Creación de una herramienta de soporte a la autoevaluación de ejercicios de programación

Nicolás Torrell Amado

Resum—El proyecto busca crear una herramienta que permita ayudar a la corrección de código, esta herramienta se encargara de analizar el código y devolver al usuario los datos que ha solicitado, como el numero de funciones totales, cuantas líneas ocupa cierta función, o cuantas y que instrucciones contiene, para ayudar en la parte del estilo y la estructura de la evaluación del código ya que aunque el código de el resultado correcto al ser ejecutado, puede tener fallos en como ha sido realizado.

Paraules clau—código, herramienta, evaluación.

Abstract— The project seeks to create a tool to help in correcting code, this tool will analyze the code and return the user the data requested, such as the number of total functions, how many lines a certain function occupies, or how many and what kind of instructions it contains, to assist in the style and structure part of the evaluation of the code since although the code may have the correct result when executed, it may have mistakes in how it has been developed.

Index Terms—Code, tool, evaluation.

1 Introducción y Objetivos

A la hora de corregir un código se suele utilizar la forma más común de comprobar que el resultado de ejecutar el código es el esperado, se puede hacer manualmente o usando herramientas de autoevaluación, pero el uso de estas herramientas[1] solo comprueba que el resultado sea correcto y no que el código esté bien construido. Para hacer estas correcciones hay que hacerlo manualmente lo que puede ser complicado debido a que el código puede ser muy largo y hay que comprobar cada línea para encontrar fallos ya que no se puede simplemente ejecutar y comprobar el resultado. Estas correcciones manuales podrían tardar varios días lo que retrasaría la corrección del código y se podría dar el caso de no haya sido corregido a tiempo o que varios elementos erróneos no hayan sido detectados y se de por bueno un código con fallos.

Por lo tanto, el proyecto propone la creación de una herramienta que autoevalúe el código de forma automática, la corrección no se basa solo en la ejecución del código y comprobar el resultado con un resultado esperado, sino que se pretende evaluar el estilo del código es el adecuado y que la estructura sea la adecuada. Para conseguirlo hay que comprobar que cada elemento del código se ajuste a unas reglas preestablecidas para comprobar que ha sido realizado de una forma correcta. Esta herramienta puede conseguir una reducción del tiempo de evaluación así como una mejor precisión a la hora de encontrar fallos en el código.

El objetivo principal del proyecto es crear una herramienta que ayude al usuario recorriendo el código de forma automática y muestre los datos que desea ver, así como proporcionar al usuario opciones para detectar ciertos elementos que indique, facilitando al usuario la corrección del código.

Para este proyecto se han definido los siguientes objetivos divididos en principales, secundarios y opcionales:

Objetivos principales:

- Comprobar fallos de estilo: Una de las principales funciones de esta herramienta es que detecte fallos relacionados con como esta escrito el código, ya que a pesar de que el resultado sea correcto hay elementos que pueden considerarse fallos como un break dentro de un while o un for.

● Comprobar fallos de estructura: Otra función principal de la herramienta es también detectar errores en las tabulaciones del código, ya que cada llave que abre ("{") y cierra ("}") las funciones y instrucciones deberían estar en la misma columna. Otra función importante es comprobar que las funciones no sean demasiado largas ya que este dificulta su comprensión.

● Mostrar el numero, tipo y localización de los fallos: Cuando un error se encuentra se tiene que mostrar al usuario donde es el fallo en cuestión y una explicación de porque es considerado un fallo a pesar de que el código se ejecute sin problemas.

● Ofrecer varias opciones para detectar fallos: A la hora de buscar errores se podrá seleccionar que errores buscar, como buscar todos los tipos errores o solo unos cuantos en concreto.

Objetivos secundarios:

● Comprobar resultado esperado: La función principal de la aplicación es comprobar el estilo, pero fácilmente se podría añadir la funcionalidad para que también compruebe que el resultado sea de ejecutar el código sea correcto.

Objetivos opcionales:

● Ofrecer opciones para varios lenguajes: El programa esta pensado para funcionar en C y C++ pero se podría adaptar para otros lenguajes como Java o Python.

