Universidad
Rey Juan Carlos

ESCUELA DE INGENIERÍA DE FUENLABRADA

INGENIERÍA TELEMÁTICA

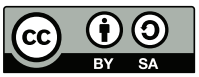
TRABAJO FIN DE GRADO

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

mi familia / mi abuelo / mi abuela

II

Agradecimientos

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

Resumen

Code-XR es una propuesta pensada para cambiar la forma en que los desarrolladores analizan y entienden su código en tiempo real. Se trata de un plugin para Visual Studio Code, que permite visualizar a tiempo real métricas avanzadas del código como la complejidad ciclomática entre otras metricas como media de parámetros por función, número de lineas 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 aceptado que, a medida que los proyectos de software crecen, entender su arquitectura interna y evaluar su calidad se vuelve cada vez más desafiante para los desarrolladores [13]. Las herramientas actuales de los entornos de desarrollo integrados (IDEs), como tablas, gráficos lineales o mapas de calor en 2D, ofrecen cierto apoyo al mostrar métricas estáticas, pero rara vez logran reflejar con claridad cómo se relacionan los distintos módulos del código. Esta limitación dificulta que los desarrolladores identifiquen, de forma rápida y precisa, problemas habituales de diseño, como la complejidad creciente o la duplicación de lógica [5, 16].

Ante estas limitaciones, la comunidad científica ha comenzado a explorar técnicas inmersivas como la Realidad Virtual (VR) y la Realidad Aumentada (AR), enmarcadas bajo el concepto más general de Realidad Extendida (XR). Estas tecnologías permiten representar el software en espacios tridimensionales, facilitando la interpretación espacial de artefactos y métricas [9]. Diversos trabajos han demostrado que el uso de entornos espaciales y la interacción corporal pueden mejorar tanto la comprensión como el nivel de compromiso del usuario [6].

Uno de los enfoques más influyentes ha sido el de la ciudad del software, introducido por herramientas como CodeCity [36] 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 o complejidad ciclomática) en atributos visuales (altura, base o color), permitiendo una percepción más inmediata de las zonas problemáticas del sistema.

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

1.2. Objetivo y aportación de Code-XR

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

A diferencia de herramientas anteriores, que operan como sistemas independientes o requieren dashboards externos, Code-XR apuesta por una integración directa en el entorno de desarrollo (concretamente Visual Studio Code), permitiendo que los desarrolladores visualicen el impacto de sus cambios de forma continua mientras programan, sin necesidad de abandonar su flujo de trabajo [36, 12].

Su propuesta se basa en mapear propiedades estáticas del código como el número de líneas, parámetros o la complejidad ciclomática a atributos visuales tridimensionales en una ciudad virtual: la altura de un edificio representa el tamaño, la base refleja la cantidad de parámetros, y el color la complejidad [36]. Esta metáfora permite identificar de forma intuitiva patrones como zonas de alta complejidad, crecimiento irregular o duplicación de lógica, incluso en proyectos grandes.

Además, Code-XR introduce una segunda aportación relevante: el soporte opcional para la visualización inmersiva de la estructura DOM en aplicaciones web. Esto amplía el campo de aplicación de la herramienta más allá del análisis de código fuente, permitiendo explorar jerarquías HTML complejas en un espacio XR tridimensional [22].

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

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

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

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

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

El sistema ofrece tres modos de visualización principales:

- **LivePanel**: muestra las métricas extraídas en una interfaz 2D tradicional tipo panel. Es útil para obtener una vista rápida, numérica o textual de los datos mientras se codifica.
- **XR Mode**: presenta una ciudad 3D donde cada función se representa como un edificio, con altura, base y color según sus métricas. Este modo es accesible tanto desde el navegador (modo escritorio) como desde un dispositivo de realidad aumentada (AR), conectándose al mismo servidor local [36, 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 [22].

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 [23].

1.4. Estructura de la memoria

Esta memoria se estructura de la siguiente manera:

- **Capítulo 1, Introducción:** Se presenta el contexto general del trabajo, las motivaciones que lo impulsan, el objetivo principal del proyecto y la aportación que supone Code-XR en el ámbito de la visualización de métricas de software. Además, se ofrece una descripción general de la herramienta y una guía sobre la estructura del documento.

- **Capítulo 2, Objetivos y planificación:** Se detallan los objetivos generales y específicos del proyecto, así como la planificación temporal seguida para su desarrollo, incluyendo una visión de los hitos principales y la metodología empleada.

- **Capítulo 3, Estado del arte:** Se analiza el estado del arte en visualización de software, con especial énfasis en técnicas inmersivas y herramientas previas relevantes. Se revisan también los enfoques más comunes para la representación de métricas estructurales en entornos de desarrollo.

- **Capítulo 4, Diseño e implementación:** Se describe el diseño e implementación de Code-XR, incluyendo su arquitectura, los componentes principales de la herramienta, las decisiones técnicas adoptadas y los diferentes modos de visualización que ofrece.

- **Capítulo 5, Validación del prototipo:** Se exponen los experimentos realizados para validar el prototipo, incluyendo los escenarios de prueba, los criterios de evaluación empleados y la documentación del comportamiento observado durante su uso en proyectos reales.

- **Capítulo 6, Resultados:** Se recogen los resultados obtenidos tras la evaluación, valorando en qué medida se han cumplido los objetivos propuestos y analizando las ventajas y limitaciones del enfoque adoptado.

- **Capítulo 7, Conclusiones y líneas futuras:** Se presentan las conclusiones generales del trabajo, se reflexiona sobre el aprendizaje obtenido durante el desarrollo del proyecto y se plantean posibles líneas de mejora o extensión futura de Code-XR.

- **Apéndices y bibliografía:** La memoria se complementa con un apéndice que incluye un manual de uso de la herramienta, así como la bibliografía consultada a lo largo

Capítulo 2

Objetivos

2.1. Objetivo general

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

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

2.2. Objetivos específicos

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

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

- **Implementar un sistema de análisis estático de código:** Desarrollar un conjunto de scripts en Python que, utilizando la librería Lizard [32], extraigan métricas relevantes como líneas de código, número de parámetros y complejidad ciclomática a partir de archivos individuales o directorios completos.

- **Desarrollar una visualización 3D inmersiva con BabiaXR:** Construir una interfaz gráfica basada en BabiaXR, A-Frame y WebXR que represente los elementos del código como edificios dentro de una ciudad tridimensional. Esta interfaz debe soportar interacción en tiempo real, diferentes estilos visuales y ser accesible desde navegador o dispositivos XR.

- **Establecer comunicación en tiempo real mediante servidor local:** Lanzar un servidor HTTPS que gestione las peticiones de análisis y envíe los datos al cliente visual usando Server-Sent Events (SSE), permitiendo una actualización continua del modelo 3D sin necesidad de recargar la vista manualmente.

- **Integrar Code-XR dentro del entorno Visual Studio Code:** Utilizar la API oficial de VS Code para proporcionar comandos, menús contextuales y vistas personalizadas que permitan al usuario ejecutar el análisis y acceder a las visualizaciones sin abandonar el editor.

- **Ofrecer múltiples modos de visualización complementarios:** Desarrollar tres formas de explorar las métricas extraídas:

  - **LivePanel**, un panel 2D con métricas detalladas.
  - **XR Mode**, escenarios 3D accesibles vía navegador o dispositivos XR a través de localhost.
  - **DOM Mode**, una vista específica para estructuras HTML en entornos web.

- **Validar la herramienta en proyectos reales de software:** Aplicar Code-XR a repositorios reales con distintos lenguajes y estructuras, con el fin de comprobar su utilidad práctica, detectar patrones complejos y obtener retroalimentación sobre su usabilidad.

