilidad de personalizar la experiencia del usuario y de disfrutar de la potencia del plugin de distintas formas, adaptándose a las necesidades y preferencias específicas de cada desarrollador.

6.2. Resultados en proyectos relevantes

Para demostrar que Code-XR no es solo un concepto académico sino una herramienta con aplicación práctica real, se llevaron a cabo pruebas extensivas con tres proyectos de software reales:

Proyecto	Archivos	Descripción
BabiaXR [26]	77	Framework para visualización de datos en XR
Express.js [6]	142	Framework web minimalista para Node.js
JetUML [7]	338	Herramienta ligera para diagramas UML

Tabla 6.1: Proyectos utilizados para validar Code-XR

En todos estos proyectos se realizaron pruebas progresivas, desde el análisis de archivos individuales hasta el análisis completo de todos los archivos del proyecto, generando visualizaciones como la siguiente que se muestra en la figura 6.1, donde se puede observar el análisis XR del proyecto JetUML con 338 archivos analizados.

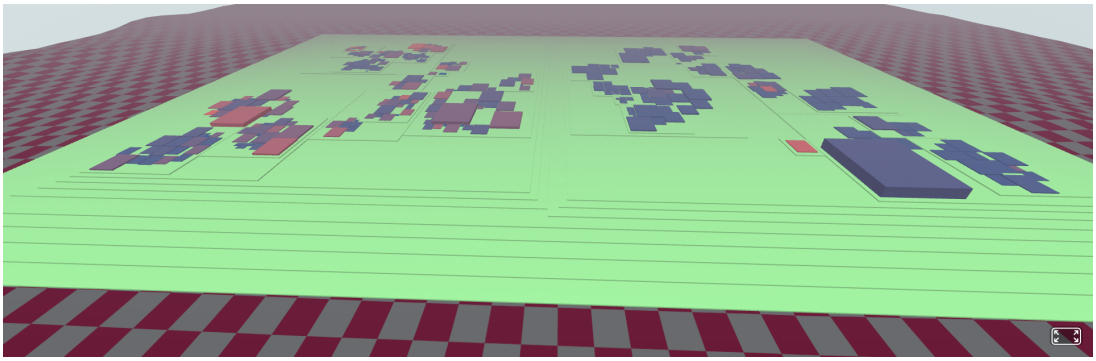


Figura 6.1: Visualización 3D de un análisis completo de proyecto con Code-XR

6.3. Disponibilidad del software

Code-XR cuenta con una página web pública alojada en GitHub Pages (`https://amontesl.github.io/code-xr-docs/`), que sirve como punto central de acceso a toda la información relacionada con el proyecto. En esta página web se pueden encontrar:

- **Galería de vídeos**: Colección de guías prácticas y demostraciones que muestran el funcionamiento del plugin en diferentes escenarios.
- **Proyectos reales**: Sección dedicada a los tres proyectos mencionados anteriormente (BabiaXR, Express.js y JetUML) con vídeos específicos de cada uno y un dashboard interactivo para cada proyecto que permite previsualizar cómo se ve el análisis en XR, similar a la visualización mostrada en la Figura 6.1.

- **Guías de instalación**: Instrucciones detalladas para instalar Code-XR desde el Marketplace oficial de VS Code, mediante línea de comandos, o descargando directamente el archivo VSIX desde GitHub Releases.
- **Guía rápida de inicio**: Tutorial paso a paso para poner en marcha y realizar un primer análisis en menos de 10 minutos.
- **Recursos académicos**: En esta sección se pueden encontrar y descargar tanto este mismo documento PDF como la presentación del TFG y otros materiales relacionados con el proyecto.