A medida que se implementaba la herramienta los objetivos establecidos se consideraron demasiado optimistas ya que fueron realizados antes del comienzo del desarrollo de la herramienta en una etapa donde la idea de como seria la herramienta final estaba poco desarrollada y a la hora del desarrollo los esfuerzos se centraron en cumplir los requisitos ya que fueron creados con mayor conocimiento de como debería ser la herramienta. Los objetivos secundarios y opcionales no se han realizado ya que debido a que eran menos prioritarios y su cumplimiento no era obligatorio ya que no representaban funcionalidades esenciales de la herramienta y el objetivo principal se podía alcanzar sin ellos, se centraron los esfuerzos en otras tareas y no encajaron en la planificación final del proyecto. El desarrollo se centro en los elementos principales que debía realizar la herramienta, por lo que el objetivos de comprobar fallos de estructura no se pudo realizar debido a la forma en la que la herramienta detecta los elementos del código no era posible cumplirlo en el tiempo establecido, esto no impidió el cumplimiento del objetivo principal ni de los demás objetivos y se cumplieron sin problema.

2 MOTIVACIÓN

La motivación principal a la hora de realizar el proyecto era la creación de una herramienta para poder facilitar la corrección de código.

Se quiso desarrollar este proyecto ya que recorrer el código manualmente para buscar fallos puede ser una tarea muy complicada si el código es muy largo o contiene muchos fallos, por lo que se quería reducir el trabajo necesario a la hora de corregir código para que los usuarios pudieran ahorrar tiempo y para que pudieran encontrar los fallos con mayor precisión, por lo que la herramienta contiene mucha utilidad para los usuarios que deseen utilizarla.

3 ESTADO DEL ARTE

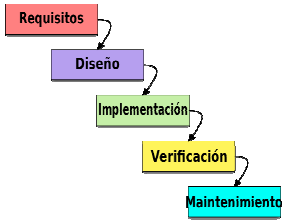
Actualmente no existe ninguna herramienta capaz de recorrer el código y ayudar al usuario a detectar los fallos, existen herramientas que ayudan a corregir el código, pero estas herramientas solo ayudan comprobado que el resultado esperado al ejecutar el código es el correcto, esto se realiza comparando la respuesta esperada con el output del código y entonces se determina que puntuación se le da el código dependiendo de la similitud de las respuestas.

La herramienta que se desarrolla en este proyecto esta pensada para trabajar conjuntamente con las demás herramientas existentes ya que se complementarían mutuamente ya que cada una realizaría una parte que la otra no es capaz de realizar, por lo que no se crea con objetivo de substituir las herramientas existentes si no de dar al usuario más opciones a la hora de corregir código que no sea solo comprobar el resultado al compilar.

4 METODOLOGÍA Y PLANIFICACIÓN

4.1 Metodología

Se utilizara la metodología de cascada, esta tipo de desarrollo se caracteriza por la secuencialidad de las fases de desarrollo, es decir, el principio de una de las etapas debe esperar a la finalización de la anterior para poder comenzar. Al final de cada etapa se revisa el trabajo realizado para comprobar si esta listo para avanzar a la siguiente etapa, si el trabajo realizado no se considera que esta listo no se cambia de etapa y se revisa el trabajo para determinar que partes hacen que no este preparado. Las etapas se dividen en: requisitos, diseño, implementación, verificación y mantenimiento.



Esquema del desarrollo en cascada.

Se escogió esta metodología ya que es fácil de entender y implementar, ya que cada fase esta bien diferenciada del resto y porque se adapta bien a las necesidades del proyecto a realizar ya que el trabajo a realizar se puede dividir entre las fases y ya que al ser un proyecto relativamente pequeño de una persona otras metodologías como Scrum no fueron considerados ya que estas metodologías funcionan mejor en equipos de varias personas y proyectos más grandes que requieran la división del trabajo entre varios integrantes del equipo y su implementación en este proyecto solo complicaría la planificación del proyecto.

4.2 Planificación

A la hora de realizar la planificación se ha considerado que se hará en cada fase del desarrollo, la fase de mantenimiento no se considero en la planificación ya que esta se realizaría cuando la herramienta sea finalizada y en la planificación solo se considera el desarrollo:

● Informe inicial: Se realizo la definición del problema a resolver y la definición de los objetivos del proyecto.

● Requisitos y valoración de herramientas a usar: Se realizo la entrevista de recopilación de requisitos con el tutor para determinar las funciones de la herramienta y se investigo posibles herramientas secundarias que podrían ayudar con el desarrollo.