2.3. Planificación temporal

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

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

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

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

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

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

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

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

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

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

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

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

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

Finalmente, el 29 de julio publiqué la versión 1.0.0 de Code-XR y creé la página web oficial del proyecto [15], donde se incluyen vídeos demostrativos, una guía rápida de

instalación y documentación completa sobre el funcionamiento del plugin.

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

TODO: Diagrama de Gantt de Code-XR

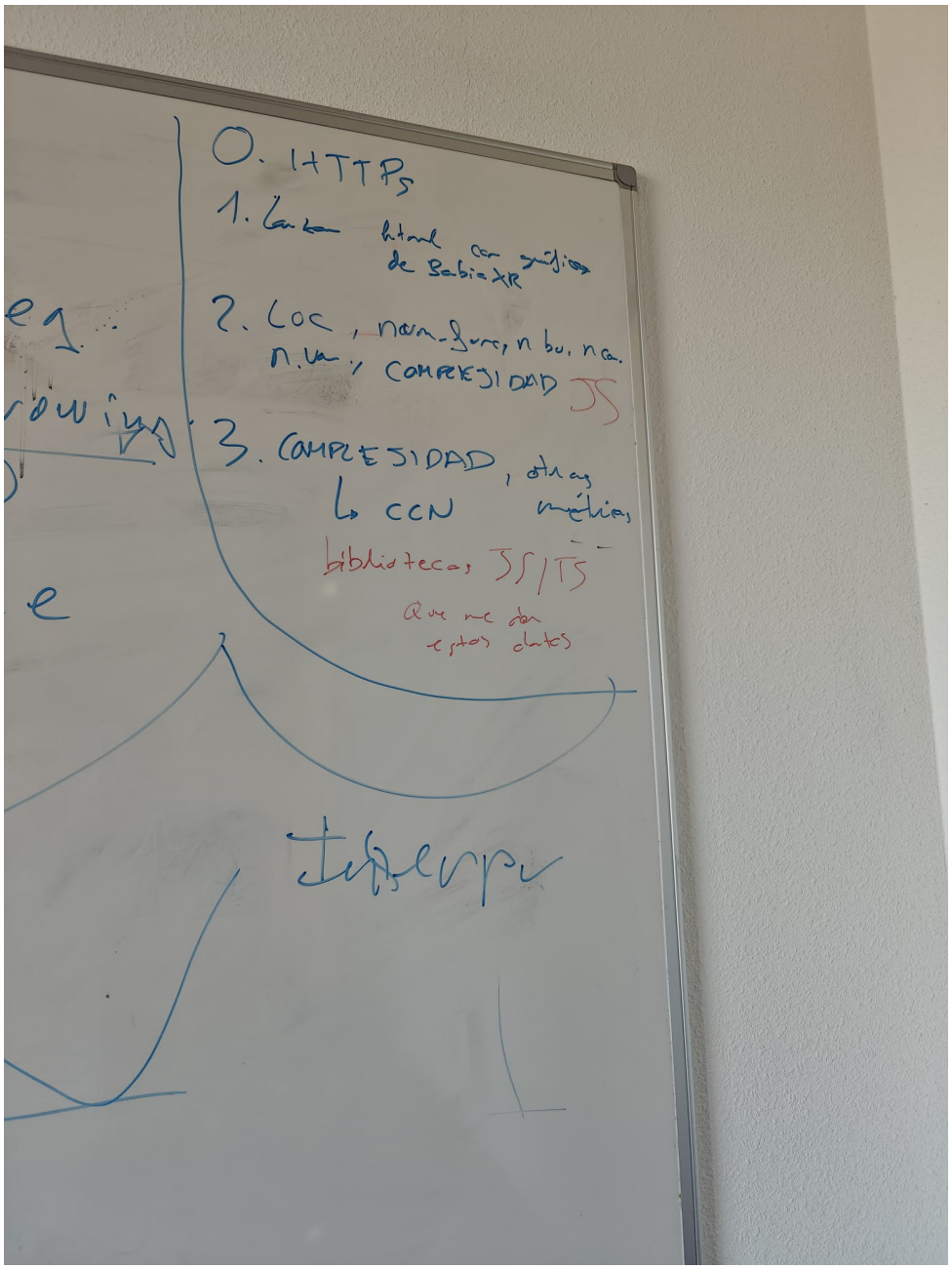


Figura 2.1: Imagen de la reunión del 12 de marzo de 2025, donde se definieron los puntos principales del TFG.

Capítulo 3

Estado del arte

3.1. Visual Studio Code y su sistema de extensiones

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

Una de las características más potentes de VS Code es su sistema de extensiones, que permite ampliar las funcionalidades del editor mediante plugins desarrollados en tecnologías web, principalmente TypeScript, JavaScript, HTML y CSS. Estas extensiones pueden interactuar directamente con el entorno mediante la API oficial de VS Code [19], la cual expone funcionalidades como el acceso al árbol de archivos, la edición del contenido de los documentos, la creación de paneles personalizados, la ejecución de comandos o la escucha de eventos internos del editor.

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

Gracias al uso de la API de extensiones [19], se han podido registrar comandos personalizados, interceptar eventos como la apertura o modificación de archivos y lanzar servidores locales para comunicar el análisis del código con las interfaces XR que forman parte del sistema. El diseño modular de VS Code y su modelo basado en eventos lo convierten en un entorno especialmente adecuado para prototipar herramientas como Code-XR, que requieren integración en tiempo real con el proceso de edición de código.

3.2. BabiaXR

BabiaXR [23] es una plataforma de visualización tridimensional e inmersiva desarrollada por investigadores del Grupo de Sistemas y Comunicaciones (GSyC) de la Universidad Rey Juan Carlos (URJC). Está diseñada para facilitar la creación de escenas interactivas en realidad extendida (XR), incluyendo realidad virtual (VR)y aumentada (AR), utilizando tecnologías web abiertas y accesibles como A-Frame, Three.js y WebXR.

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

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

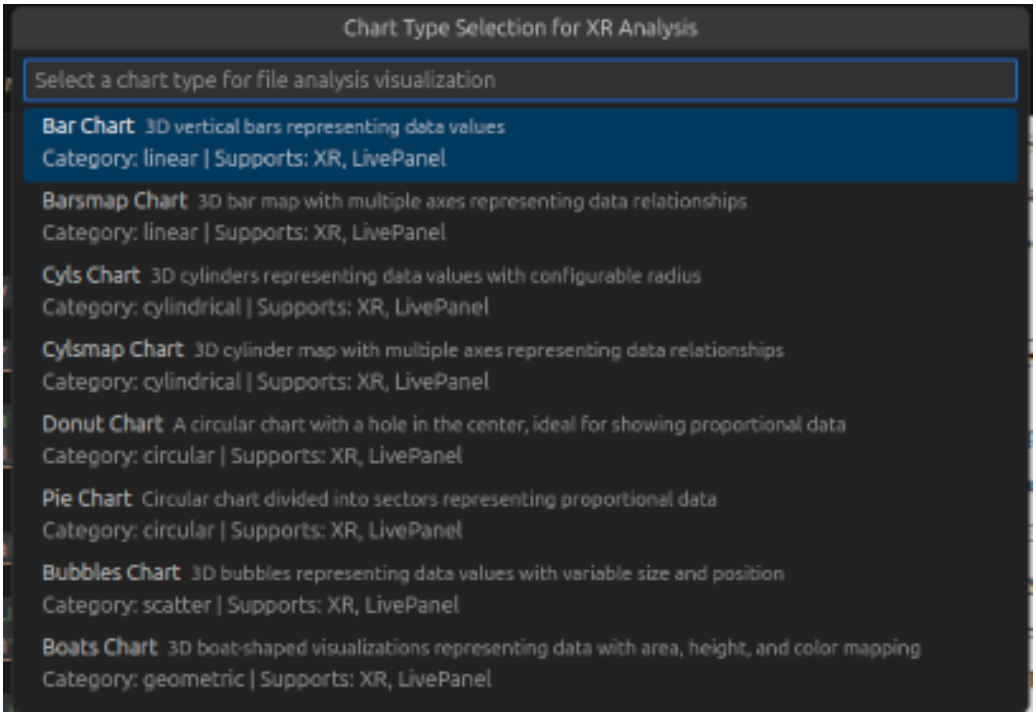


Figura 3.1: Ejemplo de los diferentes gráficos de BabiaXR utilizados en Code-XR para la visualización de métricas.