Vídeos recomendados para visualizar Todos estos vídeos están disponibles en la galería de la página web. Se recomienda encarecidamente visualizarlos para contemplar todas las posibilidades de este plugin, ya que por motivos de espacio en esta memoria es imposible documentar con imágenes todos los escenarios posibles. Además, al visualizar los vídeos se puede llegar a ver y entender mejor el funcionamiento en tiempo real de Code-XR:

1. **Interface UI**: `https://youtu.be/KRgLdLZJXHA`
2. **File Analysis LivePanel Mode**: `https://youtu.be/n5ZcjlR4pPc`
3. **File Analysis XR Mode**: `https://youtu.be/38jGwFGORvc`
4. **Directory Analysis LivePanel Mode**: `https://youtu.be/sPWjcgV-gZQ`
5. **Directory Analysis XR Mode**: `https://youtu.be/TnfS2SevtWU`
6. **Análisis de un proyecto real (BabiaXR)**: `https://youtu.be/NluAHe3BQu8`
7. **HTML DOM Tree Visualization**: `https://youtu.be/110b-AergdU`
8. **Experiencia programar con Code-XR en AR**: `https://youtu.be/d7fojpP90Dk`

El software es libre y de código abierto, disponible bajo licencia MIT en el repositorio oficial: `https://github.com/aMonteSl/CodeXR` [15].

6.4. VISSOFT/ICSME 2025

El proyecto Code-XR fue aceptado y presentado el 7 de septiembre de 2025 en la conferencia internacional VISSOFT/ICSME 2025 (IEEE Working Conference on Software Visualization / International Conference on Software Maintenance and Evolution) [13] en la categoría de poster track.

La versión presentada (0.0.8) era ligeramente anterior a la versión actual (1.0.0) y contenía las funcionalidades de análisis de ficheros individuales y visualización del DOM, pero aún no incluía el análisis de directorios completos que se ha incorporado posteriormente.

El paper completo aceptado en la conferencia está disponible para su descarga desde la página web oficial de Code-XR en la sección de Recursos, o directamente a través del DOI permanente: `https://zenodo.org/records/16352142`.

Poster presentado en la conferencia El poster académico presentado durante la conferencia sintetiza visualmente las principales características y contribuciones de Code-XR, mostrando su arquitectura, funcionalidades clave y ejemplos de visualización en tiempo real. La figura 6.2 muestra el poster completo tal como fue exhibido en el evento, consolidando de manera visual los aspectos más relevantes del proyecto y su impacto en el campo de la visualización de software.

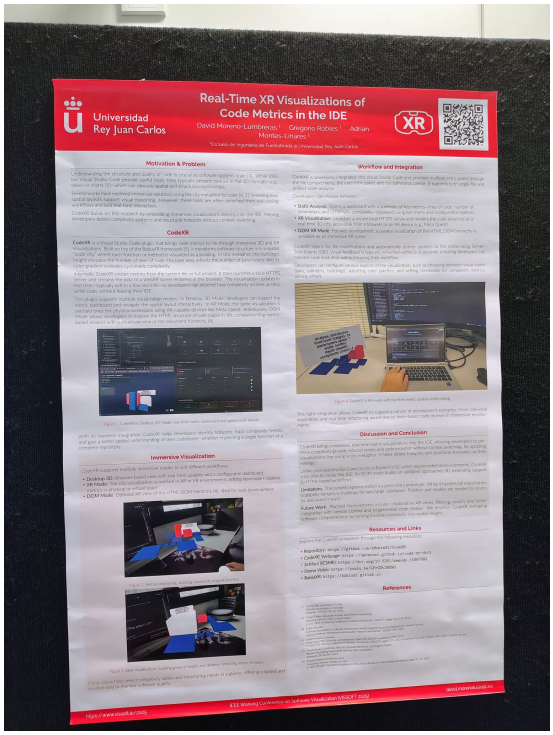


Figura 6.2: Poster de Code-XR presentado en la conferencia internacional VISSOFT/ICSME 2025, mostrando la arquitectura del sistema, las principales funcionalidades y ejemplos de visualización en tiempo real.

Esta aceptación en un foro académico internacional de prestigio confirma la relevancia y el valor de Code-XR como contribución al campo de la visualización de software, validando tanto su enfoque innovador como su implementación práctica.

6.5. Difusión y adopción de Code-XR

Además de su validación técnica, Code-XR ha sido publicado en el *Visual Studio Code Marketplace*, lo que ha permitido evaluar de manera preliminar su grado de difusión y adopción entre la comunidad de desarrolladores. La Figura 6.3 muestra las estadísticas de uso registradas en los últimos 90 días, que constituyen el rango máximo disponible en la plataforma oficial.

