Contenido

[1. Autores del trabajo, planificación y entrega 3](#_Toc450604272)

[1.1 Autores 3](#_Toc450604273)

[1.2 Planificación 3](#_Toc450604274)

[1.3 Entrega 3](#_Toc450604275)

[2. Requisitos del prototipo a implementar 4](#_Toc450604276)

[2.1 Requisitos funcionales 4](#_Toc450604277)

[2.2 Otros requisitos 4](#_Toc450604278)

[3. Criterios de comparación en la implementación 6](#_Toc450604279)

[**3.1 Criterio 1: Compatibilidad con Java** 6](#_Toc450604280)

[**3.2 Criterio 2: Velocidad de carga media.** 6](#_Toc450604281)

[**3.3 Criterio 3: Estabilidad del sistema.** 6](#_Toc450604282)

[**3.4 Criterio 4: Concurrencia del sistema.** 6](#_Toc450604283)

[**3.5 Criterio 5: Recursos necesarios.** 6](#_Toc450604284)

[**3.6 Criterio 6: Intercambio entre mapas.** 7](#_Toc450604285)

[**3.7 Criterio 7: Manejo del mapa.** 7](#_Toc450604286)

[**3.8 Criterio 8: Clases** 7](#_Toc450604287)

[**3.9 Criterio 9: Archivos.** 7](#_Toc450604288)

[**3.10 Criterio 10: Líneas.** 7](#_Toc450604289)

[**3.11 Criterio 11: Comentarios.** 7](#_Toc450604290)

[**3.12 Criterio 12: Ejecutables.** 8](#_Toc450604291)

[**3.13 Criterio 13: Ratio.** 8](#_Toc450604292)

[**3.14 Criterio 14: Complejidad.** 8](#_Toc450604293)

[**3.15 Criterio 15: Advertencias.** 8](#_Toc450604294)

[**3.16 Criterio 16: Errores.** 8](#_Toc450604295)

[4. Proyecto de implementación de un prototipo del sistema utilizando la tecnología de OpenStreet Maps. 9](#_Toc450604296)

[4.1 Documentación de diseño 9](#_Toc450604297)

[4.2 Documentación de construcción 9](#_Toc450604298)

[4.3 Documentación de pruebas 10](#_Toc450604299)

[4.4 Documentación de instalación 10](#_Toc450604300)

[Al tratarse de una aplicación portable en java, no requiere la necesidad previa de instalarse, únicamente hay que ejecutar el archivo portable y se abrirá la aplicación. 10](#_Toc450604301)

[4.5 Manual de usuario 10](#_Toc450604302)

[5. Proyecto de implementación de un prototipo del sistema utilizando la tecnología Google Maps 12](#_Toc450604303)

[5.1 Documentación de diseño 12](#_Toc450604304)

[5.2 Documentación de construcción 12](#_Toc450604305)

[5.3 Documentación de pruebas 13](#_Toc450604306)

[5.4 Documentación de instalación 14](#_Toc450604307)

[Al tratarse de una aplicación portable en java, no requiere la necesidad previa de instalarse, pero en este caso, como usamos la api de google tendremos que añadir el archivo map.html en la ubicación correspondiente(en este caso c:\). 14](#_Toc450604308)

[5.5 Manual de usuario 14](#_Toc450604309)

[6. Comparación de las dos implementaciones 17](#_Toc450604310)

[6.1 Evaluación de los criterios en la implementación usando la tecnología OpenStreet Map 17](#_Toc450604311)

[6.2 Evaluación de los criterios en la implementación usando la tecnología Google Maps 18](#_Toc450604312)

[7. Comparación de la implementación de las tecnologías 18](#_Toc450604313)

[8. Conclusiones 31](#_Toc450604314)

# 1. Autores del trabajo, planificación y entrega

## 1.1 Autores

Somos el grupo 2 del turno de mañana. Los integrantes del equipo son:

1. Korneliusz Zbyrad (Coordinador)
2. Ángel Martínez
3. David Jiménez
4. Alex Igna
5. Iván Rodas Padilla

## 1.2 Planificación

## 1.3 Entrega

https://github.com/WinterUAH/TG3

* Trabajo terminado: del trabajo terminado con el nombre TG3\_final.docx
* Presentación del trabajo: TG3\_final.pptx
* Direccion para descargar los programas: Enlaces a los programas.txt

# 2. Requisitos del prototipo a implementar

El objetivo del proyecto es comparar la implementación de un mismo prototipo de sistema utilizando dos tecnologías diferentes (A y B).

Es importante cumplimentar este apartado antes de empezar a implementar el prototipo de cada tecnología, porque ambos prototipos deben cumplir los requisitos que se establezcan en este apartado. Si se van a crear dos equipos de trabajo, uno para cada prototipo, el contenido de este apartado es lo que han de compartir ambos equipos como punto de partida.

Cuanto más detallados sean los requisitos, mayor será la precisión en la comparación que se realizará al final del trabajo. Se trata de conseguir dos prototipos con igual funcionalidad, pero utilizando diferentes tecnologías.

Se puede dar libertad a los equipos de desarrollo en cuanto al diseño, pero la funcionalidad debe ser lo más parecida posible. Por ejemplo, no es necesario que los colores utilizados en las pantallas sean exactamente los mismos en ambos prototipos, a no ser que los miembros del grupo lo hayan decidido así, en cuyo caso, esos detalles de colores deben incluirse en el catálogo de requisitos, para que ambos equipos los cumplan.

## 2.1 Requisitos funcionales

Los requisitos funcionales deben ser los mismos para las dos implementaciones.

En la siguiente tabla se indicará el catálogo de requisitos funcionales del sistema.

| **REQ.** | **DESCRIPCIÓN** |
| --- | --- |
| RF01 | **Visualizar mapa:** se debe de arrancar la aplicación visualizando alguna parte del mapa. |
| RF02 | **Acercar zoom:**  hacer zoom para ver en más detalle los mapas. |
| RF03 | **Alejar zoom:**  quitar zoom para ver en mayor amplitud los mapas. |
| RF04 | **Buscar ubicación:** el usuario podrá escribir una dirección y que la aplicación la encuentre. |
| RF05 | **Trazar rutas:** el usuario podrá poner dos direcciones y que la aplicación se encargue de trazar una ruta. |
| RF06 | **Ruta en coche:** eligiendo ruta, se podrá filtrar para que la aplicación trace la mejor ruta usando un coche. |
| RF07 | **Ruta a pie:** eligiendo ruta, se podrá filtrar para que la aplicación trace la mejora ruta para viajar a pie. |
| RF08 | **Mostrar indicaciones:** al trazar una ruta y elegir el tipo de transporte, la aplicación mostrará en forma de texto las indicaciones para realizar la ruta. |
| RF09 | **Imprimir:** el sistema deberá tener la capacidad de imprimir el mapa visualizado. |
|  |  |

## 2.2 Otros requisitos

Se pueden incluir aquí otros requisitos para el prototipo que no puedan considerarse como funcionales. Por ejemplo, requisitos de datos, de seguridad, de interfaz de usuario, de rendimientos, etc.

Se puede dejar libertad

En la siguiente tabla se indicará el catálogo de requisitos no funcionales del sistema.

| **REQ.** | **DESCRIPCIÓN** |
| --- | --- |
| R01 | **Velocidad inicio/ejecución:** el sistema deberá de responder de manera ágil tanto en el arranque del mismo como en la ejecución de las tareas del mismo.  Se valora que el sistema arranque en un tiempo inferior a 2 segundos. |
| R02 | **Comentarios:** el código de la aplicación debe de estar lo suficientemente documentado para que sea más fácil las tareas de mantenimiento/modificación.  Se valora que la tasa de comentarios no sea inferior a 0.10. |
| R03 | **Errores:** tanto el código del sistema como la aplicación no deben de tener errores debido a una mala implementación. |
| R04 | **Warnings:** el código de la aplicación deberá tener el mínimo número posible de warnings. Se permite la posibilidad de la existencia de los mismos pero se valora la total ausencia. |
| R05 | **Complejidad ciclomática:** evaluación del riesgo, uno de los requisitos fundamentales es que la complejidad del código no supere jamás la categoría 1 de la clasificación que se adjunta a continuación:   1. 1-10 Programa Simple, sin mucho riesgo 2. 11-20 Más complejo, riesgo moderado 3. 21-50Complejo, Programa de alto riesgo 4. 50Programa no testeable, Muy alto riesgo |

# 3. Criterios de comparación en la implementación

En el trabajo TG2 se definieron criterios de comparación de las dos tecnologías a nivel teórico.

En este trabajo hay que definir criterios para la comparación de la implementación de las tecnologías en la construcción del prototipo de sistema de ejemplo, cuyos requisitos son los establecidos en el apartado 2.

Se trata de criterios del tipo” “horas empleadas en el desarrollo del sistema”, “velocidad de funcionamiento del sistema”, “recursos necesarios”, etc.