● Diseño: Se contemplaran las diferentes tecnologías que pueden ser usadas, se estudiara si pueden ser usadas o no y se pensara el lenguaje y la plataforma y los módulos que tendrá la herramienta final.

● Implementación: Se realizara la programación principal del proyecto en base a las tecnología escogidas en la fase de diseño.

● Verificación: Se testeara que el código es correcto y devuelve los resultados esperados utilizando diferentes códigos de prueba que representes diferentes casos posibles que se pueden encontrar, esta fase junto con implementación son las más importantes ya que es donde se realizara la mayor parte del trabajo.

Se estableció el siguiente horario de trabajo teniendo en cuenta los plazos de entrega del proyecto:

	Fecha inicio	**Fecha fin**
Informe inicial	6 de marzo de 2017	19 de marzo de 2017
Requisitos y valoración de herramientas a usar	19 de marzo de 2017	9 de abril de 2017
Implementación	10 de abril de 2017	21 de mayo de 2017
Verificación	22 de mayo de 2017	27 de junio de 2017

Este horario se modifico ya que no se considero que la parte de diseño de la herramienta supusiera tanta carga de trabajo, el horario modificado se muestra a continuación:

	Fecha inicio	**Fecha fin**
Informe inicial	6 de marzo de 2017	19 de marzo de 2017
Requisitos y valoración de herramientas a usar	19 de marzo de 2017	9 de abril de 2017
Diseño preliminar	10 de abril de 2017	21 de mayo de 2017
Implementación y Verificación	22 de mayo de 2017	27 de junio de 2017

A pesar del cambio en el horario de planificación los plazos establecidos para cada fase no se vio afectada y se cumplieron plazos de entrega de la herramienta y se desarrollo acorde con la fechas establecidas sin ningún otro retraso ya que el tiempo requerido para la fase de implementación fue reducido al realizar el diseño y el tiempo de la fase de verificación se realizo una estimación muy optimista por lo que el tiempo necesario para la verificación era menor del esperado.

5 Requisitos

A la hora de definir las funcionalidades se ha realizado un análisis de requisitos para la herramienta, estos requisitos muestran las funcionalidades que la herramienta debe realizar y sirven de base para saber como proceder adecuadamente con el desarrollo final de la herramienta, a continuación se muestran los requisitos recopilados:

● La aplicación permitirá al usuario seleccionar que elementos quiere que sean comprobados y evaluados.

● La aplicación podrá determinar que una función tiene que ser realizada con una instrucción en concreto y marcará error aunque la función se ejecute correctamente.

● La aplicación permitirá al usuario analizar y buscar fallos en el programa entero.

● La aplicación podrá descomponer el código en las diferentes funciones que lo componen.

● La aplicación permitirá al usuario seleccionar qué número de funciones que desea evaluar.

● La aplicación permitirá comparar el código con una solución propuesta.

● La aplicación podrá contar el número de instrucciones(if, while, for...) del programa y lo comparará con el número de la solución propuesta.

● La aplicación podrá contar el número de líneas de código del programa y lo comparara con el número de la solución propuesta.

● La aplicación podrá comprobar que el número de variables locales y sus tipos se ajusten a la solución propuesta.

6 DISEÑO

A la hora de plantear el desarrollo de la herramienta se buscaron herramientas que pudieran ayudar con la implementación del código, la finalidad de estas herramientas es ayudar a la descomposición estructural del código a analizar para así tener los diferentes elementos a comprobar separados y así facilitar su análisis posterior. Algunas de las herramientas consideradas fueron las siguientes:

● GCC-XML[2]: Es una herramienta que parsea el código transformándolo en XML.

● ANTLR[3]: Es una herramienta con la que se pueden reconocer estructuras semánticas y sintácticas del código.

● Vera++[4]: Es una herramienta para verificar, analizar y transformar código.

● Cxxchecker[5]: Es una herramienta que escanea el código y comprueba consistencia de nombres y otras practicas de programación comúnmente aceptadas.

● Xerces-C++ [6]: Es una herramienta que parsea el código transformándolo en XML.