Durante este periodo se han alcanzado **143 nuevas adquisiciones (90 últimos días)**, acumulando un total de **271 instalaciones desde su publicación**. El gráfico de tendencias refleja una evolución sostenida con picos de actividad asociados a momentos de mayor visibilidad del proyecto (principalmente publicaciones de nuevas versiones). Cabe destacar que la tasa

de conversión entre visualizaciones de la página de extensión y descargas es positiva, lo que evidencia interés real por parte de los usuarios que consultan el plugin.

Estos resultados, aunque preliminares, ponen de manifiesto que Code-XR no sólo constituye una contribución académica, sino también una herramienta con proyección práctica y potencial de crecimiento dentro de la comunidad de Visual Studio Code.

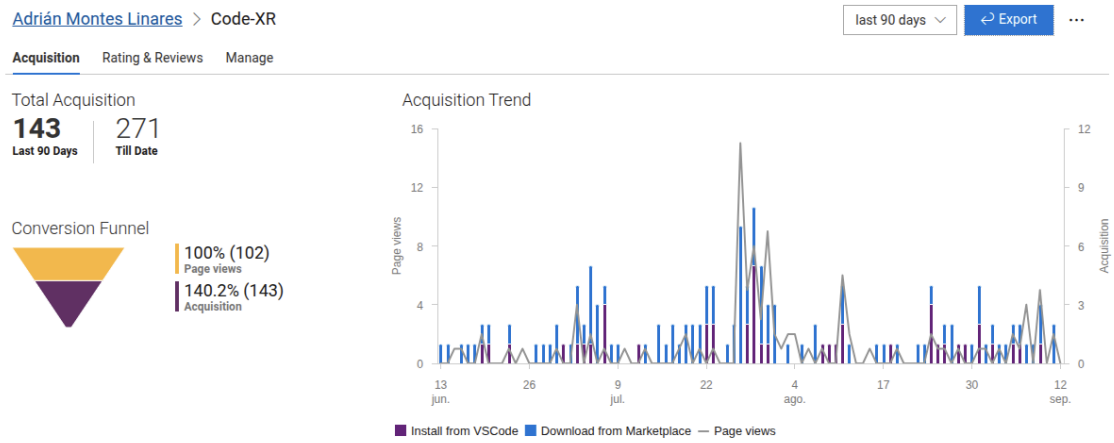


Figura 6.3: Estadísticas de adquisición de Code-XR en el *Visual Studio Code Marketplace* (últimos 90 días).

Capítulo 7

Conclusiones

7.1. Consecución de objetivos

Al finalizar este Trabajo Fin de Grado, se ha cumplido satisfactoriamente el objetivo principal y las contribuciones específicas planteadas para Code-XR:

- **Visualización inmersiva en tiempo real**: Sistema completo implementado que permite visualizar métricas de código en tiempo real desde VS Code, capturando y procesando cambios automáticamente sin intervención manual.
- **Tecnologías web XR accesibles**: Integración exitosa de A-Frame y BabiaXR para experiencias XR accesibles desde cualquier navegador, democratizando el acceso a visualizaciones tridimensionales sin software adicional.
- **Múltiples modos de visualización**: Desarrollo completo de tres modos (LivePanel, XR Mode, DOM Mode) optimizados para diferentes casos de uso, desde análisis integrados hasta experiencias inmersivas.
- **Arquitectura extensible**: Construcción de arquitectura modular eficiente con sistema de plugins, adaptadores y múltiples opciones de configuración personalizable.
- **Análisis de directorios completos**: Capacidad implementada para análisis recursivo de proyectos completos, visualizando estructuras de software a gran escala.
- **Sistema multiproceso**: Arquitectura que mantiene múltiples servidores simultáneos, facilitando comparaciones entre diferentes componentes del código.

- **Gestión automática Python**: Detección y configuración automática de entornos virtuales, eliminando configuración manual para análisis.

Los desafíos técnicos en tiempo real y optimización se resolvieron mediante análisis incremental, procesamiento por lotes y mecanismos de debounce. La aceptación en VISSOFT/ICSME 2025 valida la relevancia científica de Code-XR como contribución al campo de visualización de software mediante tecnologías XR.