3.3. A-Frame

A-Frame es un framework de código abierto desarrollado inicialmente por Mozilla, orientado a la creación de experiencias inmersivas de realidad virtual y aumentada directamente desde el navegador [24]. Su principal objetivo es reducir la complejidad del desarrollo de escenas 3D, ofreciendo una sintaxis declarativa basada en HTML que permite a los desarrolladores definir entornos tridimensionales mediante etiquetas comprensibles y reutilizables.

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

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

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

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

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

Además, se han utilizado extensiones adicionales como:

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

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

3.4. WebXR y Three.js

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

(AR) desde navegadores web modernos [35]. Esta API permite detectar dispositivos compatibles —como gafas de realidad virtual, controladores de movimiento o sensores de orientación— y renderizar contenido inmersivo directamente en el navegador, sin necesidad de plugins o instalaciones adicionales.

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

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

Por su parte, Three.js es una biblioteca JavaScript de alto nivel para el renderizado de gráficos 3D en WebGL [7]. Proporciona una interfaz sencilla para trabajar con cámaras, luces, materiales, geometrías y animaciones, y ha sido adoptada como base por muchos frameworks más accesibles como A-Frame. En otras palabras, Three.js actúa como el motor gráfico real que renderiza las escenas construidas en A-Frame, traduciéndolas a instrucciones de bajo nivel ejecutadas en la GPU mediante WebGL.

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

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

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

3.5. HTML

HTML (HyperText Markup Language) es el lenguaje de marcado fundamental para la estructuración de contenido en la web [37]. En el contexto de Code-XR, HTML actúa como la base común sobre la que se construyen todas las interfaces visuales de la herramienta, independientemente del modo de análisis utilizado: ya sea LivePanel, XR Mode o VisualizeDOM.

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

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

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

3.6. TypeScript y JavaScript

TypeScript es un lenguaje de programación desarrollado por Microsoft que extiende JavaScript con tipado estático opcional y características orientadas a objetos [18]. Se compila a JavaScript y es especialmente útil para desarrollar aplicaciones a gran escala. En el caso de Code-XR, TypeScript actúa como el motor principal del plugin para Visual Studio Code, gestionando tanto la lógica de control como la interacción con la API del editor.

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

Por su parte, JavaScript ha sido el lenguaje utilizado en la parte cliente (frontend) de la herramienta, desempeñando un papel fundamental en la actualización dinámica de las visualizaciones [25]. En particular, es el encargado de implementar el sistema de comunicación en tiempo real mediante Server-Sent Events (SSE). Cada vez que se detecta un cambio en el código fuente, el backend (escrito en TypeScript) actualiza el análisis y envía los datos mediante SSE, que son recibidos en el cliente JavaScript para actualizar la vista correspondiente.

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

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

3.7. CSS

CSS (Cascading Style Sheets) es el lenguaje estándar utilizado para definir la presentación visual de documentos HTML [34]. Permite aplicar estilos a elementos web como colores, fuentes, márgenes, disposición de contenido o animaciones. En el contexto de Code-XR, CSS se utiliza exclusivamente para dar estilo a las interfaces generadas en el modo LivePanel, es decir, en las visualizaciones bidimensionales de métricas de código.

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

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

3.8. Python y Lizard

Python es un lenguaje de programación de alto nivel, ampliamente utilizado en tareas de automatización, análisis de datos y procesamiento de texto [30]. En el marco de este proyecto, Python ha sido la tecnología seleccionada para realizar el análisis estático del código fuente, es decir, para extraer métricas estructurales que luego son representadas visualmente en los distintos modos de Code-XR.

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

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

Para calcular métricas como la complejidad ciclomática (CCN), el número de líneas de código (LOC) o el número de funciones o métodos, se ha utilizado la herramienta Lizard [32], una biblioteca de análisis estático especializada y compatible con múltiples lenguajes de programación. Lizard permite procesar archivos fuente y devolver una representación estructurada de sus componentes, lo que ha facilitado la integración directa con el sistema de visualización de Code-XR.

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

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

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

3.9. Gafas de Realidad Virtual: Meta Quest 3

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

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

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

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

3.10. Apoyo de herramientas de inteligencia artificial

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

En particular, ChatGPT [28] ha sido una herramienta de referencia para comprender el funcionamiento de bibliotecas como Lizard (usada para el análisis estático de código) o A-Frame (empleada en la construcción de escenas XR). También ha servido como apoyo a la hora de encontrar enfoques para resolver problemas concretos, mejorar la organización de ciertas secciones del código o redactar documentación técnica con un enfoque más claro y estructurado.

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

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

Capítulo 4

Diseño e implementación

4.1. Arquitectura general

En esta sección se presenta la arquitectura general de Code-XR, explicando sus componentes principales, los canales de comunicación entre ellos y las decisiones tecnológicas que sustentan su diseño. El objetivo es ofrecer una visión global suficiente para entender cómo fluye la información desde el código fuente hasta las visualizaciones en 2D (LivePanel) y XR.

En los siguientes apartados de este capítulo se explicará cómo se desarrollaron las distintas partes del plugin, los pasos o sprints seguidos durante el desarrollo y cómo se estableció la comunicación entre los componentes, incluyendo detalles sobre protocolos, formatos de intercambio y la orquestación entre TypeScript, Python y las vistas cliente.

4.1.1. Componentes y responsabilidades

- **VS Code Extension (TypeScript):** registro de comandos, UI en el editor, gestión de procesos (lanzamiento de servidores HTTP/HTTPS, análisis con Python mediante entorno virtual, etc...), y almacenamiento de configuraciones. En conclusión, TypeScript es el lenguaje que maneja toda la lógica del plugin a través de la API oficial de Visual Studio Code [19].

- **Python metrics engine:** ejecución de scripts en Python para el análisis estático de ficheros y directorios. La mayor parte de las métricas se extraen usando la librería Lizard [32], mientras que algunas métricas complementarias se calculan mediante código Python

propio. El resultado se normaliza y se exporta en formato JSON para ser entendido por los componentes de BabiaXR [23].

- **JSON Storage / Manager:** formato intermedio y almacenamiento temporal de resultados (ficheros .json o memoria). Este formato se utiliza tanto para las configuraciones del plugin —por ejemplo, parámetros a representar y el tipo de gráfico seleccionado— como para la salida de los análisis ejecutados por Python (las métricas se exportan en JSON). Las configuraciones persistentes del plugin se guardan usando la API de VS Code en globalState (ExtensionContext.globalState), de modo que se mantienen entre sesiones del editor; por su parte, los resultados de análisis y datos asociados al proyecto se almacenan en workspaceState (ExtensionContext.workspaceState), que queda ligado al workspace abierto y no se comparte entre workspaces. Para más detalles sobre ambos mecanismos de almacenamiento véase la documentación oficial de VS Code [17].