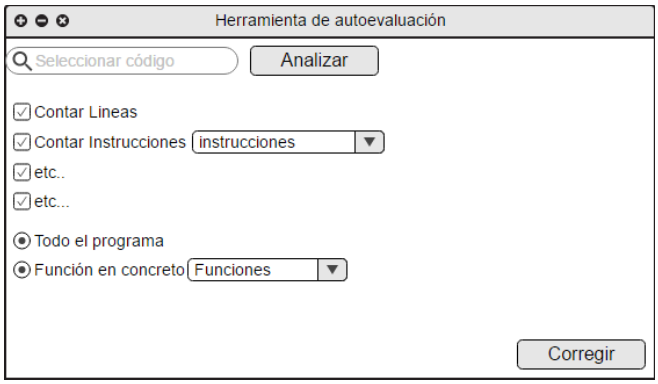
● Spirit [7]: Es una herramienta que parsea el código.

● CSCOPE [8]: Es una herramienta que busca diferentes elementos del código como definiciones globales, funciones que llaman a otra función, etc.

● Ctags [9]: Es una herramienta que genera etiquetas de los objetos creados en un código para que sean fácilmente localizables.

La utilización de las herramientas buscadas para ayudar al desarrollo fue desechada ya que su finalidad final seria obtener un archivo de código de entrada y descomponerlo en elementos como nombres de funciones o variables. Pero las herramientas no eran capaces de descomponer el código en elementos utilizables para ser analizados, o no tenían la utilidad que se pensó que tendrían inicialmente o no guardaban un registro que indicara a que función pertenecía cada elemento y por lo tanto era necesario buscar en el código a que función pertenecía cada elemento lo que hacia que no tuvieran utilidad real ya que no evitaban recorrer el código que era su funcionalidad principal.

Al diseñar la herramienta se realizo un mockup de la posible interfaz grafica que tendría la herramienta para poder tener una idea de que como se debería encaminar el desarrollo, la mockup es mostrada a continuación:

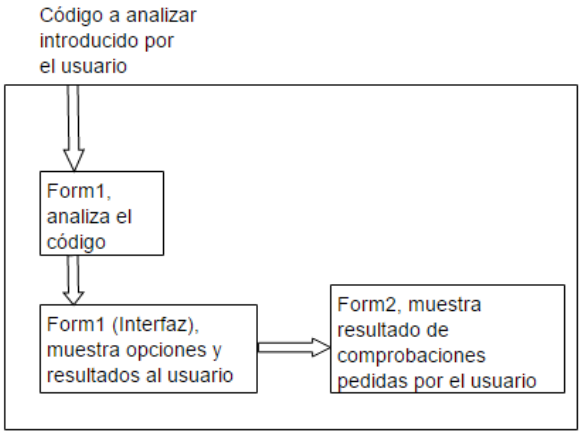


En este mockup se ve un diseño con los elementos básicos de la herramienta y ayudo en la creación de la interfaz grafica final usándola como base para saber las relaciones entre los elementos del código.

La herramienta recibirá como input un archivo C++ que será seleccionado por el usuario el menú de la herramienta, después el usuario podrá seleccionar las opciones que quiere que se muestren y sobre que función mostrarla.

La herramienta esta formada por cuatro elementos, el código de la funcionalidad principal del programa(llamado Form1) que va en conjunto con el código de la interfaz principal y el código de una ventana secundaria que muestra resultados de comprobaciones pedidas por el usuario(llamado Form2).

A continuación se muestra un esquema del diseño de la herramienta, donde se aprecia la relación entre los diferentes elementos que la componen:



En el esquema se muestra como el código principal de la herramienta una vez acabado mando los datos a la interfaz para ser mostrados y como de la interfaz principal se puede acceder a las opciones para mostrar la ventana de los resultados de las comprobaciones.

7 IMPLEMENTACIÓN

En principio a la hora de implementar la herramienta se decidió implementarla en C++ en Visual Studio 2015 utilizando la herramienta ANTLR, unas de las herramientas buscadas durante la fase de diseño, pero ANTLR y las demás herramientas consideradas no cumplían con las expectativas establecidas para ellas y se desecho su utilización. Después de desechar la herramientas de apoyo la herramienta se empezó a desarrollar en C# ya que ofrecía más y mejores posibilidades que C++, como la facilitación en la creación de una interfaz de usuario y más opciones a la hora de escribir el código.