### **3.1 Criterio 1: Compatibilidad con Java**

Nombre del criterio: Compatibilidad con Java

Descripción: Comprobar que tal se adaptan los dos Web Map Service al entorno de desarrollo Java.

Tipo de valor: Booleano (Si/No)

### **3.2 Criterio 2: Velocidad de carga media.**

Nombre del criterio: Velocidad media.

Descripción: Se comprueba cuánto tarda de media en cargar la pagina/dirección deseada.

Tipo de valor: Numérico (valor variable en función del ancho de banda y las características del servidor/ordenador)

### **3.3 Criterio 3: Estabilidad del sistema.**

Nombre del criterio: Estabilidad.

Descripción: Se comprueba qué tal responde el sistema ante una continua petición de datos a lo largo de un periodo.

Tipo de valor: Numérico (valor múltiple dentro de un rango en función de los parámetros que se analicen)

### **3.4 Criterio 4: Concurrencia del sistema.**

Nombre del criterio: Concurrencia.

Descripción: Se comprueba qué tal responde el sistema ante una situación de estrés, es decir, ante un número elevado de peticiones de manera simultánea.

Tipo de valor: Numérico (valor múltiple dentro de un rango en función de los parámetros que se analicen).

### **3.5 Criterio 5: Recursos necesarios.**

Nombre del criterio: Recursos.

Descripción: Contabilizar la cantidad de recursos empleados para el desarrollo de cada implementación.

Tipo de valor: Numérico.

### **3.6 Criterio 6: Intercambio entre mapas.**

Nombre del criterio: Cambiar mapas.

Descripción: Analizar cómo se comporta el entorno Java a la hora de intercambiar la visualización del mapa (entre Satélite y Trafico, por ejemplo).

Tipo de valor: Booleano.

### **3.7 Criterio 7: Manejo del mapa.**

Nombre del criterio: Manejo del mapa.

Descripción: Analizar cómo se comporta el entorno Java a la hora de trabajar con el mapa en cuestión (acercar y alejar una zona, por ejemplo).

Tipo de valor: Booleano.

### **3.8 Criterio 8: Clases**

Nombre del criterio: Cantidad de clases .

Descripción: Saber el número exacto de clases del proyecto ayuda a saber la complejidad del desarrollo

Tipo de valor: Numérico

### **3.9 Criterio 9: Archivos.**

Nombre del criterio: Cantidad de archivos.

Descripción: Saber el número exacto de archivos del proyecto ayuda a saber la complejidad del desarrollo

Tipo de valor: Numérico.

### **3.10 Criterio 10: Líneas.**

Nombre del criterio: Líneas de código

Descripción: Saber el número exacto de líneas de código del proyecto ayuda a saber la complejidad del desarrollo

Tipo de valor: Numérico.

### **3.11 Criterio 11: Comentarios.**

Nombre del criterio: Comentarios realizados.

Descripción: Los comentarios son vitales para entender el proceso de desarrollo del código, los métodos deben de estar documentados para posteriores modificaciones y la comprensión de la aplicación.

Tipo de valor: Numérico

### **3.12 Criterio 12: Ejecutables.**

Nombre del criterio: Sentencias ejecutables.

Descripción: Se trata de analizar las sentencias que son ejecutables dentro del código.

Tipo de valor: Numérico.

### **3.13 Criterio 13: Ratio.**

Nombre del criterio: Ratio comentarios/código.

Descripción: La cantidad de comentarios realizada debe de tener correlación con las líneas de código existentes, un valor alto puede significar que hay un exceso de comentarios y por tanto complicar la comprensión del código.

Tipo de valor: Numérico.

### **3.14 Criterio 14: Complejidad.**

Nombre del criterio: Complejidad ciclomátca

Descripción: proporciona una medición cuantitativa de la [complejidad lógica de un programa](https://es.wikipedia.org/w/index.php?title=Complejidad_l%C3%B3gica_de_un_programa&action=edit&redlink=1). Es una de las métricas de software de mayor aceptación, ya que se trata de una métrica independiente del lenguaje que se esté utilizando.

Tipo de valor: Numérico.

### **3.15 Criterio 15: Advertencias.**

Nombre del criterio: Cantidad de advertencias

Descripción: Utilizando el software Understand podemos ver cuantos “warnings” tenemos dentro de nuestro código.

Tipo de valor: Numérico.

### **3.16 Criterio 16: Errores.**

Nombre del criterio: Cantidad de errores

Descripción: Utilizando el software Understand podemos ver cuantos errores tenemos dentro del código de la aplicación.

Tipo de valor: Numérico.

# 4. Proyecto de implementación de un prototipo del sistema utilizando la tecnología de OpenStreet Maps.

## 4.1 Documentación de diseño

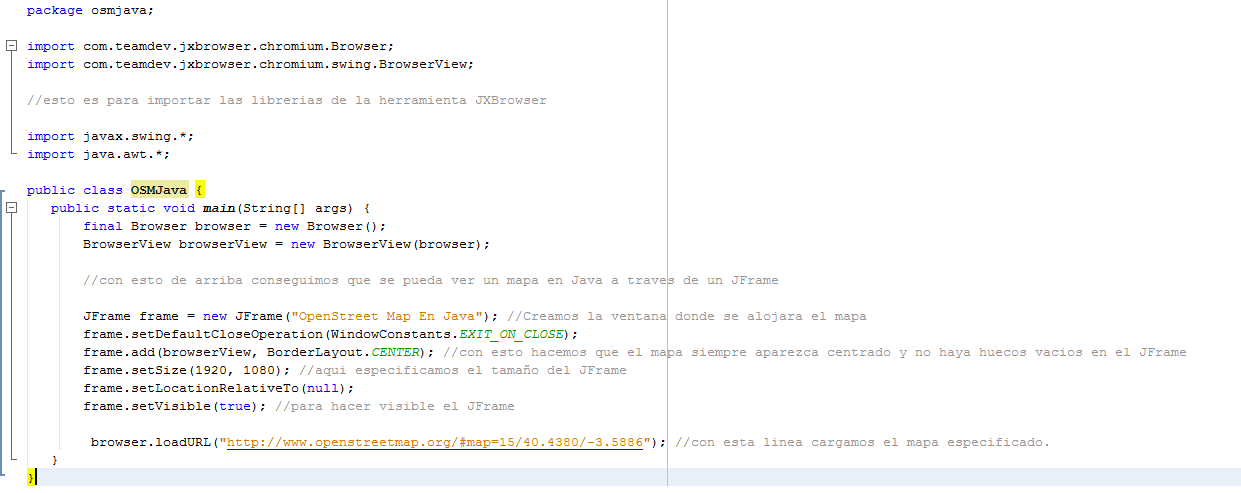
Nuestra intención era incluir el Web Map Service de OpenStreet Map en un JFrame de Java, para analizar diferentes aspectos, y a partir de ahí añadirle algunas funcionalidades a través de JButtons. De este modo, nuestra interfaz de diseño sería así:



## 4.2 Documentación de construcción

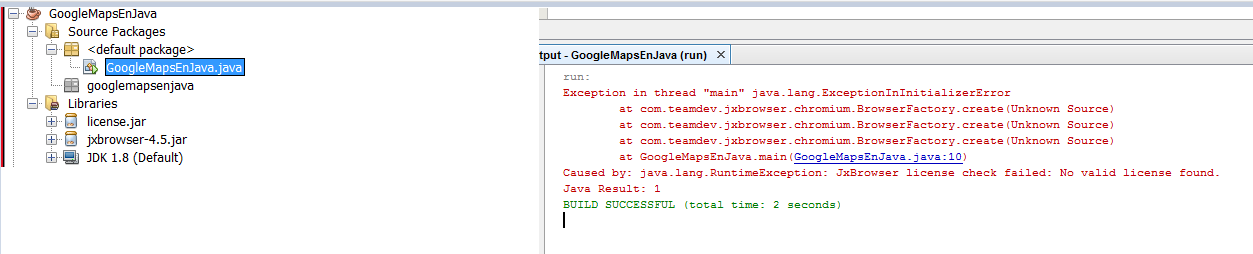
Lo primero que hicimos fue crear un JFrame donde incrustar el Web Map Service. En él se puede modificar el tamaño y la visibilidad del JFrame en cuestión.