7.2. Aplicación de lo aprendido

La realización de este TFG ha permitido aplicar y consolidar conocimientos adquiridos durante el Grado, integrando múltiples asignaturas en el desarrollo de Code-XR:

1. **Sistemas Telemáticos**: Conocimientos sobre estructura DOM y modelo cliente-servidor cruciales para desarrollar DOM Mode. El modelo Modelo-Vista-Controlador facilitó la arquitectura correcta, especialmente la separación entre motor de análisis y visualizaciones.

2. **Aplicaciones Telemáticas**: Profundización en HTML y JavaScript imprescindible para implementar visualizaciones A-Frame/BabiaXR, desarrollando estructura visual tridimensional y lógica de interacción XR.

3. **Robótica**: Introducción a Git, fundamental durante el desarrollo para mantener historial ordenado, facilitar experimentación y gestionar versiones publicadas eficientemente.

4. **Seguridad en Redes**: Conceptos de nonce y hash aplicados especialmente en el sistema de detección de cambios mediante hash de contenido.

5. **Sistemas Distribuidos**: Conocimientos básicos para manejar múltiples análisis paralelos, fundamental para la implementación multiproceso y coordinación entre componentes.

6. **Sistemas Operativos**: Dominio de terminal Linux esencial para gestión del entorno de trabajo, configuración de servicios y ejecución eficiente de procesos de soporte.

La integración de estos conocimientos fue esencial para superar retos técnicos como comunicación en tiempo real IDE-visualizaciones, gestión eficiente de recursos y procesos,

y creación de experiencias interactivas tridimensionales. Este TFG sirvió como culminación práctica de la formación recibida, demostrando la capacidad de aplicar conocimientos teóricos a un proyecto real con impacto tangible.

7.3. Lecciones aprendidas

El desarrollo de Code-XR ha proporcionado aprendizajes técnicos y metodológicos valiosos:

1. **Integración de tecnologías heterogéneas**: La combinación de entornos diversos (VS Code API, Node.js, Python, A-Frame, BabiaXR) converge en soluciones innovadoras estableciendo interfaces de comunicación claras entre componentes.

2. **Diseño arquitectónico inicial**: El diseño modular facilitó ampliaciones posteriores. La separación entre análisis, visualización y comunicación permitió añadir funcionalidades (análisis de directorios) sin reescribir código existente.

3. **Retroalimentación temprana**: El desarrollo incremental con versiones funcionales desde etapas iniciales resultó fundamental para detectar problemas de rendimiento y usabilidad tempranamente. El enfoque "mostrar en lugar de explicar"facilitó la validación continua.

4. **Optimización continua**: Las estrategias de optimización (análisis incremental, caché, procesamiento por lotes) deben incorporarse iterativamente identificando cuellos de botella. El sistema de watchers y debounce fue refinado repetidamente buscando equilibrio entre reactividad y eficiencia.

5. **Documentación integral**: Mantener documentación actualizada resultó tan importante como el código, especialmente para plantillas y API de eventos. La documentación clara facilitó implementación de nuevas características y creación de materiales académicos.

6. **Tecnologías web para XR**: El uso de tecnologías web estándar (A-Frame/WebXR) frente a plataformas propietarias permitió experiencias XR accesibles desde cualquier navegador, confirmando la democratización de visualizaciones inmersivas sin hardware especializado.

7. **Gestión de versiones**: El proceso Marketplace VS Code enseñó importancia del versionado semántico. Las múltiples versiones (0.0.1 a 1.0.0) permitieron iteración rápida y retroalimentación de usuarios reales.

8. **Experiencia de usuario (UX)**: Desarrollar interfaces intuitivas resultó tan crítico como la funcionalidad técnica. Los usuarios valoran simplicidad y resultados inmediatos sin configuración compleja.

9. **Comunicación y promoción (soft skills)**: La creación de contenido multimedia, documentación atractiva y presentación en VISSOFT/ICSME 2025 fueron habilidades esenciales. Comunicar ventajas técnicas de forma accesible para diferentes audiencias resultó fundamental para adopción.