En la interfaz final(Apéndice A) se muestra los datos y opciones disponibles al usuario, mientras no haya cargado un archivo los campos que muestran los outputs están vacíos y el apartado para indicar los resultados esperados esta invisible. Una vez el usuario ha utilizado el botón de abrir archivo y seleccionado un archivo, la herramienta comprobara que el archivo es del tipo correcto y procederá al análisis si es correcto y mostrara un error si no lo es. Los primeros pasos son la eliminación del texto de los comentarios y strings ya que estos podrían contener elementos que causarían la falsa detección de elementos ya que podría darse el caso de encontrar una función antigua o en desuso que este comentada por lo que esta no debería ser considerada ya que no esta siendo utilizada por el código analizado, en el caso de strings se podría dar el caso de que contenga palabras clave que la herramienta detectaría, lo que produciría posibles fallos de detección o una ralentización en la ejecución.

Una vez se ha limpiado el código del texto dentro de comentarios y strings, la herramienta transformara todo el código en una solo línea ya que así se facilita el análisis de todo el código en busca de las funciones implementadas, ya que se puede dar el caso de salto de líneas entre el nombre de una función y su tipo, que es valido en a la hora de compilar pero dificulta comprobar que ambos elementos estén relacionados. Al estar todo el código en una línea también resulta más optimo ya que en el caso de ir línea a línea habría de recorrer caracteres individuales igualmente para buscar ciertos elementos y sus posiciones por lo que se tendrían que recorrer el código más veces aumentando el posible tiempo de ejecución en códigos muy grandes. Otra mejora de optimización que ocurre al recorrer una sola línea es que se puede ahorrar variables de control que tendrían que comprobar si cierto elemento tiene relación con algún otro de la línea anterior.

Una vez realizado esta tarea la herramienta recorrerá todo el archivo y buscara las funciones, mediante un método recursivo que buscara elementos en el archivo que puedan ser posibles implementaciones de funciones comprobando que todas la partes que la componen sean correctos, la primer elemento que comprueba es que debe contener paréntesis y que el contenido sea correcto, para este caso utilizara otra función que recibirá el texto que hay entre los paréntesis y recursivamente analizara cada parámetro que encuentre hasta que no haya más. Una vez que los paréntesis sean comprobados y dados por buenos se comprobara que el nombre de la función sea correcto y que tiene corchetes de apertura y cerramiento. Una vez encontrada una función esta guarda su nombre, texto completo de la función, tipo, texto entre paréntesis y el texto entre las llaves en una clase que guarda los datos de las funciones y, si pertenece a una clase, se guardara a que clase pertenece para su posterior análisis y entonces se cortara el texto de la función del texto total del código a analizar y este será recorrido otra vez hasta que no encuentre más funciones.

Una vez el análisis haya acabado en la interfaz se mostrara los datos del archivo que contendrá el nombre, el numero de funciones y el numero de clases, también se encontrara en medio de la interfaz el código analizado como referencia para poder fácilmente ver el código que se esta analizando sin tener que abrir el archivo, una lista de funciones y una lista de las clases en las que aparecerán las clases a las que pertenecen ciertas funciones.

Al seleccionar una clase en la lista de clases se mostrara las funciones que pertenecen a esa clase y también se mostrara en el apartado datos de clase su nombre y numero de funciones que están implementadas en el archivo, por defecto se muestran todas las funciones que hay en el código.

Al seleccionar una función se calcularan y mostraran los datos seleccionados en la configuración general del análisis(Apéndice B), los datos se mostraran en el apartado datos de función y solo serán mostrados aquellos que hayan sido seleccionados por el usuario en la configuración

La herramienta permite al usuario definir si una función debe contener cierto numero de elementos, como las diferentes instrucciones, o especificar que la función no contenga otros elementos, mediante un input donde se establecerá el numero deseado y apretando el botón para comprobar se mostraran los resultados en otra ventana donde se indicara si la función tiene un mayor o menor numero del indicado en el input(Apéndice C), la ventana solo mostrara los datos si el usuario a introducido un valor, si no es introducido ningún valor no se comprobara y se realizaran las demás comprobaciones sin tener en cuenta los campos vacíos.

8 CONCLUSIÓN

El programa desarrollado con este proyecto tenia como finalidad crear una herramienta que ayudara con la evaluación de ejercicios de programación, no en la parte de compilar el código y comprobando que el resultado sea correcto, sino en la parte de recorrer el código para comprobar que ciertas partes están hechas de cierta manera. Se buscaba crear una forma de automatizar la inspección del código para facilitar su evaluación y también reducir el tiempo necesario.