Para hacer que el JFrame funcione, hay que añadir unas librerías del apartado JXBrowser, que es una herramienta externa que necesita una licencia para su funcionamiento. Esto nos dio ciertos problemas que comentaremos más adelante en el apartado de pruebas. Además, añadimos los botones de acercar/alejar y el de cambiar a modo trafico (Ver Transporte)

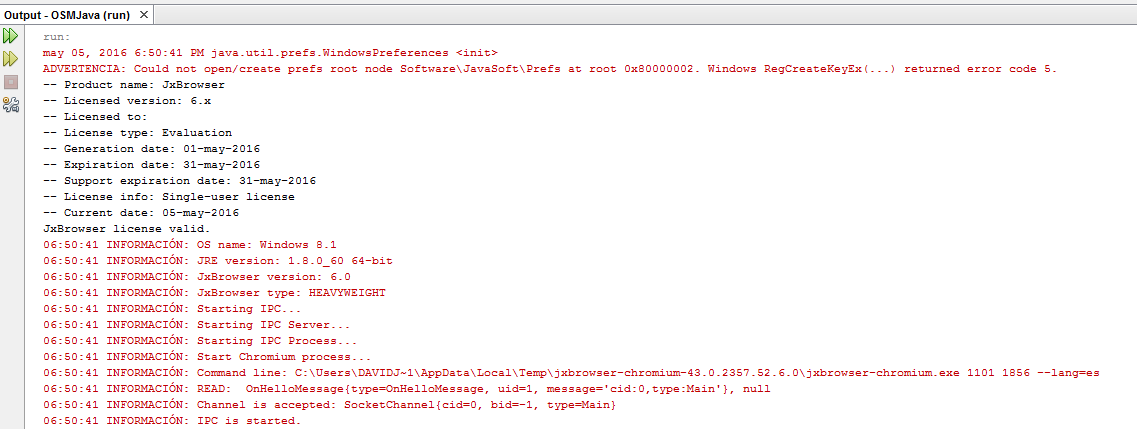


## 4.3 Documentación de pruebas

El primer error que nos salía al importar las librerías es que no teníamos una licencia para ejecutarlas. Por tanto, buscamos en la página oficial y vimos que la licencia completa cuesta dinero, pero que había una versión gratuita con 30 días de prueba, que es la que usamos. Una vez añadida al proyecto, llegamos a esta situación:



Nos decía que no encontraba una licencia válida. Buscando en diferentes webs, vimos que la versión de JXBrowser que teníamos no coincidía con la versión de la licencia que nos habían proporcionado, así que tuvimos que buscar otra más moderna, que es la versión 6.x, y con esto ya logramos que funcionara:



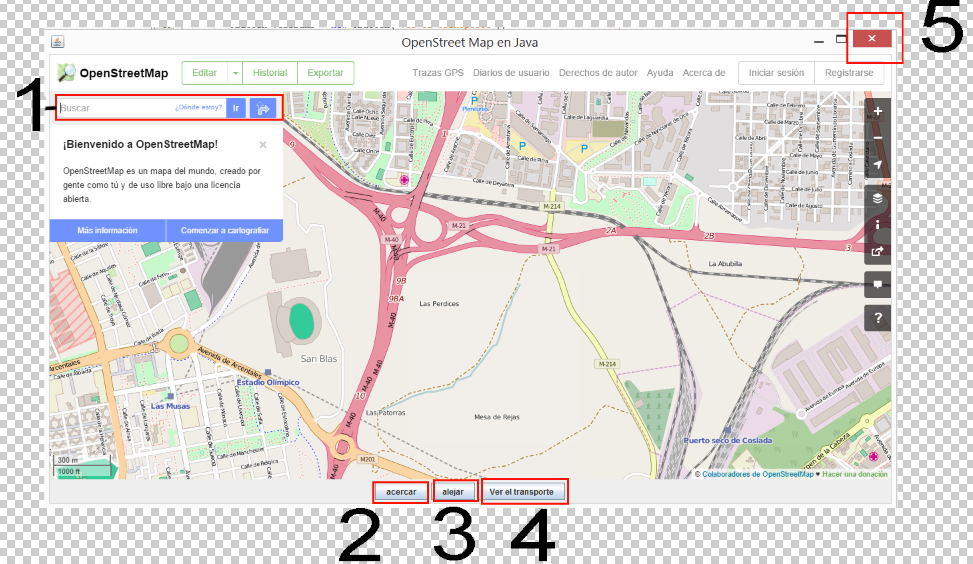
Aparte de estos problemas con las librerías, los botones de acercar/alejar no funcionan en OpenStreet Map, pero este apartado lo detallaremos más en la comparación entre tecnologías.

## 4.4 Documentación de instalación

## Al tratarse de una aplicación portable en java, no requiere la necesidad previa de instalarse, únicamente hay que ejecutar el archivo portable y se abrirá la aplicación.

## 4.5 Manual de usuario

La aplicación es muy sencilla de usar, solo con darle a ejecutar desde Java ya se carga el mapa, y tienes la posibilidad de usarlo tal y como lo harías en tu navegador, o bien usar los botones incorporados de acercar/alejar y ver transporte en caso de que por lo que sea la funcionalidad aportada por el propio Web Map Service no funcionase.



1. Buscador de dirección: como en cualquier navegador, encuentra la dirección solicitada.
2. Botón de acercar: amplia una zona seleccionada del mapa.
3. Botón de alejar: reduce una zona seleccionada del mapa.
4. Botón de Ver Transporte: cambia el mapa al modo de visualización por calles (para el tráfico).
5. Cierra la aplicación y se paran todos los procesos implicados.

**NOTA:** como hemos dicho, en OpenStreet Map los botones de acercar/alejar no funcionan, pero la implementación está correctamente realizada.

# 5. Proyecto de implementación de un prototipo del sistema utilizando la tecnología Google Maps

## 5.1 Documentación de diseño

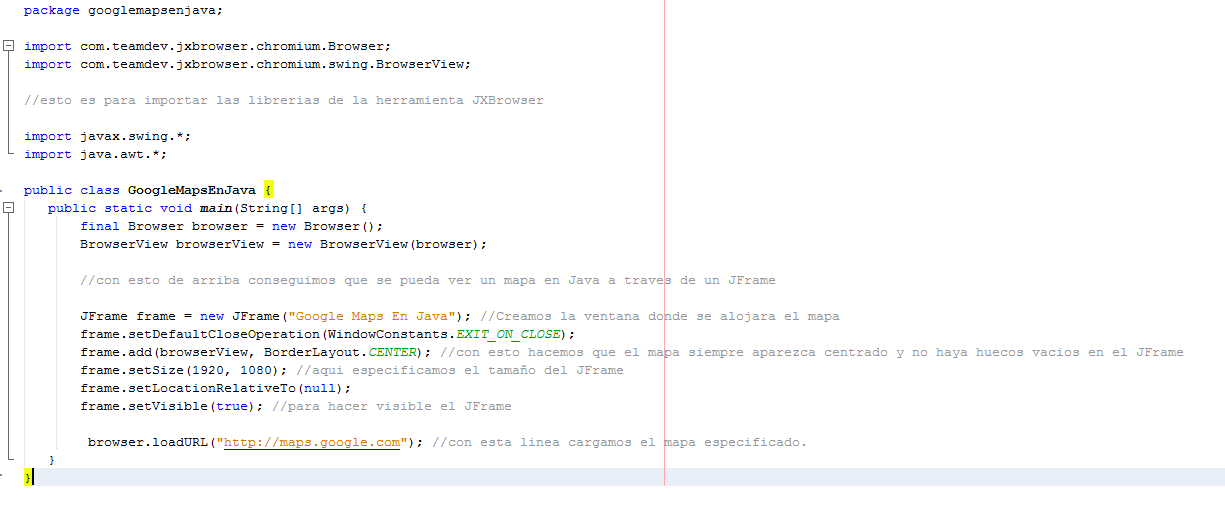
Nuestra intención era incluir el Web Map Service de Google Maps en un JFrame de Java, para analizar diferentes aspectos, y a partir de ahí añadirle algunas funcionalidades a través de JButtons. De este modo, nuestra interfaz de diseño sería así:



## 5.2 Documentación de construcción

Lo primero que hicimos fue crear un JFrame donde incrustar el Web Map Service. En él se puede modificar el tamaño y la visibilidad del JFrame en cuestión.

Para hacer que el JFrame funcione, hay que añadir unas librerías del apartado JXBrowser, que es una herramienta externa que necesita una licencia para su funcionamiento. Esto nos dio ciertos problemas que comentaremos más adelante en el apartado de pruebas. Además, añadimos los botones de acercar/alejar y el de cambiar a modo trafico (Ver Transporte):





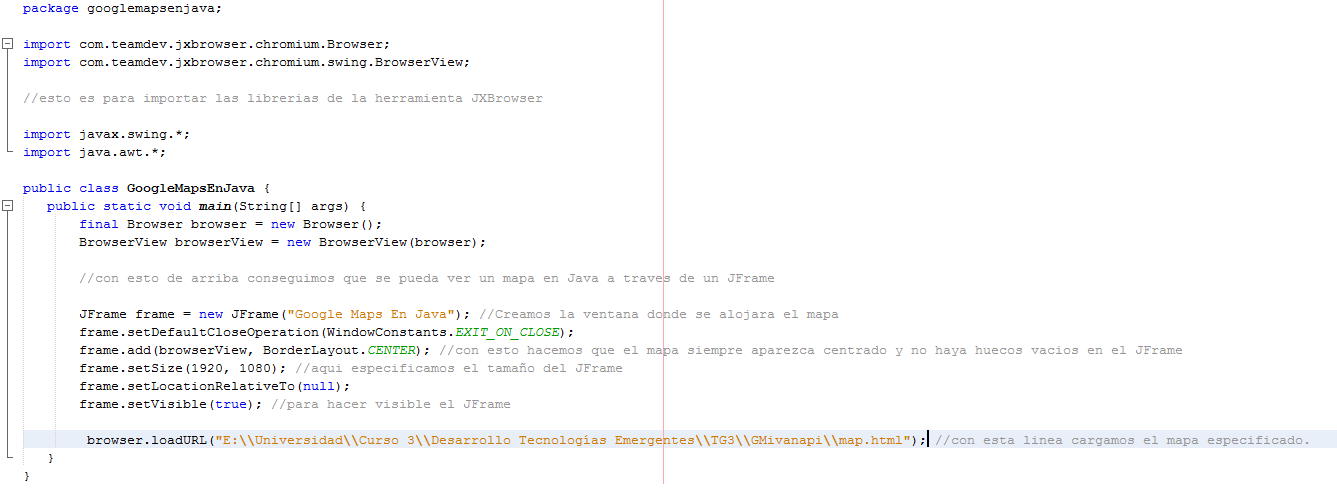
## 5.3 Documentación de pruebas

Como vimos que pasaba algo similar a lo de OpenStreet Map, es decir, no funcionaban los botones que habíamos incorporado, buscamos otro modo de realizarlo, y encontramos que en este caso Google Maps si ofrece una API de libre ejecución, por lo que tratamos de implementarla. En un primer momento lo que hicimos fue incorporarla directamente como url en el *browse.loadURL,* pero no realizaba nada. Así que pensando y buscando en Internet logramos dar con la solución: como en el *browse.loadURL* solo deja pasar como parametro una pagina web, implementamos nosotros una sencilla página html que llamara a la API de Google Maps. Para ello, tuvimos que crear una versión de prueba en la pagina de Google para que nos devolviera una API Key, y asi poderla incluir como referencia en la pagina html, quedando de la siguiente manera:



Como vemos, incluimos un apartado de javascript añadimos la API Key, y mas abajo unas coordenadas de inicio.

De este modo, en el código principal, solo cambiamos la página a la que referenciamos:



De este modo, al hacer uso de la API que proporciona Google, los botones de acercar/alejar sí que funcionan.

**NOTA:** como referenciamos a un archivo html creado por nosotros y no una web alojada en un servidor, hay que incluir la ruta donde se encuentra el archivo, por lo que hay que asegurarse de que es la correcta, ya que sino la carga del mapa será errónea.

## 5.4 Documentación de instalación

## Al tratarse de una aplicación portable en java, no requiere la necesidad previa de instalarse, pero en este caso, como usamos la api de google tendremos que añadir el archivo map.html en la ubicación correspondiente(en este caso c:\).

## 5.5 Manual de usuario

La aplicación es muy sencilla de usar, solo con darle a ejecutar desde Java ya se carga el mapa, y tienes la posibilidad de usarlo tal y como lo harías en tu navegador, o bien usar los botones incorporados de acercar/alejar y ver transporte en caso de que por lo que sea la funcionalidad aportada por el propio Web Map Service no funcionase.





1. Buscador de dirección: como en cualquier navegador, encuentra la dirección solicitada.
2. Botón de acercar: amplia una zona seleccionada del mapa.
3. Botón de alejar: reduce una zona seleccionada del mapa.
4. Botón de Ver Transporte: cambia el mapa al modo de visualización por calles (para el tráfico).
5. Cierra la aplicación y se paran todos los procesos implicados.

**NOTA:** por defecto al abrir la aplicación, al hacer uso de la API, inicia con el modo satélite, que no incluye buscador, y al pasar a modo transporte ya se genera todo.

# 6. Comparación de las dos implementaciones

Se trata de dar valores a los criterios de comparación definidos en el apartado 3 sobre la implementación de cada uno de los prototipos.

## 6.1 Evaluación de los criterios en la implementación usando la tecnología OpenStreet Map

Debe incluir al menos una tabla con la siguiente estructura.

| **CRITERIO** | **EVALUACIÓN** |
| --- | --- |
| Criterio 1: Compatibilidad con Java | Si |
| Criterio 2: Velocidad de Carga Media | 5.34 |
| Criterio 3: Estabilidad del Sistema | 7.6 sobre 10 |
| Criterio 4: Concurrencia del Sistema | 7 sobre 10 |
| Criterio 5: Recursos Necesarios | 4 |
| Criterio 6: Intercambio entre Mapas | Si |
| Criterio 7: Manejo del Mapa | No |
| Criterio 8: Clases | 4 |
| Criterio 9: Archivos | 6 |
| Criterio 10: Lineas | 1617 |
| Criterio 11: Comentarios | 197 |
| Criterio 12: Ejecutables | 20 |
| Criterio 13: Ratio | 0.14 |
| Criterio 14: Complejidad | 0.8 media, 2 máximo |
| Criterio 15: Advertencias | 19 |
| Criterio 16: Errores | 0 |

En cuanto a los recursos necesarios, en este caso habrían sido necesarios 4 elementos:

1. Entorno de desarrollo NetBeans
2. Licencia para la herramienta JXBrowser
3. Librerias JXBrowser 4.5 (versión desechada por ser demasiada antigua)
4. Librerias JXBrowser 6.0

## 6.2 Evaluación de los criterios en la implementación usando la tecnología Google Maps

| **CRITERIO** | **EVALUACIÓN** |
| --- | --- |
| Criterio 1: Compatibilidad con Java | Si |
| Criterio 2: Velocidad de Carga Media | 3.59 |
| Criterio 3: Estabilidad del Sistema | 8.3 sobre 10 |
| Criterio 4: Concurrencia del Sistema | 8.5 sobre 10 |
| Criterio 5: Recursos Necesarios | 5 |
| Criterio 6: Intercambio entre Mapas | Si |
| Criterio 7: Manejo del Mapa | Si |
| Criterio 8: Clases | 6 |
| Criterio 9: Archivos | 8 |
| Criterio 10: Líneas | 1700 |
| Criterio 11: Comentarios | 224 |
| Criterio 12: Ejecutables | 24 |
| Criterio 13: Ratio | 0.16 |
| Criterio 14: Complejidad | 0.8 media, 2 máximo |
| Criterio 15: Advertencias | 52 |
| Criterio 16: Errores | 0 |

En cuanto a los recursos necesarios, en este caso habrían sido necesarios 4 elementos:

1. Entorno de desarrollo NetBeans
2. Licencia para la herramienta JXBrowser
3. Librerias JXBrowser 6.0
4. API de Google con su Key generada.
5. Documento HTML para incluir la API Key.

# 7. Comparación de la implementación de las tecnologías

Debe incluir al menos una tabla resumen, en sección de página horizontal, cruzando los criterios y los valores de cada tecnología. Con una columna de comentarios sobre la comparación

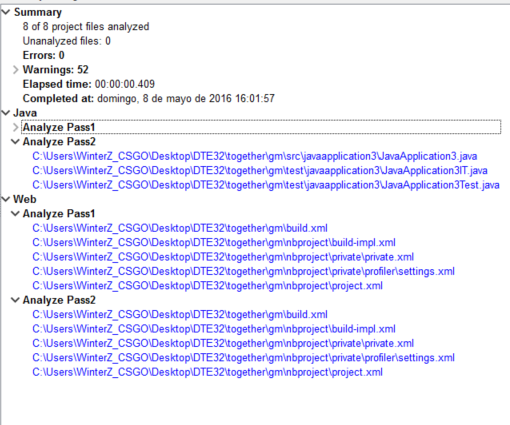
**Métricas y reportes de las aplicaciones**

Para este apartado se ha usado la herramienta Understand de Scitools, una herramienta encargada de analizar todo el código y sacar distintos reportes y métricas que se irán viendo a lo largo de este apartado.

Por otro lado, se ha usado JUnit (plugin propio del IDE Netbeans para analizar posibles errores en el código).

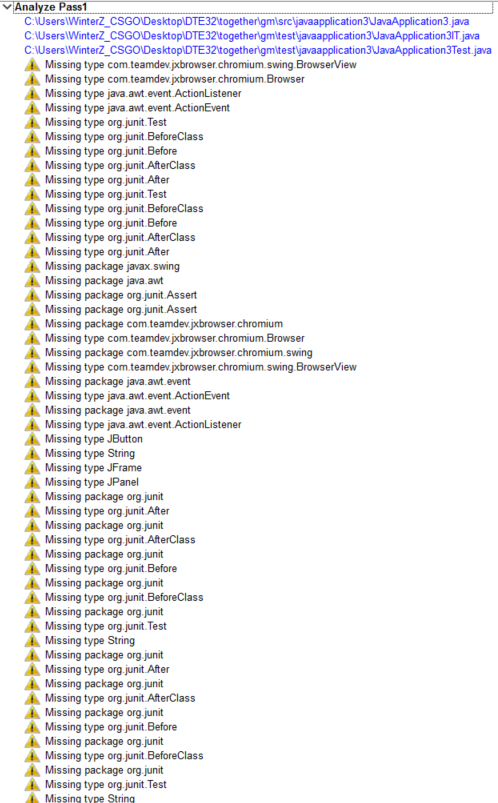
**Google Maps**

Este es el primer análisis generado con la herramienta Understand.

****

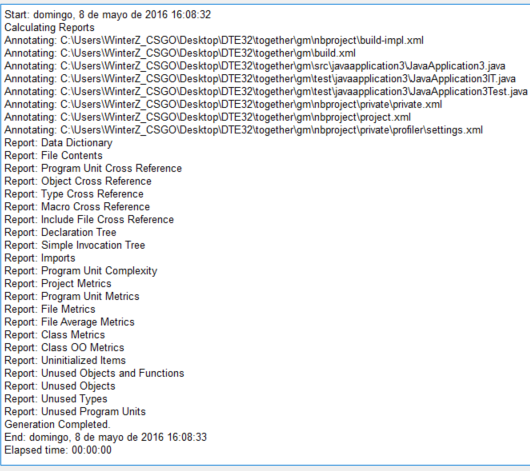
En el proyecto vemos que tenemos 0 errores y 52 advertencias en el primer paso del análisis del proyecto.

A continuación, observamos los warnings que han aparecido:

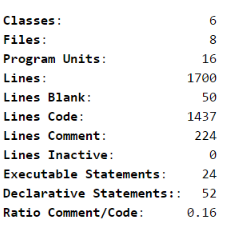


Estos tipos de warning son muy comunes en todo tipo de proyectos, el software observa que faltan imports y tipos de variables que en realidad no hace falta importar.

Generamos un report completo de nuestro proyecto.

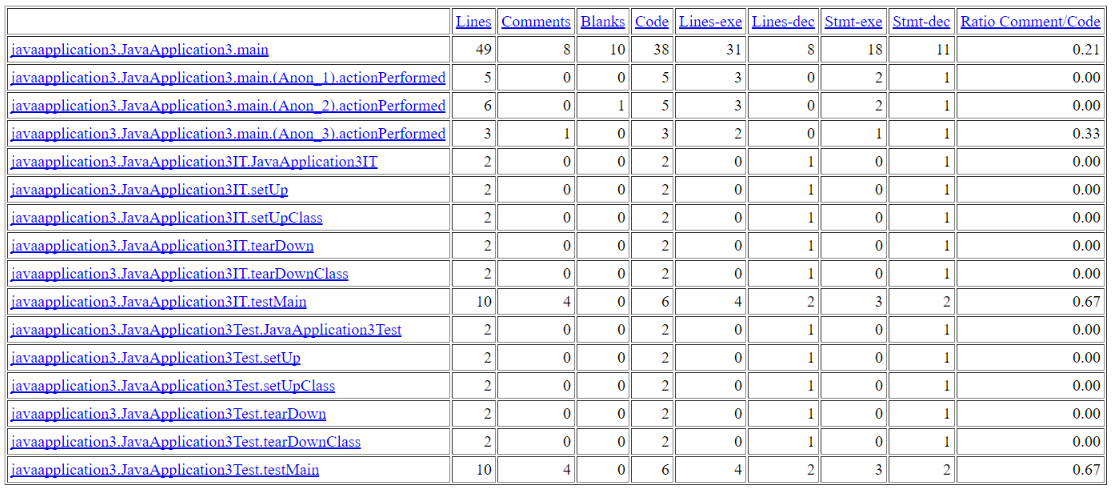


A continuación, un resumen bastante sencillo de entender al respecto del proyecto de Google Maps.

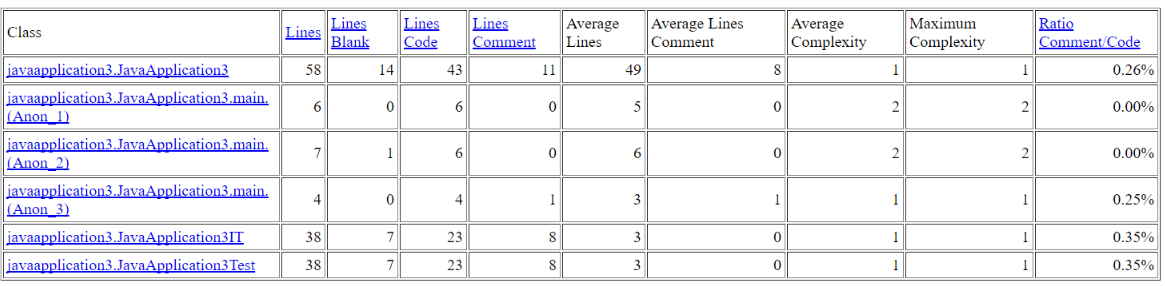


El dato más significativo de este resumen es la ratio de comentarios/código que está en 0.16.

En relación al resumen mostrado, se desglosa en la siguiente tabla sé cómo se distribuye las líneas, el código, y las ratios de comentarios/código.



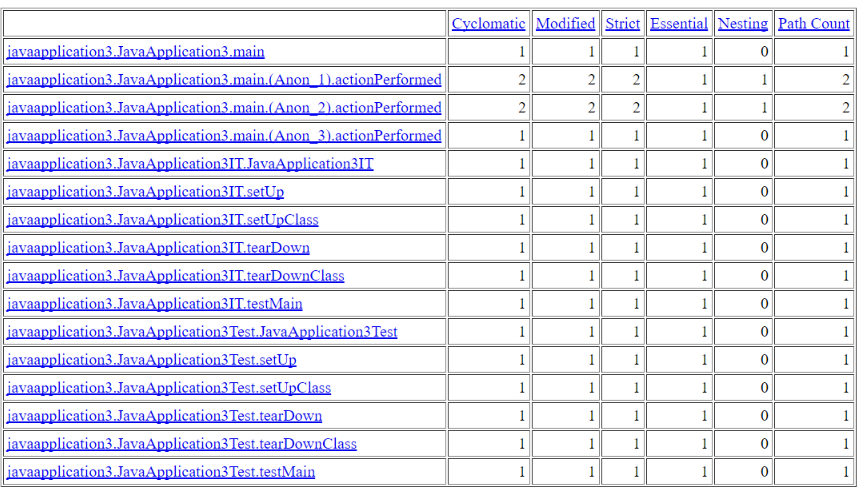
Otra tabla más, describiendo las líneas código y ratio por clases del proyecto.



Los imports usados en esta aplicación son los siguientes:



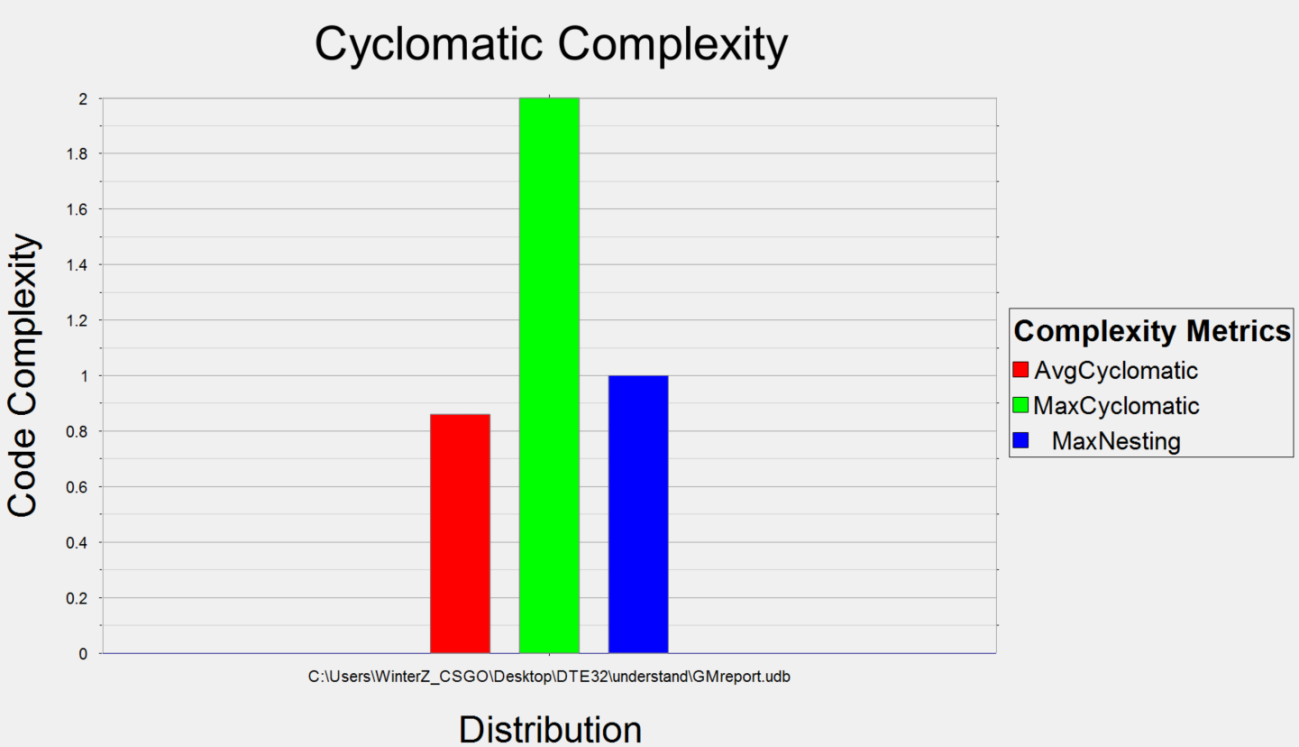
De la tabla de abajo, se ve como la complejidad ciclomatica es mínima (teniendo como valor máximo 2), dato que se considera bueno para la performance de la aplicación.

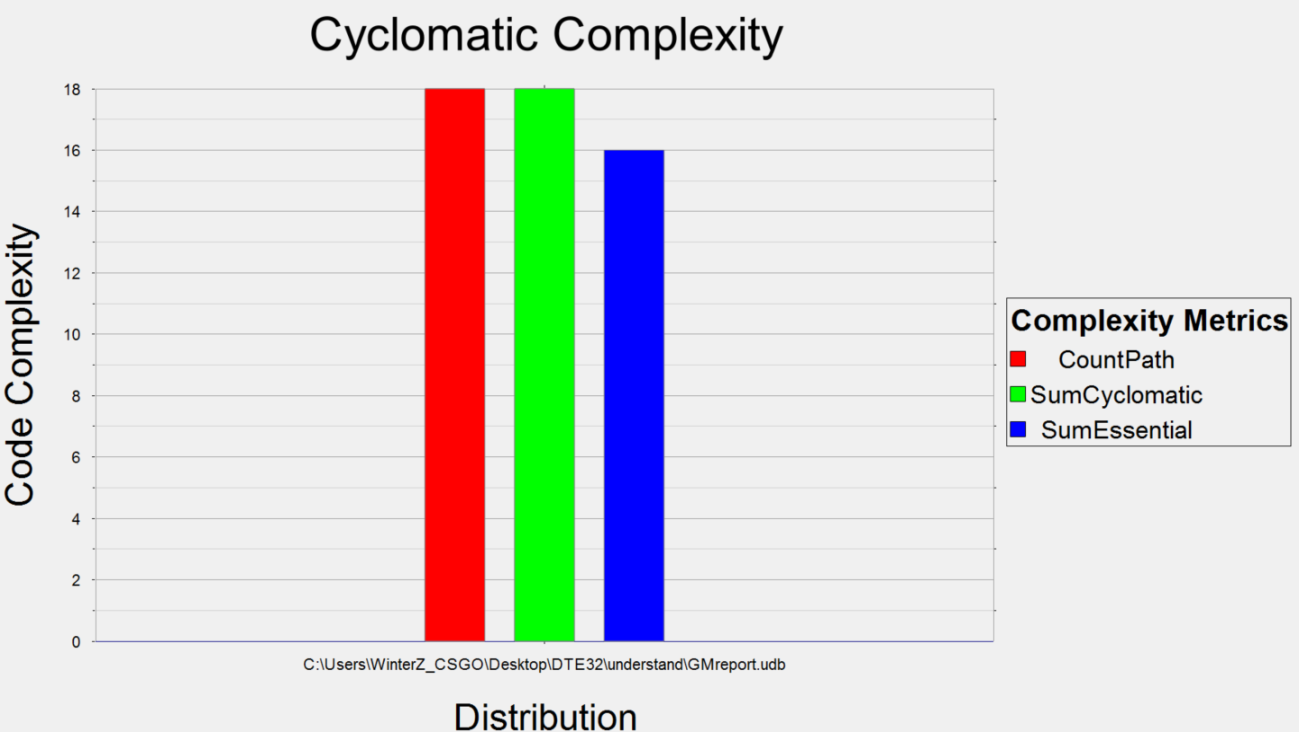


**Complejidad Ciclomática Evaluación del Riesgo**

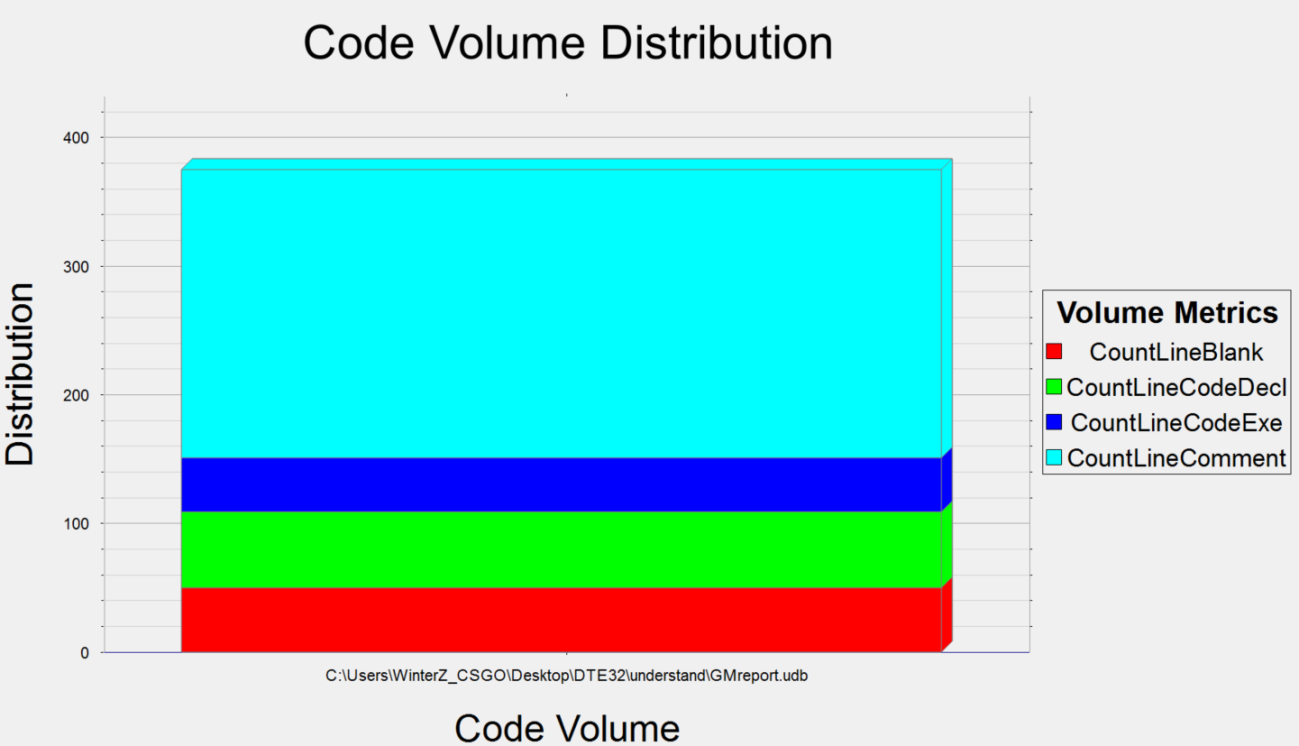
* 1-10 Programa Simple, sin mucho riesgo
* 11-20 Más complejo, riesgo moderado
* 21-50Complejo, Programa de alto riesgo
* 50Programa no testeable, Muy alto riesgo

El gráfico muestra la distribución de la complejidad ciclomática, la media se situa en 0.8 dato que es excelente, siendo el valor máximo 2.



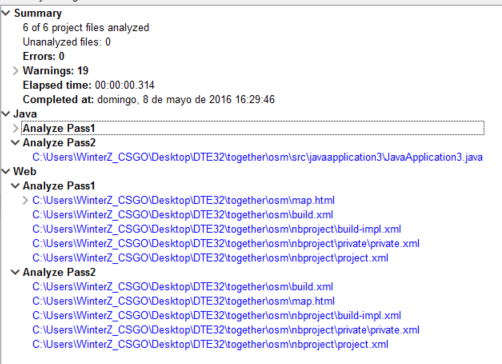


En el gráfico de a continuación, observamos la distribución de las líneas de código:



**OpenStreetMap**

Este es el primer análisis generado con la herramienta Understand.

****

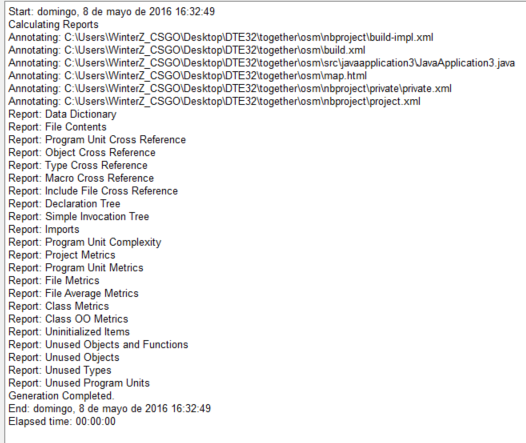
En el proyecto vemos que tenemos 0 errores y 19 advertencias en el primer paso del análisis del proyecto.

A continuación, observamos los warnings que han aparecido:

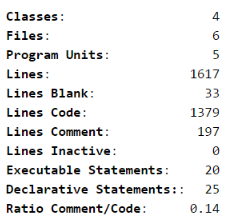


Estos tipos de warning son muy comunes en todo tipo de proyectos, el software observa que faltan imports y tipos de variables que en realidad no hace falta importar.

Generamos un report completo de nuestro proyecto.

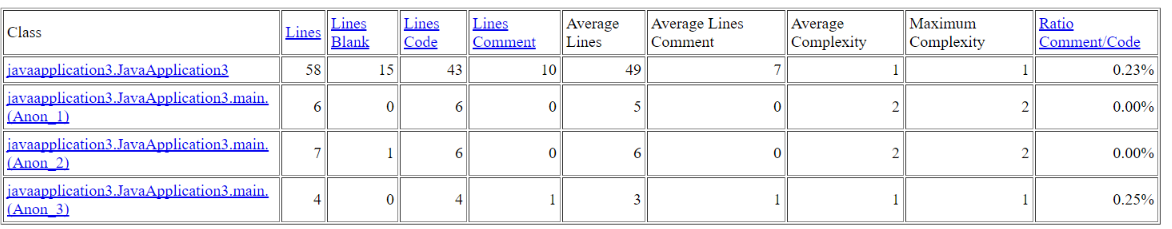


A continuación, un resumen bastante sencillo de entender al respecto del proyecto de Google Maps.

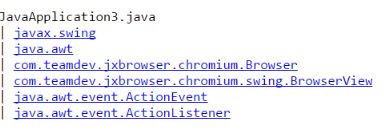


El dato más significativo de este resumen es la ratio de comentarios/código que está en 0.14.

En relación al resumen mostrado, se desglosa en la siguiente tabla sé cómo se distribuye las líneas, el código, y las ratios de comentarios/código.



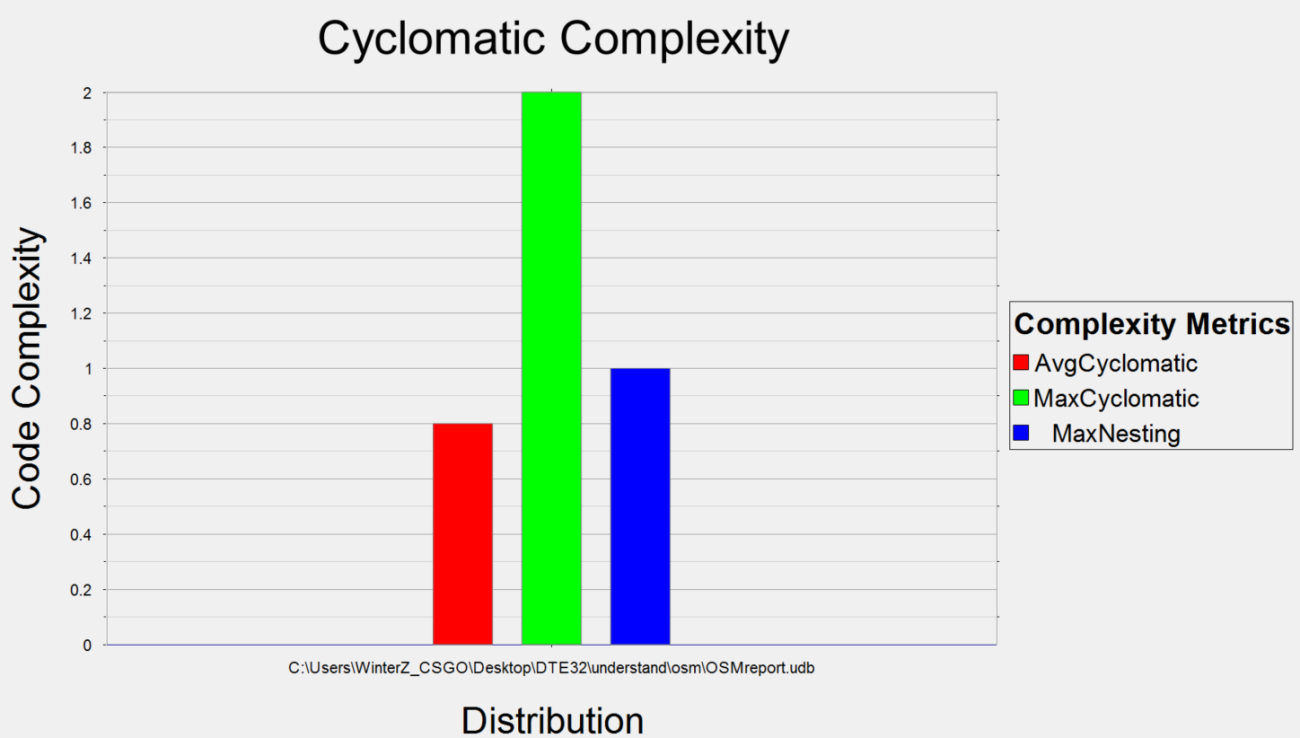
Los imports usados en esta aplicación son los siguientes:

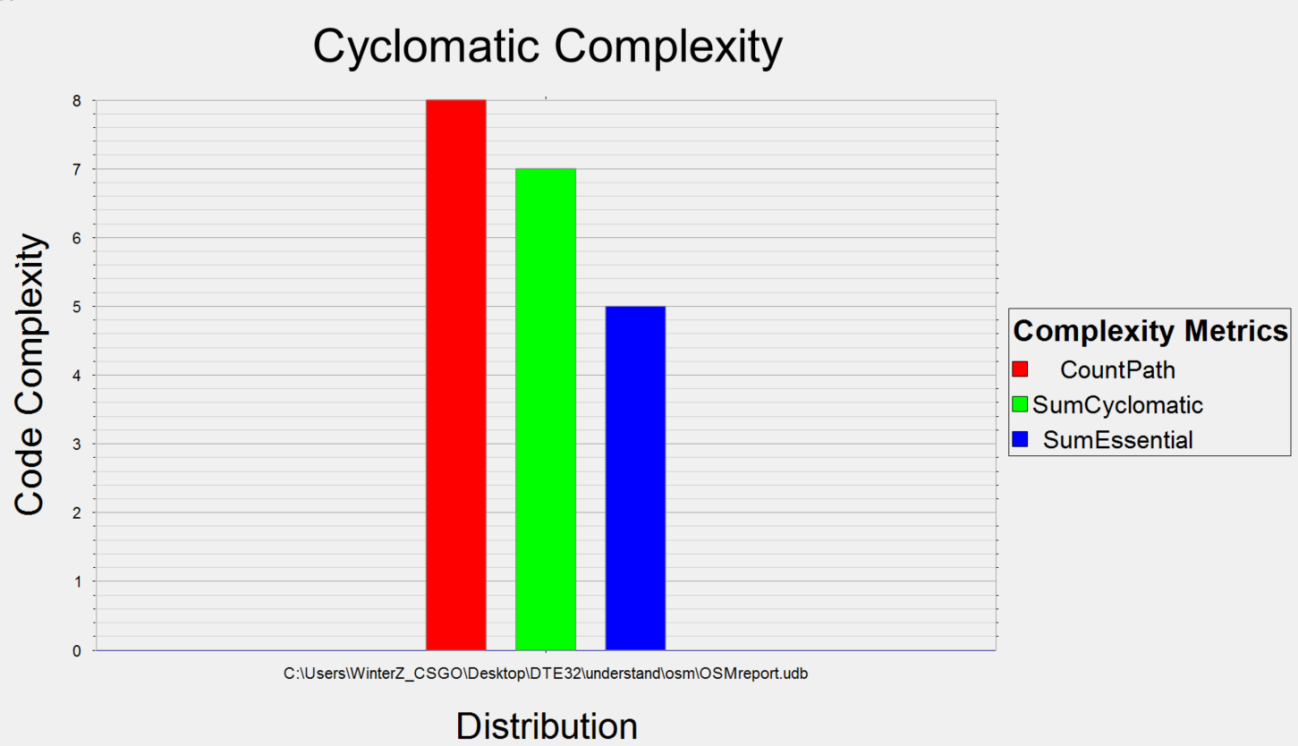


**Complejidad Ciclomática Evaluación del Riesgo**

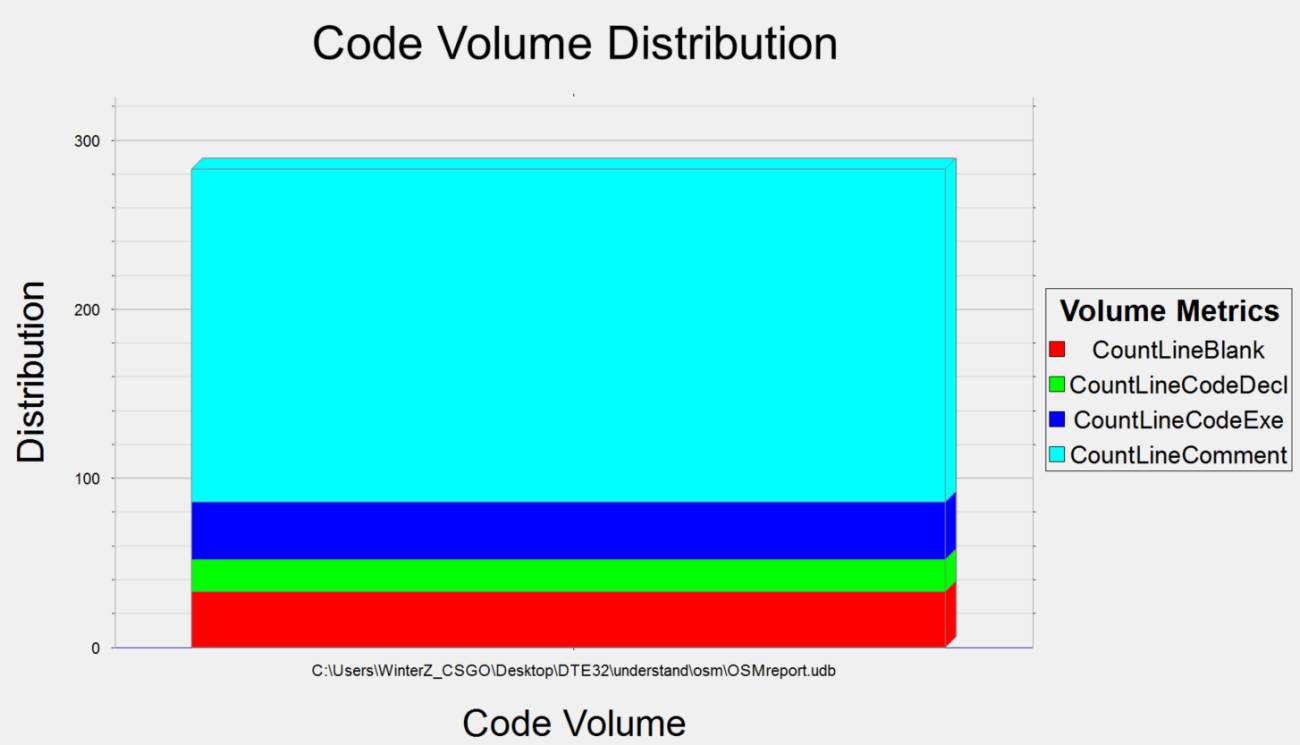
* 1-10 Programa Simple, sin mucho riesgo
* 11-20 Más complejo, riesgo moderado
* 21-50Complejo, Programa de alto riesgo
* 50Programa no testeable, Muy alto riesgo

El gráfico muestra la distribución de la complejidad ciclomática, la media se situa en 0.8 dato que es excelente, siendo el valor máximo 2.





En el gráfico de a continuación, observamos la distribución de las líneas de código:



| **CRITERIOS** | **TECNOLOGÍA OpenStreet Map** | **TECNOLOGÍA Google Maps** | **COMENTARIOS** |
| --- | --- | --- | --- |
| Criterio 1: Compatibilidad con Java | Si | Si | Ambos Web Map Service son compatibles con Java y generan el contenido pedido adecuadamente. **Empate**. |
| Criterio 2: Velocidad de Carga Media | 5.34 | 3.59 | Como podemos observar, la tecnología de Google Maps es casi un segundo y medio más rápido que OpenStreet Map, asi que **vencedor Google Maps.** |
| Criterio 3: Estabilidad del Sistema | 7.6 | 8.3 | En este apartado no hay mucha diferencia entre ambos Web Maps Services, pero es un 9% mejor **Google Maps, asi que victoria para él.** |
| Criterio 4: Concurrencia del Sistema | 7 | 8.5 | Google Maps tiene un 8.5 en este aspecto, por el 7 de OpenStreet Map. Como es un 18% superior **Google Maps, gana en este apartado.** |
| Criterio 5: Recursos Necesarios | 4 | 5 | Si contamos los casos de pruebas, Google Maps necesitaría un recurso más que OpenStreet Map, pero si contamos solo los recursos finales, OpenStreet Map necesitaría solo 3 recursos por los 5 de Google Maps, así que en este caso sale **vencedor OpenStreet Map.** |
| Criterio 6: Intercambio entre Mapas | Si | Si | En este caso, ambos Web Map Service permiten alternar al modo transporte para ver las calles de una manera más gráfica y no por satélite. **Empate**. |
| Criterio 7: Manejo del Mapa | No | Si | Como hemos visto a lo largo del documento, en OpenStreet Map no funcionan los botones de alejar/acercar mientras que en Google Maps si, asi que en este caso, damos como **vencedor a Google Maps**. |
| Criterio 8: Clases | 4 | 6 | El número de clases no es muy significativo, depende de la complejidad de la aplicación a programar el dato puede tomar cierta relevancia. La cantidad de clases en este caso no es un factor clave.  **Empate.** |
| Criterio 9: Archivos | 6 | 8 | Al igual que en el caso de las clases, la cantidad de archivos del proyecto puede ser un factor determinante; en este caso la diferencia radica en los archivos de testeo que se han generado. **Empate.** |
| Criterio 10: Lineas | 1617 | 1700 | Las líneas de código generadas son prácticamente las mismas en ambos proyectos, se han usado las mismas herramientas para la generación de las aplicaciones.  **Empate.** |
| Criterio 11: Comentarios | 197 | 224 | Al tener prácticamente las mismas líneas de código tiene una cantidad de comentarios semejante.  **Empate.** |
| Criterio 12: Ejecutables | 20 | 24 | Cuantos menos ejecutables mejor, ya que significa que necesita menos recursos, de este modo **OpenStreet Map gana.** |
| Criterio 13: Ratio | 0.14 | 0.16 | La ratio de ambas aplicaciones es muy similar, pero en ambos casos se cumple el requisito no funcional (RO2) que pide que la ratio no sea menor a 0.10.  **Empate.** |
| Criterio 14: Complejidad | 0.8 media, 2 máximo | 0.8 media, 2 máximo | Ambos tienen la misma complejidad, por lo tanto damos un **empate**. |
| Criterio 15: Advertencias | 19 | 52 | Tiene bastantes menos advertencias OpenStreet Map que Google Maps, así que damos como **ganador a OpenStreet Map** |
| Criterio 16: Errores | 0 | 0 | Ninguno de los dos tiene ningun tipo de error, asi que **empate.** |

# 8. Conclusiones

A partir de la información incluida en el apartado 7 y de la experiencia al realizar el trabajo, el grupo debe estar en condiciones de manifestar su opinión sobre la implementación del sistema utilizando ambas tecnologías, y debe plasmarla en este apartado, indicando las ventajas e inconvenientes más relevantes de utilizar una u otra tecnología para implementar el sistema.

Al tener GM una API no estamos limitados a la realización de diversas aplicaciones de muy diferente índole, sin embargo, OSM al no disponer de una API nos limita y complica el desarrollo de aplicaciones.

OSM permite realizar modificaciones sobre el mapa de manera más sencilla que GM.

Al utilizar la API de Google Maps nos genera una seria de restricciones a la hora de desarrollar, mientras que OSM nos brinda de mayor libertad.

Google Maps es a priori la mejor opción para usar como web map service, tanto por velocidad como por calidad de los mapas, además la facilidad de la API a la hora de desarrollar lo hacen la opción escogida dada la experiencia que hemos tenido en este trabajo.

---------------------------

(Hay que cumplir la estructura básica indicada de secciones. Pero si se desea se pueden añadir otras secciones como anexos. Por ejemplo, alguna encuesta de opinión realizada sobre las tecnologías, etc.)