10. **Gestión del alcance**: Equilibrar ambición técnica con viabilidad práctica. La evolución gradual desde objetivos modestos (análisis individual) hacia funcionalidades complejas (análisis directorios, múltiples modos) mantuvo calidad evitando complejidad excesiva.

11. **Validación real**: Las pruebas con proyectos reales (BabiaXR, Express.js, JetUML) identificaron limitaciones y casos no contemplados. Los prototipos académicos deben someterse a pruebas con datos reales para verificar utilidad y robustez.

Estas lecciones contribuyeron al éxito del proyecto y constituyen conocimientos aplicables a futuros desarrollos en visualización de software y tecnologías XR. Las habilidades blandas desarrolladas durante promoción y difusión demostraron ser tan valiosas como las competencias técnicas para el éxito integral.

7.4. Trabajos futuros

Code-XR presenta numerosas direcciones de expansión futuras como base para otros TFGs o desarrollos avanzados:

1. **Ampliación de métricas disponibles**: Incorporar métricas avanzadas como deuda técnica, cobertura de pruebas, acoplamiento entre componentes y cohesión de módulos para análisis más sofisticados sobre calidad y mantenibilidad del código.

2. **Experiencias colaborativas multiusuario**: Sistema multiusuario para visualizar y trabajar simultáneamente sobre análisis, facilitando sesiones de revisión en equipo con avatares representativos y herramientas para destacar y discutir aspectos específicos en tiempo real.

3. **Proyección de interfaz VS Code en XR**: Captura y retransmisión de la ventana del editor dentro del entorno XR, permitiendo consultar y modificar código sin depender de monitores físicos, mejorando la integración fluida entre programación y visualización.

4. **Integración con control de versiones**: Visualización temporal de la evolución del código mostrando cambios de métricas y estructura entre commits, proporcionando perspectiva histórica sobre calidad del software e identificación de problemas potenciales.

5. **Análisis predictivo mediante IA**: Modelos de aprendizaje automático para predecir áreas propensas a errores o refactorización, entrenados con datos históricos para identificar patrones que precedan la introducción de defectos.

6. **Interfaces gestuales avanzadas**: Controles gestuales sofisticados para manipular, filtrar y analizar código mediante gestos naturales, mejorando significativamente la experiencia en dispositivos VR como Meta Quest.

Estas propuestas ampliarían las capacidades de Code-XR abriendo nuevas posibilidades de investigación en visualización de software, experiencias inmersivas y herramientas de desarrollo colaborativo.

Apéndice A

Manual de usuario

El plugin es bastante sencillo de entender en cuanto a su manejo gracias a la interfaz gráfica. De todas formas, se dispone de un vídeo tutorial en la página web oficial del plugin, el cual detalla qué hace cada sección.

Para aquellos que quieran ver directamente el tutorial, pueden acceder a través del siguiente enlace: `https://youtu.be/KRgLdLZJXHA`

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] V. S. Code. Your first extension. https://www.youtube.com/watch?v=PGAu06_E_BU, 2024. Accedido: 2025-09-06.

[6] E. Contributors. Express.js. https://expressjs.com/, 2022. Accedido: 9 de septiembre de 2025.

[7] J. Contributors. Jetuml. https://www.jetuml.org/, 2022. Accedido: 9 de septiembre de 2025.

[8] 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.

[9] Devicon Community. Devicon: Icons for programming languages and tools. https://github.com/devicons/devicon, 2025. Accedido: 7 de septiembre de 2025.

[10] R. Dey. Live server — launch a local development server with live reload. https://marketplace.visualstudio.com/items?itemName=ritwickdey.LiveServer, 2025. Extensión de VS Code para servir páginas locales con recarga automática. Accedido: 7 de septiembre de 2025.

[11] Evan Wallace. esbuild — an extremely fast javascript bundler. https://esbuild.github.io/, 2024. Accedido: 2025-09-06.

[12] 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.

[13] IEEE. 2025 ieee working conference on software visualization (vissoft). https://vissoft.io/2025/, 2025. Accedido: 9 de septiembre de 2025.

[14] 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.