- **Realtime API (SSE):** canal unidireccional basado en Server-Sent Events (SSE) para enviar actualizaciones incrementales de métricas a los clientes (LivePanel, XR). En el cliente se implementa mediante la API nativa 'EventSource' de JavaScript, y en el servidor mediante respuestas HTTP con content-type `text/event-stream` [26].

  Aunque existen librerías como express-sse [27] que facilitan la implementación de SSE en aplicaciones Node.js, se optó por una implementación nativa para mantener un control granular sobre los eventos enviados y reducir las dependencias externas del plugin. El sistema funciona de la siguiente manera: cuando un fichero observado cambia, la extensión detecta la modificación, lanza un re-análisis (entorno virtual de Python), normaliza y actualiza el JSON correspondiente en workspaceState, y emite eventos SSE específicos como `analysis-updated` o `dataRefresh`. Los clientes suscritos (LivePanel, páginas XR) escuchan estos eventos mediante `EventSource('/events')`, reciben las notificaciones y actualizan automáticamente las visualizaciones sin recargar la página.

- **Servidores de visualización (HTML/CSS/JS):** conjunto de plantillas web servidas por servidores locales HTTP/HTTPS para la visualización de métricas. Incluye dos modos principales:

  - **LivePanel**: webview embebida en VS Code para visualización 2D con tablas, gráficos

y métricas detalladas.

- **XR Pages**: escenas 3D basadas en A-Frame y BabiaXR para visualización inmersiva en navegador o dispositivos VR.

Ambos modos comparten la misma infraestructura de servidor y comunicación SSE. Para el acceso desde dispositivos XR (como Meta Quest 3), el servidor debe configurarse en modo HTTPS con certificados autofirmados generados mediante OpenSSL [29], ya que los navegadores en dispositivos VR requieren conexiones seguras para acceder a las APIs de WebXR.

- **Sistema de plantillas dinámicas:** generador automático de plantillas HTML específicas para cada tipo de análisis y visualización. Incluye procesamiento de placeholders, validación de dimensiones de datos, y adaptadores para diferentes gráficos BabiaXR según el tipo de métricas extraídas.

- **File watcher y detección de cambios:** monitorización automática de archivos para reanálisis en tiempo real. Utiliza la API de VS Code para detectar modificaciones en los ficheros bajo análisis y triggear actualizaciones automáticas del JSON y las vistas asociadas.

- **Registry de servidores activos:** sistema de registro y gestión del ciclo de vida de servidores HTTP/HTTPS lanzados por el plugin. Permite controlar múltiples instancias simultáneas, gestionar puertos dinámicos y evitar conflictos entre diferentes análisis en ejecución.

4.1.2. Tipos de análisis

A continuación se presenta una tabla simplificada con los tipos fundamentales de análisis soportados por Code-XR, su alcance y si admiten la opción *deep* (recursión por subdirectorios) o la posibilidad de aplicarse al workspace completo (análisis de proyecto).

Tipo de análisis	**Alcance**	**Deep? / Proyecto?**
Fichero — LivePanel	Análisis puntual de un único fichero; visualización 2D en el panel integrado.	No / No
Fichero — XR	Análisis puntual de un único fichero; representación inmersiva en A-Frame/BabiaXR.	No / No
Directorio — LivePanel	Análisis de un directorio; resumen por archivo y agregados por carpeta.	Sí / Sí
Directorio — XR	Representación 3D del contenido de un directorio (escena A-Frame/BabiaXR).	Sí / Sí
DOM Mode (especial)	Análisis de la estructura DOM de un fichero HTML; visualización 3D específica del DOM.	No / No

Figura 4.1: Tipos fundamentales de análisis, alcance y soporte de *deep*/*proyecto*.

Para mayor claridad, a continuación se muestra la relación entre los comandos expuestos en la interfaz (menú contextual / editor) y el tipo de análisis que lanzan.

Comando (UI)	Identificador interno / Tipo
CodeXR: Analyze File (LivePanel)	`newCodeAnalysis.analyzeFile` — Fichero / LivePanel
CodeXR: Analyze File (XR)	`newCodeAnalysis.analyzeFileXR` — Fichero / XR
CodeXR: Analyze Directory (LivePanel)	`newCodeAnalysis.analyzeDirectory` — Directorio / LivePanel (No deep)
CodeXR: Analyze Directory (LivePanel Deep)	`newCodeAnalysis.analyzeDirectoryDeep` — Directorio / LivePanel (Deep)
CodeXR: Analyze Directory (XR)	`newCodeAnalysis.analyzeDirectoryXR` — Directorio / XR (No deep)
CodeXR: Analyze Directory (XR Deep)	`newCodeAnalysis.analyzeDirectoryXRDeep` — Directorio / XR (Deep)
CodeXR: Analyze Project (LivePanel)	`newCodeAnalysis.analyzeProject` — Proyecto / LivePanel (raíz del workspace)
CodeXR: Analyze Project (LivePanel Deep)	`newCodeAnalysis.analyzeProjectDeep` — Proyecto / LivePanel (Deep, raíz del workspace)
CodeXR: Analyze Project (XR)	`newCodeAnalysis.analyzeProjectXR` — Proyecto / XR (raíz del workspace)
CodeXR: Analyze Project (XR Deep)	`newCodeAnalysis.analyzeProjectXRDeep` — Proyecto / XR (Deep, raíz del workspace)
CodeXR: Visualize DOM	`newCodeAnalysis.visualizeDOM` — DOM Mode

Figura 4.2: Mapeo entre comandos de la UI y tipos de análisis (incluye variantes de proyecto).

Notas:

- Los comandos de “Proyecto” operan reutilizando la misma lógica interna que los comandos de “Directorio”; la única diferencia es que, al invocarse, toman la raíz del workspace como directorio a analizar.

- La forma de invocar estos comandos desde la interfaz es mediante el menú contextual (click derecho) en el explorador de archivos o en el editor. La asignación de entradas de menú, reglas de visibilidad y los identificadores de comando están declarados en el fichero `package.json` del plugin; en un apartado posterior se detalla la estructura y funcionamiento de `package.json` en extensiones de VS Code. Otra forma también de invocarlos es a través de un acceso rápido integrado en la interfaz del plugin, que permite seleccionar ficheros y directorios mediante las vistas "Files by Language" y "Project Structure". Esto se explicará con más detalle en la sección dedicada a la interfaz del plugin.

- "Deep" indica análisis recursivo sobre subdirectorios; su uso incrementa la cantidad de datos y las entidades en la escena XR, por lo que puede requerir optimizaciones de rendimiento (batching, análisis incremental, reducción de nivel de detalle).

4.2. Flujo de datos / Secuencia de ejecución

A continuación se explica, de forma general y sin referencias a rutas concretas, la secuencia básica que sigue un análisis en Code-XR desde la petición del usuario hasta la actualización de la visualización. Esta descripción está pensada para ofrecer una visión global del flujo de trabajo.

1. **Petición del análisis.** El usuario solicita un análisis mediante la interfaz gráfica (clic derecho), a través del menú rápido del plugin (*Files by Language* o *Project Structure*) o ejecutando el comando desde la paleta de comandos (F1). La petición especifica el tipo de análisis deseado y el objetivo (ruta al fichero o al directorio a analizar).

2. **Registro de la sesión de análisis.** Se crea una sesión de análisis con un identificador único (nonce) y se registran los datos necesarios para ejecutar y rastrear el análisis: la ruta de almacenamiento en el directorio `workspaceStorage` (ver [17]), la lista de ficheros involucrados, el tipo de análisis, el estado actual de la sesión y los parámetros de configuración. Tras crear la sesión, el flujo continúa con la obtención de los ficheros necesarios para el análisis.

