Universidad
Rey Juan Carlos

ESCUELA DE INGENIERÍA DE FUENLABRADA

INGENIERÍA TELEMÁTICA

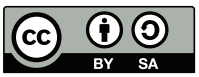
TRABAJO FIN DE GRADO

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

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

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

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

Índice general

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 [5]. 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 [1, 6].

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 [3]. 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 [2].

Uno de los enfoques más influyentes ha sido el de la ciudad del software, introducido por herramientas como CodeCity [9] o ExplorViz [4], 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 [9, 4].

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 [9]. 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 [7].

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 [4].

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 [9, 4].

- **DOM Mode**: pensado para proyectos web, representa la estructura del DOM HTML en 3D, facilitando la exploración jerárquica y la detección de anidamientos complejos [7].

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 [8].

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 como CodeCity, ExplorViz o BabiaXR. 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

Aquí vendría el objetivo general en una frase: Mi trabajo fin de grado consiste en crear de una herramienta de análisis de los comentarios jocosos en repositorios de software libre alojados en la plataforma GitHub.

Recuerda que los objetivos siempre vienen en infinitivo.

2.2. Objetivos específicos

Los objetivos específicos se pueden entender como las tareas en las que se ha desglosado el objetivo general. Y, sí, también vienen en infinitivo.

2.3. Planificación temporal

A mí me gusta que aquí pongáis una descripción de lo que os ha llevado realizar el trabajo. Hay gente que añade un diagrama de GANTT. Lo importante es que quede claro cuánto tiempo llevas (tiempo natural, p.ej., 6 meses) y a qué nivel de esfuerzo (p.ej., principalmente los fines de semana).

Capítulo 3

Estado del arte

Descripción de las tecnologías que utilizas en tu trabajo. Con dos o tres párrafos por cada tecnología, vale. Se supone que aquí viene todo lo que no has hecho tú.

Puedes citar libros, como el de Bonabeau et al., sobre procesos estigmérgicos [**?**]. Me encantan los procesos estigmérgicos. Deberías leer más sobre ellos. Pero quizás no ahora, que tenemos que terminar la memoria para sacarnos por fin el título. Nota que el ~ añade un espacio en blanco, pero no deja que exista un salto de línea. Imprescindible ponerlo para las citas.

Citar es importantísimo en textos científico-técnicos. Porque no partimos de cero. Es más, partir de cero es de tontos; lo suyo es aprovecharse de lo ya existente para construir encima y hacer cosas más sofisticadas. ¿Dónde puedo encontrar textos científicos que referenciar? Un buen sitio es Google Scholar[1]. Por ejemplo, si buscas por "stigmergy libre software" para encontrar trabajo sobre software libre y el concepto de *estigmergia* (¿te he comentado que me gusta el concepto de estigmergia ya?), encontrarás un artículo que escribí hace tiempo cuyo título es "Self-organized development in libre software: a model based on the stigmergy concept". Si pulsas sobre las comillas dobles (entre la estrella y el "citado por ...", justo debajo del extracto del resumen del artículo, te saldrá una ventana emergente con cómo citar. Abajo a la derecha, aparece un enlace BibTeX. Púlsalo y encontrarás la referencia en formato BibTeX, tal que así:

```
@inproceedings{robles2005self,
  title={Self-organized development in libre software:
         a model based on the stigmergy concept},
  author={Robles, Gregorio and Merelo, Juan Juli\'an
```

[1] `http://scholar.google.com`

Uno	2	3
Cuatro	5	6
Siete	8	9

Cuadro 3.1: Ejemplo de tabla. Aquí viene una pequeña descripción (el *caption*, el pie de tabla/figura) del contenido de la tabla. Si la tabla no es autoexplicativa, siempre viene bien aclararla aquí.

```
          and Gonz\'alez-Barahona, Jes\'us M.},
  booktitle={ProSim'05},
  year={2005}
}
```

Copia el texto en BibTeX y pégalo en el fichero `memoria.bib`, que es donde están las referencias bibliográficas. Para incluir la referencia en el texto de la memoria, deberás citarlo, como hemos hecho antes con [**?**], lo que pasa es que en vez de el identificador de la cita anterior (bonabeau:_swarm), tendrás que poner el nuevo (robles2005self). Compila el fichero `memoria.tex` (`pdflatex memoria.tex`), añade la bibliografía (`bibtex memoria.aux`) y vuelve a compilar `memoria.tex` (`pdflatex memoria.tex`)...y *voilà* ¡tenemos una nueva cita [**?**]!

También existe la posibilidad de poner notas al pie de página, por ejemplo, una para indicarte que visite la página del GSyC[2].

3.1. Sección 1

Hemos hablado de cómo incluir figuras. Pero no hemos dicho nada de tablas. A mí me gustan las tablas. Mucho. Aquí un ejemplo de tabla, la Tabla 3.1 (siento ser pesado, pero nota cómo he puesto la referencia).

Hay un sitio en Internet donde puedes diseñar las tablas fácilmente y luego hacer un corta y pega del resultado en tu editor. Puedes probarlo en `https://www.tablesgenerator.com/`.

[2] `http://gsyc.es`

Capítulo 4

Diseño e implementación

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

4.1. Arquitectura general

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

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

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

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

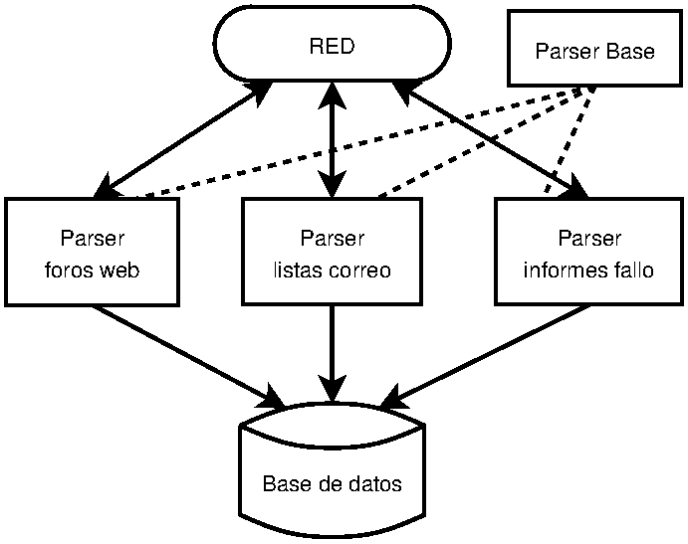


Figura 4.1: Estructura del parser básico

Capítulo 5

Experimentos y validación

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

Capítulo 6

Resultados

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

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

Capítulo 7

Conclusiones

7.1. Consecución de objetivos

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

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

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

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

7.2. Aplicación de lo aprendido

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

1. a

2. b

7.3. Lecciones aprendidas

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

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

7.4. Trabajos futuros

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

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

Apéndice A

Manual de usuario

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

Bibliografía

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

[2] 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.

[3] 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.

[4] 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.

[5] 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.

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

[7] 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.

[8] 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.

[9] 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.