[15] A. M. Linares. Code-xr: Vs code extension for 3d/xr software visualizations. https://github.com/aMonteSl/CodeXR, 2025. Accedido: 7 de septiembre de 2025.

[16] A. M. Linares. Oficial website of code-xr. https://amontesl.github.io/code-xr-docs/, 2025. Accedido el 3 de agosto de 2025.

[17] Meta Platforms, Inc. Meta quest 3. https://www.meta.com/quest/quest-3/, 2024. Accedido el 3 de agosto de 2025.

[18] B. Meyer. Seven principles of software testing. *Computer*, 41(8):99–101, 2008.

[19] Microsoft. Extension context — global state and workspace state. https://code.visualstudio.com/api/references/vscode-api#ExtensionContext, 2024. Documentación sobre ExtensionContext.globalState y ExtensionContext.workspaceState. Accedido el 3 de agosto de 2025.

[20] Microsoft. Typescript: Javascript with syntax for types. https://www.typescriptlang.org/, 2024. Accedido el 3 de agosto de 2025.

[21] Microsoft. Visual studio code api. `https://code.visualstudio.com/api`, 2024. Disponible en `https://code.visualstudio.com/api`.

[22] Microsoft. Your first extension. `https://code.visualstudio.com/api/get-started/your-first-extension`, 2024. Accedido: 2025-09-06.

[23] Microsoft. Codicons: Vs code icon library. `https://microsoft.github.io/vscode-codicons/dist/codicon.html`, 2025. Accedido: 7 de septiembre de 2025.

[24] D. Moreno-Lumbreras. Enhancing html structure comprehension: Real-time 3d/xr visualization of the dom. In *2024 IEEE Working Conference on Software Visualization (VISSOFT)*, pages 127–132, 2024. `https://doi.ieeecomputersociety.org/10.1109/VISSOFT64034.2024.00025`.

[25] D. Moreno-Lumbreras. Babiaxr examples. `https://babiaxr.gitlab.io/aframe-babia-components/`, 2025. Accedido: 9 de septiembre de 2025.

[26] 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.

[27] Mozilla. A-frame: A web framework for building virtual reality experiences. `https://aframe.io/`, 2024. Accedido el 3 de agosto de 2025.

[28] Mozilla Developer Network. Javascript | mdn web docs. `https://developer.mozilla.org/docs/Web/JavaScript`, 2024. Accedido el 3 de agosto de 2025.

[29] OpenAI. Chatgpt. `https://chat.openai.com/`, 2024. Accedido el 3 de agosto de 2025.

[30] OpenSSL Software Foundation. Openssl: Cryptography and ssl/tls toolkit. `https://www.openssl.org/`, 2024. Herramienta para generación de certificados SSL/TLS; Accedido el 3 de agosto de 2025.

[31] Python Software Foundation. Python programming language. `https://www.python.org/`, 2024. Accedido el 3 de agosto de 2025.

[32] S. Sorhus and colaboradores. get-port — get an available tcp port (node.js). https://www.npmjs.com/package/get-port, 2025. Paquete npm para buscar puertos libres; repositorio: https://github.com/sindresorhus/get-port. Accedido: 7 de septiembre de 2025.

[33] 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.

[34] Visual Studio Code Team. Bundling extensions. https://code.visualstudio.com/api/working-with-extensions/bundling-extension, 2025. Accedido: 6 de septiembre de 2025.

[35] W3C. Css: Cascading style sheets. https://www.w3.org/Style/CSS/, 2024. Accedido el 3 de agosto de 2025.

[36] W3C. Webxr device api. https://www.w3.org/TR/webxr/, 2024. Accedido el 3 de agosto de 2025.

[37] Webpack Contributors. Webpack — static module bundler for modern javascript. https://webpack.js.org/, 2024. Accedido: 2025-09-06.

[38] 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.

[39] WHATWG. Html living standard. https://html.spec.whatwg.org/, 2024. Accedido el 3 de agosto de 2025.

[40] Yeoman Team. Yo code — generator for visual studio code extensions. https://vscode-docs.readthedocs.io/en/stable/tools/yocode/, 2024. Accedido: 2025-09-06.