La herramienta final cumple con los requisitos establecidos y realiza la finalidad final del proyecto de ayudar con la evaluación.

Hay ciertos elementos que no han estado establecidos dentro de la definición y alcance del proyecto pero que podrían ser considerados en futuras actualizaciones, una de estos elementos es la definición de una métrica para dar una puntuación a los resultados obtenidos al comprobar el código analizado con los resultados esperados que define el usuario.

Como consideraciones futuras se podrían ampliar la herramienta con diferentes funcionalidades que o no han podido ser implementadas o que no fueron consideradas, como la modificación de la herramienta para poder detectar fallos de estructura como tabulaciones, hacer que la herramienta ejecute el código y compruebe el resultado o modificarla para que funcione con otros lenguajes.

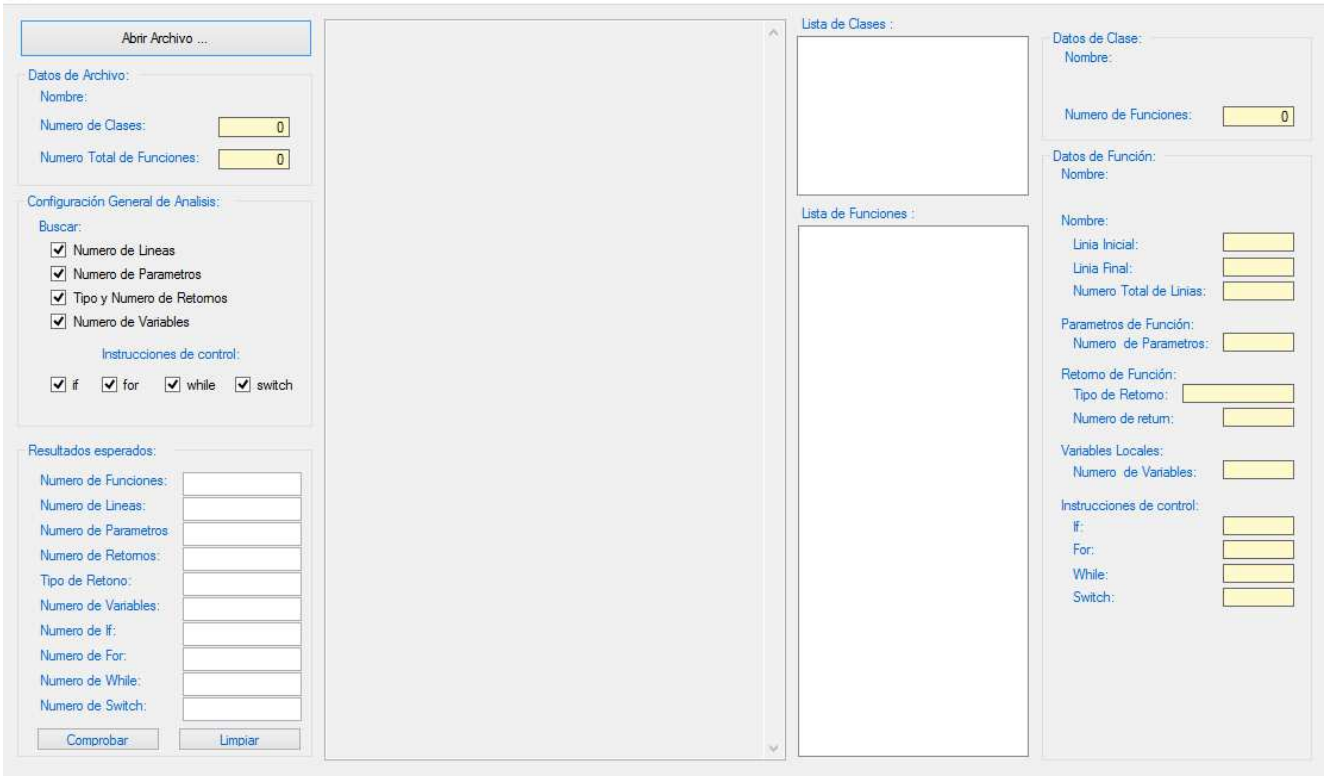
BIBLIOGRAFIA

[1] Virtual programming lab "http://vpl.dis.ulpgc.es/index.php/about/what-is-vpl"
[2] GCC-XML "http://gccxml.github.io/HTML/Index.html"
[3] ANTLR "http://www.antlr2.org/pccts133.html"
[4] Vera++ https://bitbucket.org/verateam/vera/wiki/Home"
[5] Cxxchecker "https://gna.org/projects/cxxchecker"
[6] Xerces-C++ "https://xerces.apache.org/xerces-c/"
[7] Spirit "http://boost-spirit.com/home/"
[8] CSCOPE "http://cscope.sourceforge.net/"
[9] Ctags "http://ctags.sourceforge.net/"