3. **Obtención de los ficheros necesarios.** Según el tipo de análisis, el motor determina los ficheros y parámetros necesarios para ejecutar y presentar el resultado: las plantillas

HTML/CSS/JS que se usarán en la vista, y los argumentos con los que ejecutar el script principal `main.py` que realizará el análisis para generar el fichero `data.json` con las métricas. En la práctica se prepara un paquete compuesto por la plantilla HTML, los recursos CSS/JS y el fichero de datos; adicionalmente se copia un `favicon.ico` para la interfaz (no relevante para la lógica del análisis). Todos estos ficheros se guardan en la ruta de almacenamiento creada para la sesión dentro del directorio `workspaceStorage`. El parseado de la plantilla HTML cambia según el tipo de análisis solicitado (p. ej. LivePanel vs XR); el detalle del parseado se describe en la sección dedicada a explicar cómo funciona el análisis de Code-XR.

4. **Persistencia temporal de la sesión.** Una vez generado el paquete con todos los datos necesarios, éstos se guardan en la ruta indicada para la sesión. Esta persistencia facilita el reanálisis y el arranque del servidor de visualización. Los ficheros resultantes de los análisis realizados durante la sesión se eliminan de `workspaceStorage` al cerrar VS Code, evitando así dejar *analysis zombies* que ocupen espacio innecesario.

5. **Activación de watchers con debounce.** Se registran observadores (watchers) sobre los ficheros implicados para detectar cambios. Los watchers aplican un *debounce time*: un intervalo configurable que sirve para agrupar eventos de cambio que ocurren en rápida sucesión (por ejemplo, mientras un desarrollador escribe). En ausencia de debounce, cada modificación podría disparar un nuevo análisis y generar sobrecarga; con debounce, el sistema espera hasta que no se producen más cambios durante el intervalo configurado y entonces ejecuta el análisis, evitando reanálisis innecesarios. Si el usuario cambia dinámicamente el valor del *debounce time* durante una sesión, los watchers se reconfiguran para emplear el nuevo intervalo inmediatamente, afectando a las detecciones posteriores.

6. **Lanzamiento del servidor de visualización y canal SSE.** Se inicia un servidor local que sirve los ficheros HTML/CSS/JS preparados para la sesión como ‘localhost‘ (o ‘https://localhost‘ si el usuario tiene configurado HTTPS). La forma concreta de arrancar el servidor (puerto, HTTP/HTTPS, certificados, etc...) depende de la configuración del usuario; en cualquier caso el servidor expone además un canal de eventos en tiempo real mediante Server-Sent Events (SSE) que se utiliza para notificar a las vistas cuando hay resultados nuevos tras un re-análisis.

7. **Detección de cambios y re-análisis incremental.** Cuando un watcher detecta un cambio en un fichero, se siguen estos pasos:
   a) Identificar la sesión de análisis asociada al fichero modificado.
   b) Re-analizar únicamente los ficheros afectados (análisis incremental) y actualizar el JSON/metadata de la sesión.
   c) Persistir los resultados en la ruta de la sesión para que el servidor los sirva inmediatamente.
   d) Emitir un evento por el canal SSE (`analysis-updated`); las vistas suscritas reciben la notificación y actualizan la visualización en tiempo real.

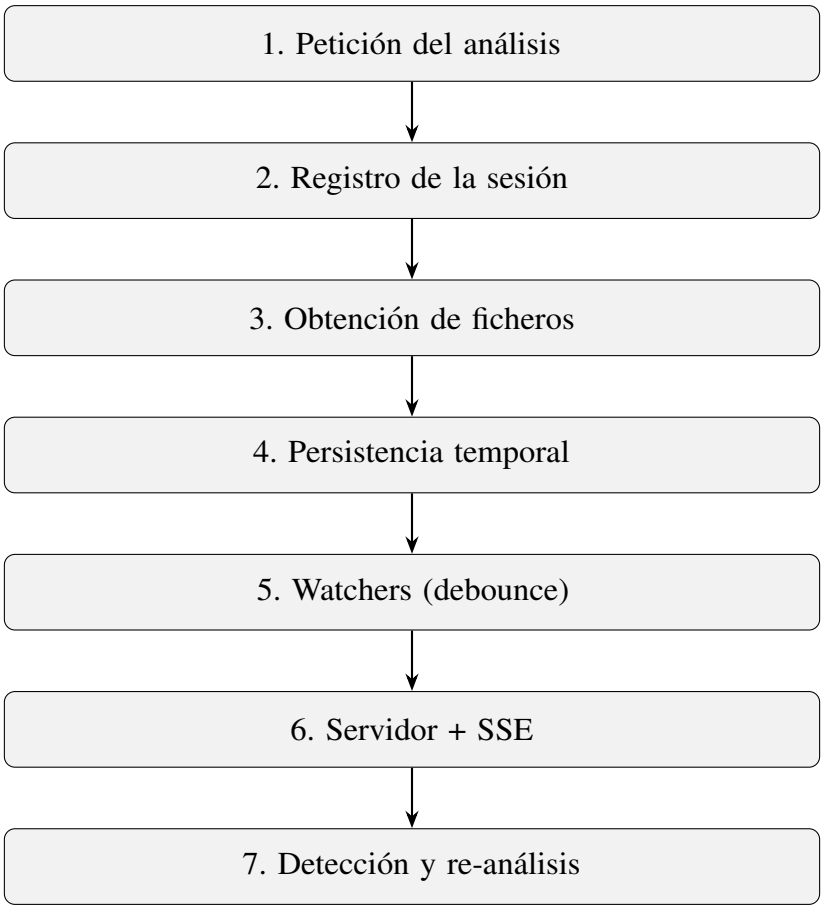


Figura 4.3: Secuencia vertical simplificada del flujo de un análisis.

En el siguiente diagrama se ofrece una visión abstracta y lineal del proceso. En la práctica, varias sesiones pueden coexistir, teniendo para un fichero mas de un watcher activo (p. ej. un análisis de directorio y otro de proyecto que incluyan el mismo fichero). El sistema gestiona

estas situaciones mediante el registro de sesiones y watchers, asegurando que cada cambio se propaga correctamente a todas las vistas afectadas.

4.3. Resumen cronológico de versiones

A continuación se presenta un resumen cronológico compacto de las versiones principales del plugin (desde la 0.0.1 hasta la 1.0.0). La tabla está ordenada empezando por la versión inicial (0.0.1) y finalizando en la versión estable (1.0.0). En la subsección siguiente se describirán con más detalle los sprints/hitos relacionados con estas versiones.

Versión	Fecha (aprox.)	Resumen / hito principal
0.0.1	2025-03-11	Primer prototipo: servidor HTTP que servía un HTML A-Frame predefinido (prueba de concepto XR) y comandos básicos para lanzar la visualización.
0.0.2	2025-03-20	Primer análisis básico en modo LivePanel para ficheros JavaScript/TypeScript; ya se comenzó a controlar la infraestructura XR (selección del HTML a servir y primeras comprobaciones de HTTPS), pero las visualizaciones principales seguían siendo 2D (LivePanel).
0.0.3	2025-04-11	Introducción de auto-reanalysis, selección inteligente de ejes, integración SSE, watchers por fichero y primer soporte de análisis estático (LOC, CCN, funciones).
0.0.4	2025-04-27	Soporte multi-lenguaje ampliado y sistema configurable de debounce para auto-análisis; mejoras en XR y análisis múltiple simultáneo.
0.0.5	2025-04-29	Nuevas visualizaciones (babia-boats), mapeo de parámetros a dimensiones visuales y mejoras en plantillas y resolución de rutas.
0.0.6	2025-04-29	Publicación inicial en marketplace: estabilización de servidores, UX y preparación para distribución pública.
0.0.7	2025-06-01	Mejora del live reload y experiencia XR: SSE robusto, controles VR/AR mejorados y configuración avanzada de entornos.
0.0.8	2025-07-03	Revisión mayor del motor de análisis: soporte Lizard ampliado, Visualize DOM, nuevas visualizaciones (bubble chart) y optimizaciones de watcher/caché.
0.0.9	2025-07-27	Re-arquitecturación del motor de análisis: análisis de directorios completos, soporte deep y mejoras de rendimiento/-fiabilidad.
1.0.0	2025-07-28	Versión estable: sistema de auto-análisis configurable, gestión avanzada de sesiones con prevención de duplicados, filtrado inteligente de ficheros HTML, optimización de configuración y watchers, y publicación de la web oficial de documentación (https://amontesl.github.io/code-xr-docs/).

Cuadro 4.1: Resumen cronológico de versiones — hitos principales (orden: 0.0.1 → 1.0.0).

Nota: los detalles en esta tabla se han extraído y condensado desde el fichero `CHANGELOG.md`. Hay registros públicos en `CHANGELOG.md` a partir de la versión 0.0.3; el plugin estuvo publicado en el marketplace desde la versión 0.0.6. Las fechas aproximadas indican momentos representativos del desarrollo y se usan para situar cronológicamente los hitos.

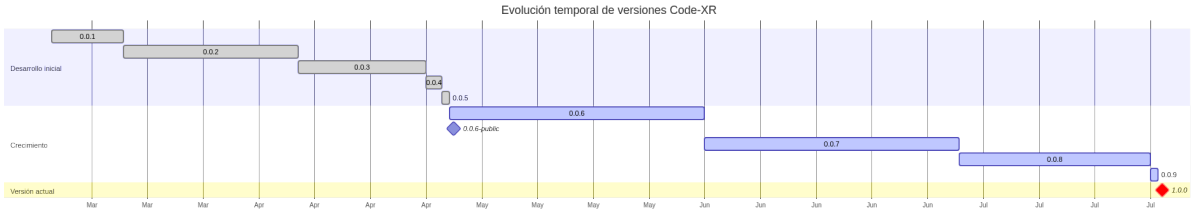


Figura 4.4: Evolución temporal resumida de las versiones principales del proyecto.

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.

4.4. Primeros pasos

Esta sección recoge los conceptos básicos y decisiones iniciales adquiridos durante la creación de la extensión para VS Code. El propio editor ofrece un tutorial introductorio para desarrollar una primera extensión[20], el cual se siguió en paralelo con recursos en vídeo, como el tutorial oficial de VS Code en YouTube[8]. A partir de estos materiales, y utilizando el generador oficial de extensiones (“Yo Code”)[3], se establecieron las bases del proyecto y se tomaron las primeras decisiones técnicas que marcaron su desarrollo inicial.

4.4.1. Generador *Yo Code*

El generador oficial de extensiones para VS Code, denominado “Yo Code”, guía al desarrollador mediante una serie de preguntas que definen la plantilla inicial del plugin: identificador y nombre de la extensión, descripción, publicador, lenguaje de desarrollo (TypeScript o JavaScript), inicialización opcional de un repositorio Git, gestor de paquetes y el tipo de empaquetado (opciones típicas: *unbundled*, Webpack, Esbuild). Estas decisiones iniciales configuran la estructura base del proyecto y los scripts para compilar, empaquetar y desplegar la extensión.

4.4.2. Elección de lenguaje

Se optó por **TypeScript**, dado que ofrece tipado estático que reduce errores, integración directa con las definiciones de la API de VS Code y mayor mantenibilidad en proyectos de mediana y gran escala[18]. Este lenguaje permitió a Code-XR evolucionar de forma segura al crecer en número de ficheros y complejidad.

4.4.3. Bundlers

El proceso de empaquetado de una extensión de VS Code puede realizarse de distintas formas, dependiendo del flujo de desarrollo y de los requisitos de distribución. La documentación oficial recomienda evaluar diferentes bundlers según las necesidades del proyecto[33]. Entre las opciones más habituales se incluyen:

- **Unbundled**: sin bundler, útil para prototipado rápido pero obliga a distribuir dependencias manualmente.
- **Webpack**: bundler clásico y ampliamente usado; genera un único paquete y minimiza dependencias externas[2].
- **Esbuild**: bundler moderno, extremadamente rápido y con tiempos de compilación reducidos[1].

En Code-XR se comenzó con la plantilla *unbundled* para acelerar el prototipado y validar la integración con A-Frame y BabiaXR. Posteriormente se migró a un flujo con **Webpack**, que sigue siendo el bundler activo en la actualidad. El cambio no solo aportó mayor fiabilidad en la distribución, sino que también resolvió limitaciones prácticas: en versiones recientes, el empaquetado a formato `.vsix` generaba advertencias de `npm` por tamaño excesivo. Con Webpack, dichas advertencias desaparecieron y los tiempos de compilación se redujeron de forma notable.

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

El fichero `package.json` es el manifiesto central de toda extensión de VS Code. Contiene los metadatos que identifican la extensión en el Marketplace (nombre, versión, publicador, categorías, icono y repositorio), además de la compatibilidad con el motor de VS Code y la lista de ficheros incluidos en la distribución.

También controla el proceso de compilación y empaquetado mediante la sección `scripts`, que en Code-XR se apoya en Webpack para generar un bundle reproducible siguiendo las recomendaciones oficiales[33].

Otra sección clave es `contributes`, donde se definen los elementos que la extensión añade a la interfaz: más de 40 comandos, la vista lateral `codexrTree`, y menús contextuales en el explorador, el editor y las vistas de elementos. Esta configuración conecta la extensión con el usuario final y organiza la navegación.

En cuanto a `activationEvents`, Code-XR declara una lista vacía, lo que implica activación ansiosa al abrir VS Code. Aunque lo recomendable es activar bajo demanda (`onCommand`, `onView`, `onLanguage`), esta decisión garantizó que todos los menús y vistas estuviesen disponibles desde el inicio.

Finalmente, la **iconografía** de la extensión se construyó principalmente con *Codicons*, la librería oficial de VS Code[21]. Se emplearon iconos de *Devicon*[10] para representar ficheros en vistas como *Files per Language* y *Project Structure*, mientras que el logotipo de Code-XR se generó de manera personalizada mediante técnicas de inteligencia artificial.

4.4.5. Archivo `extension.ts`

El archivo `extension.ts` implementa la lógica declarada en el manifiesto. Es el punto de entrada de la extensión y define dos funciones fundamentales:

- `activate`: se ejecuta al activar la extensión y orquesta la inicialización de servicios (gestión de servidores, restauración de configuraciones, limpieza automática de análisis obsoletos), el registro de la vista modular (`codexrTree`), la asociación de comandos y la sincronización del estado de visualización de datos.

- `deactivate`: se invoca al desactivar la extensión y garantiza la liberación ordenada de recursos, como servidores activos, gestores SSE, mapeos de ficheros y vistas modulares.

En Code-XR, este fichero asegura que la extensión arranque en un estado limpio y predecible, y que al cerrarse no queden procesos en segundo plano ni recursos bloqueados.

4.4.6. Registro de comandos e interfaz de usuario

El registro de comandos es el mecanismo que conecta las acciones del usuario con la lógica implementada en la extensión. Cada comando se declara en `package.json` y se enlaza en `extension.ts` mediante su *handler*. En Code-XR, los comandos abarcan desde el análisis de ficheros y directorios hasta la gestión de servidores y ejemplos de BabiaXR.

La interfaz de usuario se apoya en primitivas de la API de VS Code como:

- **TreeView / TreeDataProvider**: para organizar servidores, ejemplos y análisis activos en la vista lateral.
- **WebviewPanel** y **WebviewView**: para mostrar visualizaciones interactivas en 2D (LivePanel) o XR.
- **StatusBarItem** y elementos contextuales: para notificaciones rápidas y accesos directos.

Estas primitivas permitieron a Code-XR ofrecer una experiencia integrada en el flujo habitual del editor, combinando menús, paneles y vistas enriquecidas en una misma extensión.

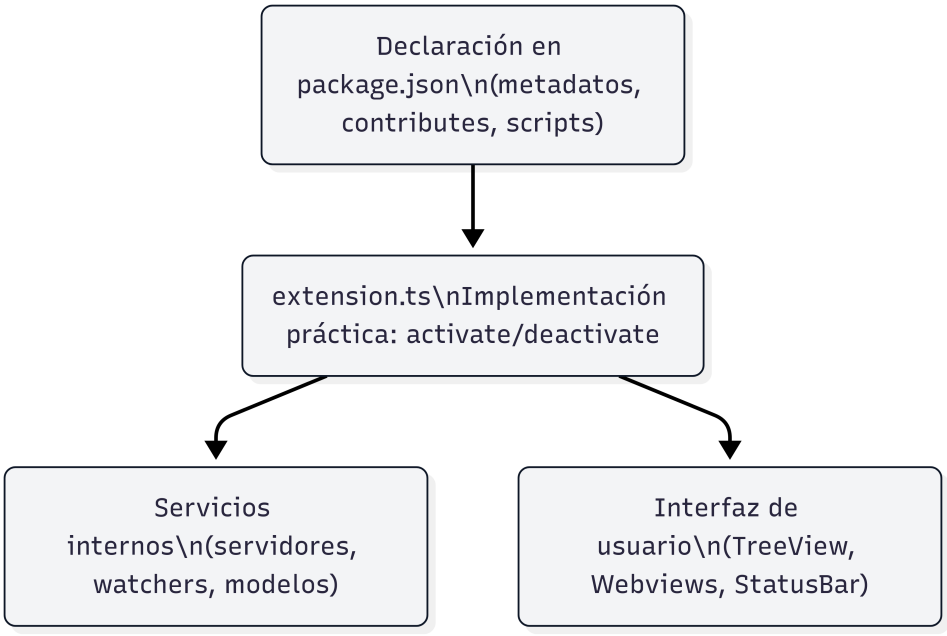


Figura 4.5: Relación entre los componentes clave de una extensión VS Code en Code-XR.

4.5. Sprint 1 — Lanzamiento de servidores locales

Introducción. Este fue el primer objetivo planteado para poner en marcha Code-XR: dotar a la extensión de la capacidad de lanzar servidores locales en *localhost*, permitiendo que los recursos de la extensión pudieran servirse de manera accesible desde un navegador o desde la propia interfaz del editor. La idea se inspiraba en el funcionamiento de otras extensiones consolidadas como *Live Server*[11], que facilitan un flujo de trabajo más ágil al eliminar la necesidad de configurar manualmente un servidor externo.

4.5.1. Primeras versiones — lanzamiento de servidores

En las primeras versiones el servidor era mínimamente funcional: servía un HTML fijo con una escena A-Frame de ejemplo (prueba de concepto) y ofrecía comandos básicos para arrancar y parar la visualización desde la paleta de comandos. Posteriormente se fueron incorporando capacidades incrementales:

- selección del fichero HTML a servir en lugar de usar una plantilla fija;
- Control de múltiples servidores y gestión automática de puertos: la extensión incorporó la librería `get-port`[31], que permite detectar de forma dinámica puertos disponibles en el sistema. De este modo se evitaba la colisión con puertos ya ocupados y se garantizaba que cada servidor local pudiera lanzarse sin interferencias. Además, este mecanismo facilitó la reutilización de puertos en sesiones posteriores cuando resultaba apropiado, proporcionando una experiencia más fluida y predecible para el usuario.
- Soporte HTTPS: se añadió la posibilidad de lanzar los servidores en modo seguro utilizando certificados por defecto incluidos en la extensión el par `cert`/`key` ubicado en la carpeta interna `certs` preparados con `OpenSSL`[29]. Como alternativa, el usuario puede seleccionar sus propios certificados autofirmados mediante el explorador de ficheros, indicando las rutas de `cert` y `key`. Esta doble modalidad (certificados internos frente a certificados personalizados) aporta flexibilidad y facilita la adopción de HTTPS en distintos entornos sin complicar el flujo de lanzamiento.

Llegados a este punto de mínima funcionalidad, con la capacidad de servir los servidores en modo HTTPS, se pudo dar por concluido el primer hito y avanzar hacia las siguientes etapas del

desarrollo. No obstante, antes de continuar, en la próxima subsección se detalla el estado actual del sistema de lanzamiento de servidores, tanto en lo referente a la interfaz de usuario como a su funcionamiento interno dentro del plugin.

4.5.2. Versión actual — lanzamiento y gestión de servidores

En la versión actual de Code-XR, el subsistema de servidores proporciona un entorno robusto para servir visualizaciones con **arquitectura modular**, **persistencia de preferencias**, **multi-servidor concurrente**, **resolución automática de puertos**, **soporte HTTPS completo** y **gestión centralizada de estado**. El diseño sigue patrones de desarrollo escalables con separación clara de responsabilidades y gestión de ciclo de vida automatizada, a continuación se detallan los componentes clave, para detalles mas específicos ver el código fuente en el repositorio[14].

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 `httpServer.ts`, `httpsDefaultServer.ts`, `httpsCustomServer.ts`, `multiServerLauncher.ts`, `portManager.ts` y subsistema `sse/` para comunicación bidireccional.
- **storage**: `serverSettingsManager.ts` implementa persistencia con *deep merge*, validación atómica y migraciones de esquema.
- **registry**: `activeServerRegistry.ts` (singleton) gestiona el estado centralizado con eventos síncronos para UI.
- **services**: `serverRegistrar.ts` valida registros, `panelManager.ts` controla WebviewPanels del editor.
- **views**: Integración con TreeView gracias a `ActiveServersSectionProvider.ts`.

Configuración y persistencia (globalStorage) El `ServerSettingsManager` (patrón singleton) gestiona preferencias mediante archivo JSON `server-settings.json` guardado en `globalStorage` usando `context.globalStorageUri`, garantizando **persistencia**

cross-workspace. Cuando el usuario cambia la configuracion ya sea usando la interfaz del UI o a través de comandos el fichero se actualiza automáticamente. La estructura de configuración consta con los siguientes campos:

Listing 4.1: Ejemplo del archivo `server-settings.json` que persiste la configuración de servidores en Code-XR.

```
{
  "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"
}
```

El `configNonce` (SHA-256) previene corrupción en actualizaciones concurrentes, mientras que `version` permitiría hacer migraciones si el esquema cambia en futuras versiones.

UI de configuración avanzada El panel de configuración (Figura 4.6) expone controles granulares: selección **HTTP/HTTPS** con validación de certificados, configuración de **puerto por defecto** con verificación de disponibilidad, es decir, se buscara un puerto libre desde el selecionado por defecto, **apertura automática** configurable, **modo de lanzamiento** (panel VS Code vs navegador), funciones de **reset/restore** y lanzamiento directo con la configuración activa.

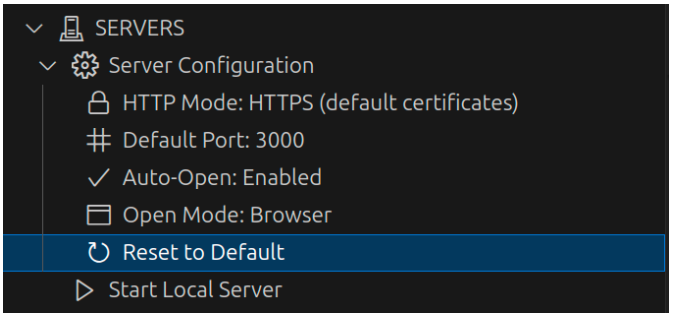


Figura 4.6: Panel de configuración avanzada de servidores en Code-XR con validación en tiempo real.

Multi-servidor concurrente y gestión inteligente de puertos El `MultiServerLauncher` orquesta **instancias concurrentes** con identificadores únicos (`server-{port}-{timestamp}`), tipado fuerte (`http | https-default | https-custom`) y metadatos completos (puerto, archivo servido, registro activo). El `PortManager` implementa descubrimiento automático de puertos mediante:

1. Verificación de disponibilidad en puerto preferido usando bind a `0.0.0.0`
2. Resolución automática de colisiones mediante la librería `get-port`[31], que realiza una búsqueda secuencial desde el puerto por defecto configurado hasta el rango superior (8080), seleccionando el primer puerto disponible.
3. Fallback manual con límites configurables y notificación al usuario
4. Soporte multi-puerto para servicios paralelos

Ante colisiones, el sistema notifica transparentemente la reasignación: *"Puerto 3000 ocupado, servidor lanzado en puerto 3001"*.

HTTPS y gestión de certificados El soporte HTTPS implementa dos modalidades con validación previa:

- **Certificados por defecto**: Par `cert/key` incluido en `certs/` (generados con `OpenSSL`[29]), permitiendo HTTPS inmediato sin configuración.

- **Certificados personalizados**: Selector de archivos con validación de compatibilidad OpenSSL, rutas absolutas y verificación de integridad antes del lanzamiento.

El sistema detecta automáticamente conflictos **HTTPS + panel lateral** (incompatibilidad de seguridad VS Code) y sugiere apertura del servidor en navegador web con un mensaje contextual.

Active Servers: registro centralizado y observabilidad La vista *Active Servers* expone un **registro centralizado** (`ActiveServerRegistry`, singleton con patrón Observer) que rastrea instancias activas con:

- **Estado completo**: ID único, puerto, URL, modo de lanzamiento, certificados, timestamp, estado (`running|stopped|error`).
- **Metadatos contextuales**: Archivo servido, nombre personalizado, sesión de análisis asociada, configuración de red.
- **Etiquetado inteligente**: Prioridad `customName` → `localhost:port`, descripción con `launchMode` y indicadores de estado.
- **Acciones contextuales**: *Open in Browser*, *Open in Panel* (solo HTTP), *Copy URL* (local/-red), *Stop*, *Server Details* y *Stop All* (disponible con 2+ servidores).
- **Iconografía diferenciada**: Globe (HTTP), Shield (HTTPS-default), Shield-check (HTTPS-custom) con colorimetría por estado (verde/gris/rojo).

El registro es **efímero** (no persiste entre sesiones) para prevenir servidores zombie y garantizar coherencia de entorno (puertos disponibles, interfaces de red, certificados válidos).

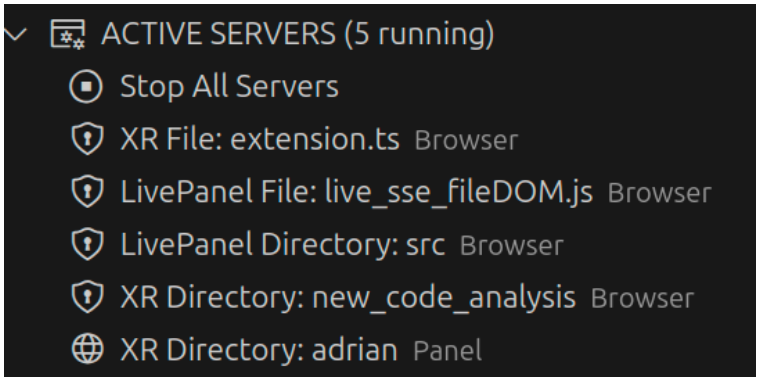


Figura 4.7: Vista *Active Servers*: registro centralizado con acciones contextuales y estado en tiempo real.

Comunicación en tiempo real (SSE) y sincronización Los servidores exponen endpoints `/events` con *Server-Sent Events* (SSE) que implementan:

- **Propagación automática**: Cambios de análisis, actualizaciones de datos y eventos de sistema sin polling.
- **Gestión de clientes**: Registro/limpieza automática de conexiones SSE por archivo analizado.
- **Mapeo archivo-servidor**: Asociación directa entre sesiones de análisis y servidores activos para updates dirigidos.
- **Bus de comunicación**: El servidor actúa como intermediario entre extensión y visualizaciones web.

Integración con análisis y limpieza automática El sistema implementa **limpieza coordinada** entre componentes:

1. **Parada de servidor**: Cierra paneles laterales, limpia clientes SSE, elimina mapeos archivo-servidor, cierra sesiones de análisis asociadas.
2. **Gestión de ciclo de vida**: Validación periódica de estado, detección de servidores zombie, recovery automático de errores.

3. **Sincronización UI**: Eventos del registro activo actualizan TreeView en tiempo real, comandos VS Code integrados.

4. **Cleanup de extensión**: Liberación ordenada de todos los recursos al desactivar Code-XR.

Acceso de red y compatibilidad VR/móvil El `NetworkUtils` detecta automáticamente la IP local y genera URLs de acceso externo, facilitando el uso desde dispositivos VR o móviles en la misma red. Los servidores se configuran con bind a `0.0.0.0` (todas las interfaces) y proporcionan diagnósticos de red para troubleshooting.

Síntesis técnica El subsistema combina **arquitectura modular escalable**, **configuración persistente con validación**, **multi-servidor concurrente robusto**, **gestión inteligente de puertos**, **HTTPS** y **registro centralizado con observabilidad**. La integración completa con VS Code (TreeView, comandos, paneles) y el canal SSE bidireccional posicionan este módulo como la columna vertebral técnica de Code-XR, proporcionando una base sólida para el ecosistema de visualización XR.

Capítulo 5

Experimentos y validación

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

Capítulo 6

Resultados

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

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

Capítulo 7

Conclusiones

7.1. Consecución de objetivos

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

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

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

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

7.2. Aplicación de lo aprendido

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

1. a
2. b

7.3. Lecciones aprendidas

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

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

7.4. Trabajos futuros

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

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

Apéndice A

Manual de usuario

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

Bibliografía

[1] esbuild — an extremely fast javascript bundler. https://esbuild.github.io/. Accedido: 2025-09-06.

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

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

[4] Anthropic. Claude 4. https://claude.ai/, 2024. Accedido el 3 de agosto de 2025.

[5] T. Ball and S. G. Eick. Software visualization in the large. *Computer*, 29(4):33–43, 1996. https://doi.org/10.1109/2.488299.

[6] 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.

[7] R. y. c. Cabello. Three.js: Javascript 3d library. https://threejs.org/, 2024. Accedido el 3 de agosto de 2025.

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

[9] 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.

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

[11] 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.

[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] 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.

[14] 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.

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

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

[17] 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.

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

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

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

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

[22] 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.

[23] 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.

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

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

[26] Mozilla Developer Network. Server-sent events (sse) — mdn web docs. https://developer.mozilla.org/en-US/docs/Web/API/Server-sent_events, 2024. Accedido el 3 de agosto de 2025.

[27] npm. express-sse (npm package). https://www.npmjs.com/package/express-sse, 2024. Librería para implementación simplificada de Server-Sent Events en Express; Accedido el 3 de agosto de 2025.

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

[29] 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.

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

[31] 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.

[32] 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.

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

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

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

[36] 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.

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