APÉNDICE

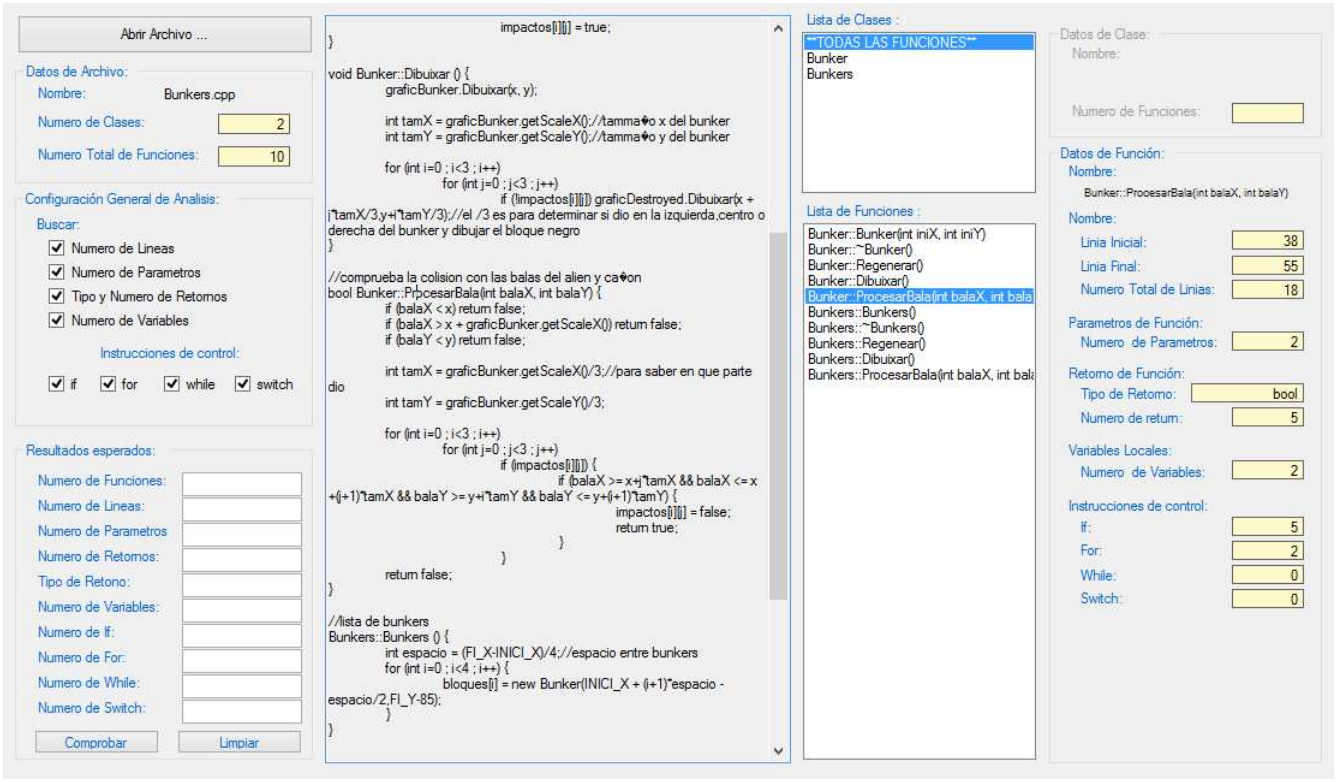
A. APÉNDICE

Interfaz de la herramienta:



B. APÉNDICE

Interfaz de la herramienta mostrando datos del archivo y de la funcion Bunker::ProcesarBala en Bunkers.cpp del juego Space Invaders:



C. APÉNDICE

Interfaz de la herramienta mostrando todas las comprobaciones en la funcion Bunker::ProcesarBala en Bunkers.cpp del juego Space Invaders: