



# Análisis, Diseño e Implementación de un Prototipo de Transpilador para Interceptar Código

Grado en Ingeniería Informática

Curso: 2016 - 2017

Autor: Adrián Fernández Rodríguez

Tutor: Alejandro Calderón Mateos

Universidad **Carlos III** de Madrid



# Índice de Contenidos

Índice de Contenidos.....	3
Índice de Ilustraciones .....	5
Índice de Tablas .....	7
Agradecimientos .....	9
Abstract .....	11
Capítulo 1 Introducción .....	21
1.1.- Motivación .....	21
1.2.- Objetivos.....	23
1.2.1.- Objetivos Principales .....	24
1.3.- Resto del documento.....	25
Capítulo 2 Estado de la Cuestión .....	27
2.1.- Comparativa de las diferentes herramientas.....	27
2.2.- Estudio de la Actualidad.....	28
Capítulo 3 Análisis, diseño, implementación e implantación.....	37
3.1.- Ciclo de Vida.....	37
3.2.- Marco Regulador.....	39
3.3.- Metodología.....	40
3.4.- Análisis .....	42
RX-XX .....	42
3.4.1.- Requisitos Funcionales .....	44
3.4.2.- Requisitos No Funcionales .....	47
3.2.3.- Matriz de Trazabilidad.....	48
3.5.- Diseño.....	51
3.5.1.- JavaCC [19] .....	51
3.5.2.- Yacc [20] y Lex [21] .....	52
3.5.3.- Bison [24] y Flex [25] .....	54
3.5.4.- ANTLR [27] [28] .....	57
3.5.5.- Comparativa con las distintas soluciones y la propuesta .....	60
3.5.6.- Casos de Uso .....	63
3.6.- Implementación .....	67
3.7.- Implantación .....	73
Capítulo 4 Evaluación.....	75
4.1.- Plan de Pruebas.....	75
Capítulo 5 Planificación y Presupuesto .....	83
5.1.- Planificación.....	83
5.2.- Presupuesto.....	87
5.2.1.- Alcance del Proyecto.....	87

5.2.2.- Coste del Personal .....	87
5.2.3.- Coste de los Equipos de Desarrollo.....	88
5.2.4.- Coste de las licencias y otros costes .....	89
5.2.5.- Coste Total del Proyecto.....	89
5.3.- Impacto Socio-Económico .....	90
Capítulo 6 Conclusiones y Trabajos Futuros.....	91
6.1.- Conclusiones.....	91
6.1.1.- Producto .....	92
6.1.3.- Personales .....	92
6.2.- Trabajos Futuros .....	93
Anexo I: Manual de Instalación del Entorno de Desarrollo.	95
Anexo II: Tablas para Impresión .....	97
Anexo III: Repositorio.....	103

# Índice de Ilustraciones

Ilustración 1 Crecimiento del N° de Aplicaciones en las Principales Tiendas .....	22
Ilustración 2 Ranking de los Lenguajes de Programación más usados según GitHub [2] .....	29
Ilustración 3 Estadísticas de la Comunidad que envuelve cada lenguaje [3] .....	30
Ilustración 4 Ventas Actuales y Previsión de Dispositivos Tecnológicos	31
Ilustración 5 Preferencias de los usuarios de Smartphone.....	32
Ilustración 6 Resultados de un test de batería ejecutando Asphalt 8 ....	33
Ilustración 7 Proceso que realiza la memorización .....	35
Ilustración 8 Autómata Finito de estados Lex .....	53
Ilustración 9 Resultado Modo Debug Flex.....	55
Ilustración 10 Detección de Errores de Diseño.....	57
Ilustración 11 Resultado Autómata Bison .....	57
Ilustración 12 Ejemplo Depuración Gramática.....	59
Ilustración 13 Ejemplo de Selección de Reglas a Analizar .....	59
Ilustración 14 Árbol sintáctico generado por una gramática LL [29] .....	61
Ilustración 15 Árbol sintáctico generado por una gramática LR [29].....	62
Ilustración 16 Árbol generado por ANTLR.....	68
Ilustración 17 Ejemplo de Modificación de la Gramática.....	68
Ilustración 18 Ejemplo de Modificación de la Gramática.....	69
Ilustración 19 Ejemplo de Modificación de la Gramática.....	69
Ilustración 20 Detección de la etiqueta correspondiente a la memorización .....	69
Ilustración 21 Ejemplo de Cabecera de Método .....	70
Ilustración 22 Ejemplo de funcionamiento boolean.....	71
Ilustración 23 Paso 1 en la detección de los argumentos .....	71

Ilustración 24 Paso 2 en la detección de los argumentos.....	72
Ilustración 25 Versión Java.....	73
Ilustración 26 Planificación Estimada.....	85
Ilustración 27 Planificación Rea .....	85
Ilustración 28 Planificación Estimada para Impresión .....	99
Ilustración 29 Planificación Real para Impresión.....	100

# Índice de Tablas

Tabla 1 Equivalencia TFG - Métrica V3 .....	41
Tabla 2 Ejemplo de especificación de requisito .....	42
Tabla 3 RF 1 .....	44
Tabla 4 RF 2 .....	44
Tabla 5 RF 3 .....	44
Tabla 6 RF 4 .....	44
Tabla 7 RF 5 .....	45
Tabla 8 RF 6 .....	45
Tabla 9 RF 7 .....	45
Tabla 10 RF 8 .....	45
Tabla 11 RF 9 .....	46
Tabla 12 RF 10 .....	46
Tabla 13 RF 11 .....	46
Tabla 14 RF 12 .....	47
Tabla 15 RF 1 .....	47
Tabla 16 RNF 2 .....	47
Tabla 17 RNF 3 .....	48
Tabla 18 RNF 4 .....	48
Tabla 19 RNF 5 .....	48
Tabla 20 Matriz de Trazabilidad .....	50
Tabla 21 Entorno de Pruebas .....	76
Tabla 22 Tabla de Ejemplo Pruebas .....	76
Tabla 23 Prueba 01 .....	77
Tabla 24 Prueba 02 .....	77
Tabla 25 Prueba 03 .....	78
Tabla 26 Prueba 04 .....	79

Tabla 27 Prueba 05 .....	80
Tabla 28 Prueba 06 .....	81
Tabla 29 Prueba 07 .....	82
Tabla 30 Planificación del Proyecto .....	87
Tabla 31 Salarios por Rol al Mes .....	87
Tabla 32 Cuota Patronal .....	88
Tabla 33 Coste total del Personal .....	88
Tabla 34 Costes de los Equipos y Amortización .....	88
Tabla 35 Coste de las Licencias Software .....	89
Tabla 36 Otros Costes .....	89
Tabla 37 Coste Total del Proyecto IVA Incluido .....	89

# Agradecimientos

“Han de saber decidí no dar nombres,  
por si alguno se quedaba en el tintero.  
Asique va por las mujeres y los hombres,  
por todos aquellos que me quieren y que quiero” [1]



# Abstract

In the present section, it is intended to present a summary version of the document with the objective of discussing about the most remarkable aspects.

In order to be able to present a cleaner and clearer structure, a chapter structure, in which different parts of the final project are covered has been chosen to organize the content.

## Chapter 1: Introduction.

In this chapter, the aim is to explain in an instructive and appealing way the reasons why this final project has raised, and the objectives we intend to reach in order to solve the proposed problem.

At the same time, the structure designed for this document and its content will be briefly exposed in each chapter.

### 1.1. – Motivation

In the 21st century we can affirm without any doubt that we live in the digital information age, in which technology has changed the human perception of life and all that surrounds us.

**Technological companies have become the most important pillar in the world economy**, which can be easily noticed if attention is paid to how the stock market listing of companies like Google or Apple is changing nowadays.

Looking into this explosive growth, there is a remarkable booming market based on smartphones transforming our lives to the extent that the iPhone presentation is to be considered as a historical fact.

**This fact has brought with it a fierce competence between the two main OS** that try to lead the field of the mobile phones market, iOS & Android. This two OS offer to the user a huge range of applications in their respective app stores, which try to be a part of the lives of millions and millions of users that make use of these systems on a daily basis. On the other hand, there are new intrinsic needs both for the proprietary and

third-party applications, as well as for the development of the OS, which can be summarized as it follows:

- **The constant improvements in the Operating Systems**, which are supposed to make user's life easier and make them feel attracted to their systems, causes that the applications must adapt those enhancements to show a strong identity of the service.
- **Security problems in the OS and APIs**, due to the fact that while technological independency is increasing there is an increasing need of keeping information secure.

The second fact has as a consequence that a part of the developers, which could be improving the application or operating system, are actually making efforts to keep the code clean and avoid security breaches that may put in risk the application and affect the final users. This effort means, thus investing money in maintenance.

The required maintenance is the basis of this final project. **Rewriting the code is the most proper solution for a situation in which developers need to save time in order to be able to spend it in other tasks such as those related to improvement in the software.** Likewise, time saved means money saved.

There are also some optimisation techniques that could be applied to some parts of the code without investing developers' time in them.

That's the reason why the current final project, *Analysis, design and implementation of a prototype for a programming language transpiler to intercept code*, is being presented. Its main aims come as follows:

## 1.2.- Objectives

As already discussed, **it is intended to use a programming language transpiler**. It is a tool that enables the transformation of an input code into a code with another output language previously chosen. Those tools are known as source-to-source, due to the fact that they are a programming

language transpiler that instead of transforming code into machine language try to transform it into another language.

In this final project, **instead of transforming the code into another language, there will be a rewriting of specific parts chosen by the developer**. In order to reach this objective, three stages of the compilation process will be taken into account:

- **Lexical analysis:** It is the first part of the analysis. Data is packaged through regular expressions in the smallest unit in this field, the ‘token’.
- **Parsing:** Once the data has been transformed into tokens, a specific syntax tree is created. It is generated from rules that create a grammar with no context, establishing the hierarchy and order in which tokens must appear.
- **Semantic analysis:** Finally, in this part of the analysis, once the specific syntax tree has been obtained, a series of attributes which are necessary to make checks or create actions that are not possible during the parsing is added to the rules.

#### 1.2.1.- Main aims

- **Try to find the most suitable and simple tool to develop a programming language transpiler.** In the field of compilation there is a wide variety of tools allowing the development of complex tasks. That is why we must choose one of them according to its use parameters and utilities.
- **Pick one of the tools and develop a programming language transpiler.** To this end, a programming language to work on the transformation must be chosen.
- **Explore the tool and add functionalities to its core.** That must make the aforementioned task easier, quicker and more efficient. Different functionalities to make developers work simpler must be studied and reached within a reasonable time set for the current final project.

### 1.3.- Rest of the document

After presenting the objectives on the basis of the present final project, its structure must be presented, together with a brief introduction of the different sections included in every chapter.

- **Chapter 1: Introduction.** It presents the project motivation, the targets set and the document's structure.
- **Chapter 2: Current Status.** It presents the different available tools in the market related to the objectives of this final project and a study of their links with the present day.
- **Chapter 3: Analysis, design, and implementation.** It presents the regulatory framework that affects the project, the process that must be followed and the different proposals that have been offered and the final election. On the other hand, the more complex aspects found during the development are remarked. It also explains the necessary aspects for the solution in order to be executed.
- **Chapter 4: Assessment.** It includes an assessment test of the tool, taking into account that it has met the requirements for development and the impact it could have in a real situation example.
- **Chapter 5. Planning and budget.** It contains a proposal of scheduling for this final project and the necessary budget that would be needed if it had to be executed by a developers' team
- **Chapter 6. Dissertation and future works.** Finally, this chapter contains a dissertation for the whole process and for any other future work related to this project that could be interesting.

In turn, the content of the annexes that have been included is discussed:

- **Annex I: Development Environment Installation Manual.** In this part, the necessary steps to configure the project are discussed.
- **Annex II: Tables for Printing.** It includes tables that by size have been necessary their rotation for reading in a PDF.
- **Annex III: Repository.** It presents the repository where all the developed code is found.

## Chapter 2: State of the art.

In this chapter, a market analysis is carried out to know the different proposals that can solve the problem posed. Different available languages for the use with the tool and other possible areas for which rewriting code could be interesting are also studied.

Once the study area has been chosen, an example to develop will be selected to show the viability of the solution.

First of all, it is necessary to say that after an active search on the internet, it has not been possible to find any tool able to solve the problem posed.

**This fact means a growth in the motivation to create the proposed tool**, due to the fact that it means a very important lack in the market that needs to be covered.

Consequently, an analysis of the different existing languages is carried out in order to reach one of the objectives: using the tool as a basis for the subsequent use of developers that may use it for their own benefit.

With the objective of being able of taking the most proper decision, the necessity of studying which are the most used languages needs to be covered in two ways:

**Firstly, it is necessary to find which languages are the most common** in one of the most popular developers repository as GitHub, in function of the active repository of each of them [2].

Secondly, some data collected through Tiobec [3], a webpage, were used. It analyses the number of engineers, official courses and third-party suppliers of each of the languages existing in the world.

With both databases and their results, it was concluded that Java was the most appropriate language to develop the tool, due to the fact that it was in the highest places in both the repository and the data collected.

After deciding the selected language, the selection of the field of application depending of **it interest to apply the rewriting of the code was the next step**.

To that end, a study about which were the technological devices leading the market and its expectations in the future was carried out. It was found out that smartphones were leading the field without any doubt.

Once the field was selected, the most important features for users needed to be analysed. **Battery was the most important aspect.**

With all this information, a study about optimization techniques was carried out and memorization was selected as a very interesting application.

**Memorization [4]** consists on the storage of results of a pure mathematical function [4] and its associations with its arguments. If function is called and the result is stored, the result is given without performing calculations.

This technique would improve the device autonomy, making RAM memory work harder and stay in a second stage due to the fact that devices have a greater capacity nowadays.

## Chapter 3: Analysis, design and implementation

After a description of the different tools found in the market that can be used to rewrite code, the most important aspects of analysis, design and implementation of the tool need to be explained.

The aspects of the aforementioned process necessary to be taken into account are, as follows:

- **Antir, the chosen tool to develop the programming language transpiler.** This tool has been chosen due to the wide range of languages it supports and the facilities for development it contains thanks to its EDI integration and its error control.
- **The difficulty found at the time of understanding Java grammar.** This is due to the huge quantity of syntactic rules of this language, which make difficult to distinguish the different ones used in interesting design cases.
- **The need to know the previous executed rules.** This means a great effort for the designer and the construction of a methodology through booleans parameters enabling him or her to know every previous executed rule to add more information to the grammar.

- **The fact of associating result with arguments.** As mentioned before, a structure is used to storage the data. HashMap was the chosen structure, due to its access speed. This structure uses a key/value structure, which presents the problem of functions with more than one argument. To solve this problem an automatic generated class depending on the function of a specific arguments was used. It also contained the arguments of the function, which allows the key use on HashMap.

## Chapter 4: Assessment.

After presenting the most remarkable aspects in the previous chapter, the assessment of the developed tool took place, for which several tests have been designed on the basis of the requirements.

These requirements will be checked in each test with the aim of validating the whole process.

**After testing the tool, correct results were obtained.** It met all the design requirements mentioned in this chapter.

## Chapter 5. Budget and planning

In the current chapter, everything real and estimated planning will be explained. The first one will be used to create an estimated budget for the execution of the project.

Firstly, it is necessary to talk about budgetary overspending. The sum of real planning was higher than the estimated one, so a study must be carried out to find out the mistakes that have led to underestimate the time needed.

This fact has happened...

- **Because of the huge amount of knowledge necessary to cover every aspect of Language Theory,** used to describe some parts of the process carried out in this final project.

- 
- Because the documentation about the different proposed tools had a wide range of functionalities that have been studied to get to know which one would be the best to solve the problem.

Once the aspect relating to planning have been commented, it is necessary to talk about the budget. There is a comment on the fact that due to the delay we have to apply a percentage depending on the time overspent which has been necessary to carry out the project.

## Chapter 6. Conclusions and Future Works

After a brief explanation of the content in the current chapter, we proceed to comment on the conclusions extracted from the process carried out to develop the tool.

In order to present orderly, explanatory and clear conclusions, the current section has been divided into two subsections:

- **Product conclusions:** All the aspects related to the tool are discussed.
- **Personal conclusions:** All the aspects related to the personal experience of the student who has carried out this final project are discussed.

As it has been said, in this part, a discussion about the obtained conclusions including all their related aspects will take place.

Firstly, it has been possible to observe the power and the great amount of possibilities that the rewriting of code offers. Having a tool which allows to save time on daily and mechanical tasks makes working very much easier. If this concept of tool becomes popular, we could get a great saving on daily tasks.

Secondly and finally, it must be commented that the use of ANTLR, and, in a linked way, the use of type LL grammars, can greatly facilitate the growth of our tool due to the fact that on the one hand, ANTLR offers great facilities, as for example the capability to show how

a syntactic tree is generated and its integration with Eclipse. These facts were previously commented in the section 3.5.5.- Comparativa con las distintas soluciones y la propuesta, being one of the reasons why it was chosen to develop this tool. And what's more, once it has been used, they are reaffirmed by the tests. On the other hand, the use of type LL grammars allows the use of half-rule actions thanks to the way that syntactic rules apply. These have been very useful in different situations and seemed very interesting, but they were not finally used. In addition, it should be commented that the grammar used, in comparison with type LR grammars, was much more understandable.

After presenting the conclusions related to the aspects of the product, the conclusions drawn by the student are going to be presented

First of all, it is worth mentioning the fact that the present student did not attend any lesson on the specialty of "computer science", but "computer engineering", so this final project can be categorized as a challenge, forcing him to leave the comfort zone of concepts learned in his usual area of work. At the same time, the commented fact, rather than an impediment, was a motivation because it allowed the student to get to know the operation of a compiler and the different phases that make it up by himself.

In addition to the compilers, it has been known the concept of a programming language transpiler and the potentiality it may have in the present situation, where the amount of lines of code of the tools are getting bigger and bigger, and in which making a change to recurrent error to improve security can become very complicated and lead to critical situation due to the fact that any person would get confused when trying to modify the code and decide not to change any part of it.

To sum up, the knowledge acquired on grammars was reinforced, allowing to know more deeply the different features they contain, as well as different cases that can be interesting for a parser. Using a specific grammar and decide which one would be more suitable or interesting.

**Once we have commented on the most remarkable conclusions** found during the completion of this final project, with the aim of motivating any developer to make the project go ahead and support its continuity, as well

as working on the concept of rewriting code, different ideas that considered interesting to apply on this tool are presented:

- **Support any kinds of arguments.** As commented in 3.6.- Implementación, no support is included for non-primitive variables. Now, equals method must be used for the call, so it is important to add this feature to the tool.
- **A neural network which decides which methods should be applied for memorization.** As commented in 2.2.- Estudio de la Actualidad, the developer must select a method for memorization to be applied, so the technique may be applied in an incorrect way. Therefore, it could be very interesting that the tool was able to select by itself which functions need to be applied, using different analysis, such as test performances or others.
- **Rewriting of function calls and variables.** Currently, APIs are continuously evolving, due in part to the security problems that are always there. It is very common that any argument, regardless of its value, implies a security flaw in an application. Taking this into account, it is interesting to enable the value change of those arguments and an indication to show what function they belong to. In turn, changing a code structure (a function with its corresponding arguments that was obsolete) for example, for a new one with its respective arguments.
- **Integration with the most popular IDE's.** For a greater ease of use and due to its popularity, it would be interesting that the tool was presented as a plugin compatible with different IDE systems such as Eclipse. That would make it more accessible to users.

# Capítulo 1

## Introducción

En el presente capítulo, se busca presentar, de una manera amena y explicativa, los motivos por los cuáles se ha llegado a plantear el presente Trabajo de Fin de Carrera, así como los objetivos que se han planteado alcanzar para solucionar el problema propuesto.

A su vez, se comentará, brevemente, la estructura diseñada para este documento, así como el contenido que cada uno de los capítulos que componen el documento.

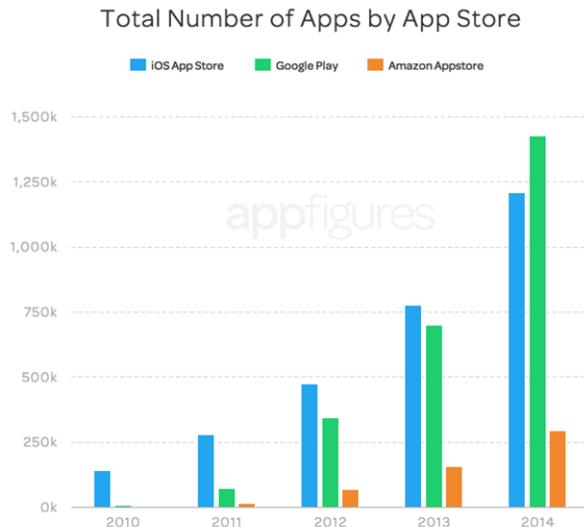
### 1.1.- Motivación

En pleno siglo XXI, se puede afirmar sin temor a equivocarse que vivimos la Era de la Informática, donde la tecnología ha cambiado la percepción que las personas tenemos de la vida y todo lo que nos rodea.

Las empresas de tecnología se han convertido en uno de los pilares de la economía mundial, tal y como se puede observar en la valoración en bolsa que tienen empresas como Apple y Alphabet, entre otras.

Haciendo un inciso en una de ellas, con el objetivo de mostrar el alcance de la tecnología, el lanzamiento del iPhone de Apple, es considerado como un punto de inflexión en la historia, cambiando para siempre el concepto de teléfono móvil que teníamos hasta el momento. [5]

Junto al iPhone, nació la App Store, una tienda con más de 500 aplicaciones nativas para instalar en el dispositivo. [6]. Este hecho provocó una explosión en el desarrollo de las aplicaciones que dura hasta nuestros días.



*Ilustración 1 Crecimiento del N° de Aplicaciones en las Principales Tiendas*

Tal y como podemos ver en el gráfico, [7], el año 2008 dio el pistoletazo de salida al desarrollo de aplicaciones para las plataformas más populares de Sistemas Operativos para móviles, Android e iOS.

Este destacable hecho, tiene intrínsecas una serie de necesidades ligadas al desarrollo de las aplicaciones para las diferentes plataformas, que resumen en los siguientes hechos:

- **Las constantes mejoras de los Sistemas Operativos**, que se realizan para facilitar la vida a los usuarios y atraerlos a su plataforma, provocan que las aplicaciones deban adaptar dichas mejoras mostrando una imagen sólida del servicio.
- **Los problemas de seguridad de los SO y API**, ya que al mismo tiempo que crece la dependencia respecto a la tecnología, crece la necesidad de poder mantener segura la información que trata.

Estos dos hechos, especialmente el segundo, comprometen a los desarrolladores a mantener una constante actualización de las aplicaciones para poder evitar, en la medida de lo posible, problemas de seguridad que permitan ataques que aprovechen ciertas vulnerabilidades.

El mantenimiento de una aplicación, es un coste que siempre se encuentra implícito cuando se desarrolla una aplicación, que es muy importante pero que no mejora la aplicación, sino que simplemente permite que se mantenga el nivel de la aplicación.

Basándose en este mantenimiento y reescritura del código por parte de los desarrollares se basa la motivación de este trabajo fin de carrera: **La posibilidad de poder realizar la reescritura de estructuras de código** con el fin de mejorar viejas implementaciones, cambiar estructuras de código para la adopción de nuevas API o la aplicación de técnicas de optimización para el cálculo de valores muy costosos a nivel de cálculo, muestra la posibilidad de abaratar los costes en el mantenimiento de una aplicación.

## 1.2.- Objetivos

Un transpilador es una herramienta encargada de convertir un código de entrada, en otro lenguaje de salida escogido. Estas herramientas son conocidas como *source-to-source* por ello, ya que son un tipo de compilador, que en vez de transformar el código a lenguaje máquina, lo traducen a otro lenguaje.

En este Trabajo Fin de Grado, en vez de transformarlo a otro lenguaje, se va a transformar una función en otra, utilizando la reescritura de código. Para ello, se utilizarán 3 fases pertenecientes al proceso de compilación, que son las siguientes:

- **Análisis léxico:** En este análisis se realiza el proceso de transformación de un flujo de datos de entrada, en *Tokens*, que son identificadores que utilizan expresiones regulares para detectar los datos de entrada que se corresponde con esa etiqueta. La salida de este análisis es un flujo de *Tokens*.
- **Análisis sintáctico:** Una vez realizada la identificación de los diferentes *Tokens* que conforman el flujo de entrada, se procede a realizar la creación del árbol de sintaxis concreto. Para ello, se apoya en una gramática libre de contexto, que está formada por reglas de producción, las cuales, establecen que *Tokens* pertenecen a las mismas y el orden en el que deben pertenecer.
- **Análisis semántico [8]:** Una vez se ha terminado el anterior análisis, es necesario realizar el análisis semántico, para realizar ciertas comprobaciones que complicarían en gran medida el análisis sintáctico. Por ello, este análisis consiste en la adición de atributos y comprobaciones, que doten y comprueben el sentido del programa realizado.

Una vez se ha realizado una breve y sencilla explicación de los conceptos que envuelve un transpilador, se procede a realizar la descripción de los objetivos que priman este Trabajo Fin de Carrera.

### 1.2.1.- Objetivos Principales

- **Encontrar la herramientas más adecuada y sencilla para desarrollar un transpilador.** En el mundo de los compiladores de código, existe una gran variedad de herramientas que permiten desarrollar grandes y complejas tareas. Esta complejidad, en algunos casos, puede ahuyentar a un desarrollador medio de su aprendizaje, por lo que, se considera muy importante la elección de una herramienta qué a la vez de potente, sea sencilla en su uso y facilite el desarrollo.
- **Elegir una de las herramientas y desarrollar un transpilador.** Para ello, se deberá a su vez, elegir un lenguaje de programación sobre el que se va a realizar la transformación de código
- **Estudiar e incluir funcionalidades a la herramienta.** Tal y como se ha presentado, la actualización de código y reescritura del mismo se presenta como una ardua tarea para cualquier desarrollador. La herramienta que se presenta, pretende facilitar dicha tarea, para que pueda ser realizada de una manera mucho más rápida y eficiente. Para ello, se deberán de estudiar diferentes funcionalidades que faciliten el trabajo a los desarrolladores y que sean alcanzables dentro de un tiempo razonable para el actual TFG.

### 1.3.- Resto del documento

Tras la presentación de los objetivos sobre los que se cimienta el presente Trabajo Fin de Carrera, se continúa con la presentación de la estructura del presente documento, así como una breve introducción de los diferentes apartados que conforman estos capítulos.

- **Capítulo 1** **Introducción.** Presenta la motivación del proyecto, los objetivos planteados y la estructura del documento.
- **Capítulo 2** **Estado de la Cuestión.** En este capítulo se presentan las diferentes herramientas que se encuentran disponibles en el mercado relacionadas con los objetivos del actual TFG y un estudio de la actualidad, relacionada con los objetivos planteados.
- **Capítulo 3** **Análisis, diseño, implementación e implantación.** Se presenta el marco regulador que afecta al proyecto, el proceso que compone del mismo, así como las diferentes propuestas que han sido planteadas para aplicarse y la elección que se ha realizado. A su vez, se destacan los aspectos más complicados que han sido encontrados en el desarrollo de la solución y, se indican los aspectos necesarios para poder ser ejecutada.
- **Capítulo 4** **Evaluación.** Se realiza una evaluación de la herramienta, tanto que los requisitos hayan sido cumplidos en la herramienta desarrollada, cómo el impacto de la solución en un ejemplo real.
- **Capítulo 5** **Planificación y Presupuesto.** En este apartado se presenta la planificación propuesta para el cumplimiento en plazos del TFG, así como el presupuesto necesario para que fuese realizado por un equipo de desarrollo.
- **Capítulo 6** **Conclusiones y Trabajos Futuros.** Finalmente, en este capítulo, se presentan las conclusiones extraídas en todo el proceso llevado a cabo, así como los trabajos futuros que se consideran interesantes relacionados con este proyecto.

A su vez, se comenta el contenido de los Anexos que han sido incluidos:

- **Anexo I: Manual de Instalación del Entorno de Desarrollo.** En esta parte, se comentan los pasos necesarios para configurar el proyecto.

- **Anexo II: Tablas para Impresión.** Se incluyen las tablas que por tamaño ha sido necesaria su rotación para su lectura en un PDF.
- **Anexo III: Repositorio.** Se presenta el repositorio donde se encuentra todo el código desarrollado.

# Capítulo 2

## Estado de la Cuestión

Tras realizar una breve, pero ilustrativa descripción de los pilares sobre los que se cimentar este Trabajo Fin de Carrera y los diferentes propósitos que se buscan conseguir con el mismo, se continúa con un desglose de las diferentes propuestas que se pueden encontrar en el mercado referentes a los objetivos mencionados

Cabe también comentar, que se realiza también un análisis de la actualidad, en cuanto a el posible lenguaje a utilizar para desarrollar la herramienta, como el ejemplo que se va a desarrollar para la misma.

### 2.1.- Comparativa de las diferentes herramientas

Tras una intensa búsqueda por internet, **no ha sido posible encontrar ninguna herramienta que presente una solución para el problema que ha sido planteado.**

Este hecho refuerza aún más la motivación para llevar a cabo la herramienta planteada, ya que se considera una carencia de gran importancia que debería ser cubierta.

## 2.2.- Estudio de la Actualidad

Una vez realizado el análisis de las diferentes herramientas que se pueden encontrar en el mercado, se continúa este capítulo realizando **una investigación en profundidad de las diferentes propuestas que se han encontrado en cuanto a los lenguajes de programación más usados** para valorar el lenguaje de programación a desarrollar y sobre el que plantear la herramienta de interpolación. A su vez, se va a presentar un estudio de los diferentes ejemplos que se pueden desarrollar en el presente trabajo fin de carrera, con el objetivo de demostrar la utilidad de la reescritura de código.

El hecho de analizar los diferentes lenguajes sobre los que desarrollar la herramienta erradica en la premisa de este TFG, el cual, no es otro que facilitar el mantenimiento de código por parte de los desarrolladores aplicando reescritura de código, por lo que, **el lenguaje que tenga el mayor número de líneas, será el más propicio aplicar en una primera instancia**, ya que se aumentará el porcentaje de código sobre el cual, se podrá aplicar los casos implementados.

A su vez, la decisión del lenguaje sobre el que desarrollar la propia herramienta, se considera crucial a la hora de llamar la atención del resto de desarrolladores, debido a que, **a mayor número de desarrolladores, mayor es la probabilidad de encontrar personas interesadas en implementar mejoras sobre la presente herramienta**.

Una vez presentados los dos pilares sobre los que se cimienta este análisis, se continúa presentando los informes encontrados sobre los lenguajes de programación más usados.

Para este análisis se han consultado diversas fuentes para poder tener una base lo más fiable y cercana a la realidad posible.

Para ello, en primer lugar, **se ha consultado uno de los repositorios más conocidos y usados por la comunidad, GitHub**, el cual, ofrece un ranking con los lenguajes más usados en su plataforma, por repositorio. Esto permite poder ver, de una manera general, la importancia que le es otorgado a cada uno de ellos, pudiendo observar el número total de repositorios de cada lenguaje.

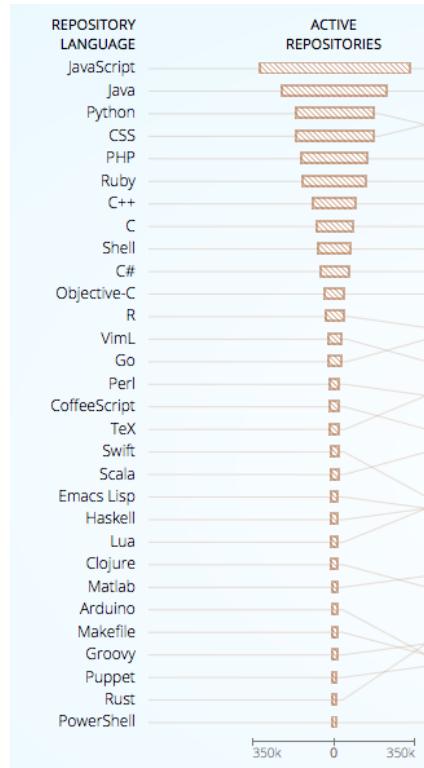


Ilustración 2 Ranking de los Lenguajes de Programación más usados según GitHub [2]

Tal y como se puede observar en la Ilustración 2 Ranking de los Lenguajes de Programación más usados según GitHub , los tres lenguajes más usados son:

- JavaScript
- Java
- Python

Estos 3 serían los más interesantes de analizar, pero, debido a que estos datos solo corresponden a GitHub, **no se considera oportuno decantar la balanza por uno de ellos utilizando una única fuente**, ya que es uno de los más conocidos, pero existen más repositorios.

Debido al hecho relatado anteriormente, y con el objetivo de buscar un enfoque lo más general posible, se ha decidido apoyarse en otra estadística que muestre los lenguajes de programación más usados, pero esta vez, en lugar de utilizar como referencia los repositorios utilizados por cada uno de los lenguajes, **se utiliza el número ingenieros cualificados en todo el mundo, cursos oficiales y proveedores de terceros.** [3]

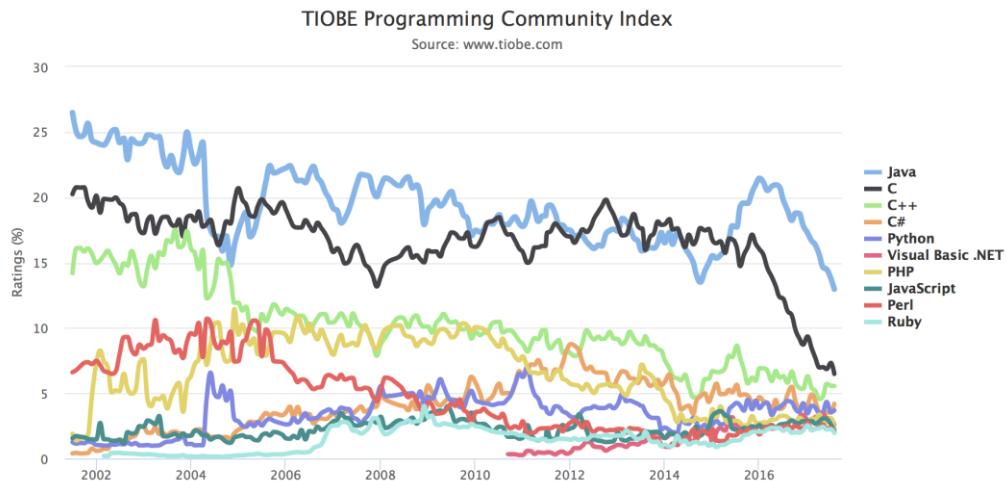


Ilustración 3 Estadísticas de la Comunidad que envuelve cada lenguaje [3]

Tal y como se puede observar en la Ilustración 3 Estadísticas de la Comunidad que envuelve cada lenguaje , se producen una serie de cambios en las posiciones más altas de la comparativa, donde se puede ver como, según estas estadísticas, los lenguajes más utilizados son:

- Java
- C
- C++

Este cambio respecto a las anteriores estadísticas, muestra que es complicado encontrar el lenguaje más usado de manera total y absoluta, ya que según en qué datos se apoye, los resultados pueden diferir y mostrar conclusiones diferentes.

A pesar de la conclusión anterior, si se puede observar como Java, se encuentra entre los lenguajes más utilizados en las dos estadísticas, usando dos puntos de enfoque totalmente diferentes. Este hecho, teniendo en mente en todo momento el objetivo de abarcar el mayor número de desarrolladores posibles, se puede concluir que Java es el lenguaje más atractivo a la hora de cumplir, de la manera más óptima, el objetivo citado.

Antes de finalizar el análisis actual, cabe comentar que no se ha realizado un análisis de las búsquedas de cada uno de los lenguajes en los buscadores, ya que la palabra Java es utilizado tanto para el lenguaje de programación como para su JRE, entre otras herramientas que incluyen estas palabras, por lo que los resultados se ha concluido que no serían acertados.

Una vez se han analizado las diferentes posibilidades de lenguajes de programación, se continúa con el mencionado estudio de los posibles ejemplos de aplicación de la reescritura de código.

A pesar del deseo de presentar una herramienta lo más completa posible, es necesario acotar el problema presentado para encontrar una solución abarcable para el Trabajo Fin de Grado.

Debido al hecho anteriormente expuesto, y con el objetivo de optimizar el esfuerzo del desarrollo y mostrar un ejemplo lo más aplicable posible a programas actuales, se ha decidido enfocare el esfuerzo en las necesidades actuales de los consumidores de tecnología.

Esta decisión se basa en el hecho de que las necesidades que demandan los usuarios, son, a su vez, exigencias a los desarrolladores a la hora de ofertar sus diferentes aplicaciones, por lo que se convierten en consideraciones prioritarias del desarrollo y mantenimiento de código.

Para poder tomar una determinación en dicho ejemplo, y debido al gran número de datos que se pueden encontrar sobre tecnología, se ha acotado la búsqueda a las preferencias de los usuarios de telefonía, ya que, se encuentra como una de las tecnologías inteligentes más vendidas, y sigue con un crecimiento ascendente, tal y como se puede ver en los datos ofrecidos por la empresa Gartnet, especializada ofrecer datos sobre diferentes aspectos del mercado. [9]

Device Type	2016	2017	2018	2019
Traditional PCs (Desk-Based and Notebook)	219	205	198	193
Ultramobiles (Premium)	49	61	74	85
<b>PC Market</b>	<b>268</b>	<b>266</b>	<b>272</b>	<b>278</b>
Ultramobiles (Basic and Utility)	168	165	166	166
<b>Computing Devices Market</b>	<b>436</b>	<b>432</b>	<b>438</b>	<b>444</b>
Mobile Phones	1,888	1,893	1,920	1,937
<b>Total Devices Market</b>	<b>2,324</b>	<b>2,324</b>	<b>2,357</b>	<b>2,380</b>

Ilustración 4 Ventas Actuales y Previsión de Dispositivos Tecnológicos

Basándose en estos datos, Ilustración 4 Ventas Actuales y Previsión de Dispositivos Tecnológicos, se considera lógico profundizar dentro de este mercado sobre las diferentes necesidades que muestran sus usuarios.

Una vez mostrado el campo de la tecnología elegido, se realiza una búsqueda sobre las diferentes preferencias mostradas actualmente por parte de los usuarios de *Smartphone*. Al igual que en casos anteriores, se ha encontrado cierta dificultad a la hora de poder decidirse por una o varias encuestas, debido al gran número de las mismas que se pueden encontrar en diferentes medios.

En consecuencia, se ha decidido elegir la encuesta realizada por el foro de internet XDA, en el cual, se congregan un gran número de desarrolladores y apasionados de la telefonía, ya que los usuarios con un mayor conocimiento en la materia, fruto de su afición a la misma, son los más exigentes, y, por lo tanto, los que más información pueden arrojar sobre el tema. Si se consigue contentar a los usuarios más exigentes, es más probable que también se haga en aquellos que tienen menores pretensiones. [10]

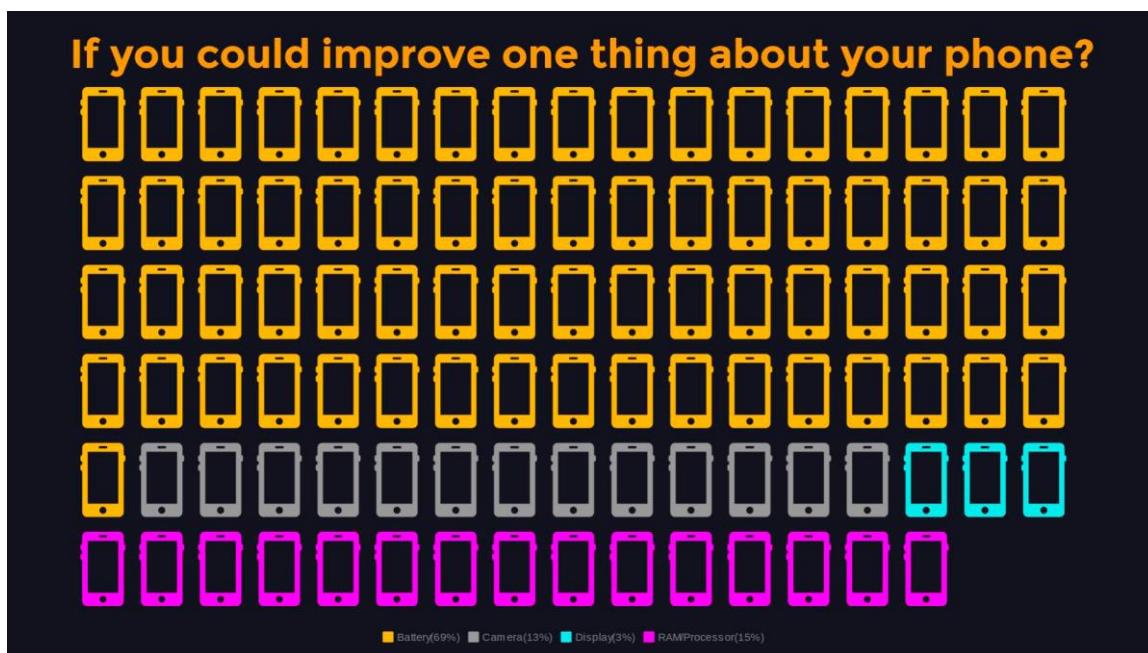


Ilustración 5 Preferencias de los usuarios de Smartphone

Tal y como se puede observar en el gráfico, Ilustración 5 Preferencias de los usuarios de Smartphone, la mayoría de los usuarios encuestados, consideran la batería de un Smartphone como un aspecto primordial.

Esta característica, se muestra en contraposición con el aspecto del dispositivo, ya que esta necesita un espacio que entra en conflicto con el grosor que ocupa el mismo. Este grosor se encuentra como objetivo primordial por parte de los diseñadores, que buscan en todo momento, que sea el menor posible e intentar que sea más atractivo de manera visual por parte del usuario.

Ante este conflicto, aparece otro actor en escena. Las aplicaciones y el uso de los recursos. Un teléfono móvil puede tener una buena batería, pero si utiliza aplicaciones que requieren muchos recursos, se acabará lastrando su autonomía, resultando inútil el esfuerzo dedicado a su diseño.

Estas aplicaciones pueden realizar este uso de los recursos debido a diferentes aspectos, entre ellos, se destaca:

- Un diseño poco eficiente
- Aplicaciones exigentes, como, por ejemplo, Juegos

Entre estos dos aspectos, **se observa más interesante el hecho del uso de recursos por parte de aplicaciones exigente** como, por ejemplo: Asphalt 8. Este popular juego de carreras es común en los test de baterías de los teléfonos debido a sus gráficos avanzados. En el siguiente gráfico, Ilustración 6 Resultados de un test de batería ejecutando Asphalt 8, se puede mostrar cómo la batería de un Smartphone es

acabada en tal sólo 4:37 horas, siendo este el resultado más abultado.  
[11]

Device running Asphalt 8	Median frame rate (fps)	Frame rate stability (%)
Galaxy Note 4 (Exynos 5433, SM-N910C)	51	74
Galaxy Note 4 (Exynos 5433, SM-N910U)	55	78
Galaxy Note 3 (Exynos 5420, SM-N900)	25	61
Galaxy Note 3 (Snapdragon 800, SM-N9005)	29	85
LG G3 (Snapdragon 801, D855)*	27	78

*Ilustración 6 Resultados de un test de batería ejecutando Asphalt 8*

**Estos gráficos, requieren complejas funciones matemáticas** que se utilizan para mostrar las diferentes texturas y recreaciones que se utilizan en el juego.

Este es sólo un ejemplo de lo que pueden lastrar las baterías diferentes aplicaciones y la necesidad que existe de utilizar todas las herramientas posibles con el fin de poder aumentar la autonomía de los mismos.

Dentro de las funciones matemáticas que utilizan estas aplicaciones, **se ha puesto un interés especial en las funciones matemáticas puras**, la

cual, es una función que, para una misma entrada, ofrece el mismo resultado todas las veces que se realiza. [4]

**Debido a la facilidad que ofrece el hecho de que siempre sea el mismo resultado,** se pueden aplicar diferentes técnicas que aprovechan este aspecto.

Entre todas ellas, **se ha elegido la utilización de la técnica llamada Memorización** [4].

**Esta técnica consiste en el almacenamiento de los diferentes resultados que devuelve una función pura,** siendo asociados dichos argumentos a su resultado correspondiente. Si alguna de las llamadas que se realizan a dicha función, ha sido realizada con anterioridad con los mismos argumentos, se devuelve directamente el resultado almacenado, ahorrando así el coste que implicarían realizar los cálculos.

**Esta técnica permite, ahorrar cierto coste en el cálculo de funciones, permitiendo, a su vez, un ahorro de la batería del dispositivo móvil.** Si bien su aplicación conlleva un aumento en el consumo de la memoria RAM, esta, cada vez se encuentra en mayores cantidades en los dispositivos móviles, como por ejemplo el OnePlus 5, con 8GB de memoria RAM [12], siendo asumible el aumento de su consumo, para permitir un aumento de la autonomía de la batería, siempre dentro de unos parámetros.

Como se comenta en el párrafo anterior, **esta técnica conlleva aumentar el uso de la RAM, por lo que debe ser juzgada por el desarrollador,** ya que, si bien plantea un ahorro de cálculos, este puede ser contraproducente en el caso de que el rango de la función sea muy grande, debido a que, como se almacena la estructura que contiene los datos en la memoria RAM del dispositivo, se podría producir un consumo desmedido de la misma, poniendo en peligro la estabilidad del sistema.

Por lo tanto, **siempre el desarrollador debería sopesar sobre qué funciones es interesante aplicar dicha técnica,** para encontrar el equilibrio en dicha aplicación.

Para poder mostrar el funcionamiento de esta técnica de una manera visual y clara, se muestra el siguiente autómata, Ilustración 7 Proceso que realiza la memorización, para una mejor compresión:

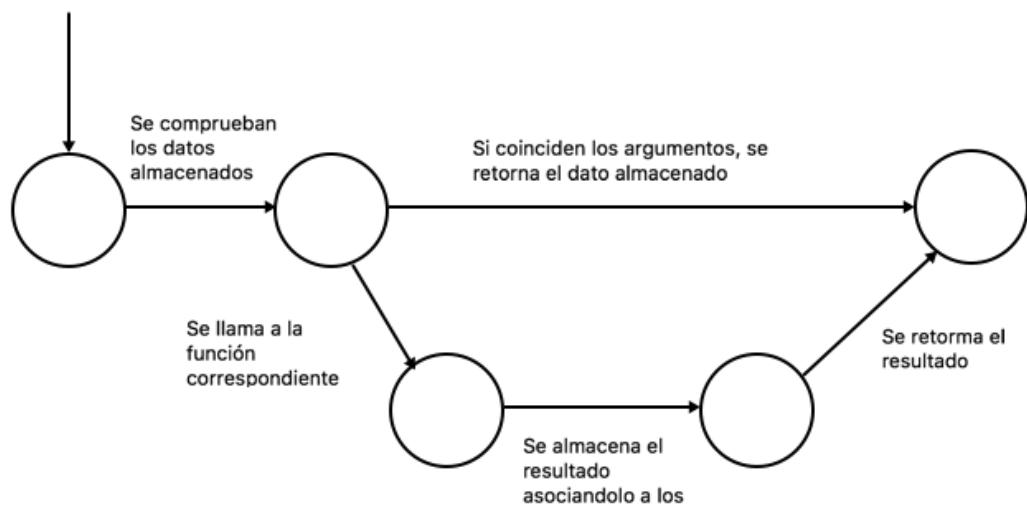


Ilustración 7 Proceso que realiza la memorización

Una vez realizado el estudio deseado, mostrando las diferentes conclusiones que han sido expuestas, se da por finalizado el actual apartado, permitiendo así continuar con el proceso de análisis de requisitos.



# Capítulo 3

## Análisis, diseño, implementación e implantación

Tras una descripción de las diferentes herramientas encontradas en el mercado que realizan reescritura de código, se procede a continuar con lo referente al análisis, diseño, implementación e implantación de la herramienta deseada.

Siguiendo la premisa presentada en el apartado anterior, se ha realizado un desglose del actual apartado en diferentes subapartados, explicando en cada uno de ellos un aspecto diferente del proceso de desarrollo necesario para dicha herramienta.

### 3.1.- Ciclo de Vida

**En lo que respecta al ciclo de vida, se ha decidido que sea prototipado [13].** La idea principal sobre la que se cimienta este modelo, es el hecho de no cerrar los requisitos de la herramienta, sino que van siendo añadidos y modificados en función de los diferentes prototipos que se van generando en cada iteración.

Se ha considerado el ideal, debido a la naturaleza de la herramienta y a los objetivos planteados por la misma, donde, la herramienta será rediseñada por el resto de desarrolladores de la comunidad, a partir de la presentada en el actual TFG.

Este hecho, hace que la herramienta vaya sufriendo cambios desde su planteamiento inicial hasta su versión definitiva, que van a ser más

fácilmente vistos, si se implementa un prototipo y se observa su funcionalidad.

Las ventajas que se consideran determinantes que han decantado la balanza a favor del prototipado son las siguientes:

- **Participación por parte de los usuarios finales en el diseño de la aplicación.** Tal y como se ha presentado esta herramienta, tiene como objetivo facilitar el desarrollo y el mantenimiento del código que han escrito con anterioridad, para reducir el tiempo necesario y el coste que implica. Esto, facilita en gran medida el diseño, ya que los propios desarrolladores son los usuarios finales, y no hay nadie mejor que ellos mismos para conocer las necesidades de la misma.
- **Retroalimentación por parte de los usuarios.** Gracias al anterior argumento, se cimienta el actual, ya que, al ser los usuarios finales, los propios desarrolladores de la herramienta, existe una retroalimentación por parte de los mismos, para ver si los requisitos diseñados, son útiles aplicados en la práctica y los posibles errores que se hayan podido realizar.
- **Menor probabilidad de rechazo.** Basándose en el hecho de que esta herramienta será abierta a que otros desarrolladores implementen y mejoren lo ofrecido hasta el momento, se reduce en gran medida el rechazo de la misma, debido a que, si uno de ellos dedica parte de su tiempo a esta, es por su interés en utilizarla cuando haya sido terminada.

Una vez han sido presentados las ventajas que trae, se considera interesante plantear, a su vez, las desventajas que presentan y el por qué se consideran menos determinantes que las ventajas.

- **Es necesario mucho tiempo.** Este tipo de ciclo de vida, hace que la herramienta final necesite mucho tiempo para poder ser alcanzada, debido al tiempo que es empleado en el desarrollo de los prototipos y en el rediseño de los requisitos.  
Esto no es importante, debido a que no se tiene indicado ningún plazo de tiempo.
- **El esfuerzo invertido en el desarrollo de prototipos puede ser desmedido.** Este hecho se debe al carácter iterativo de los requisitos del prototipado, y se deja a criterio del desarrollador, ya que, al ser para su propio beneficio, este será el principal conocedor del esfuerzo que requerirá este desarrollo y los beneficios de alcanzar el objetivo que se plantee.

### 3.2.- Marco Regulador

Una vez finalizada la presentación del ciclo de vida elegido para la herramienta, se continúa con el marco regulador que concierne al desarrollo actual.

Una vez realizado un estudio de las leyes y normas que pueden afectar a la herramienta presente, se han considerado que las dos siguientes tienen una relación directa con el presente desarrollo.

#### Ley de Protección de Datos (LOPD)

Objeto de la ley:

*“La presente Ley Orgánica tiene por objeto garantizar y proteger, en lo que concierne al tratamiento de los datos personales, las libertades públicas y los derechos fundamentales de las personas físicas, y especialmente de su honor e intimidad personal y familiar” [14]*

Aplicación a la presente herramienta:

Una vez presentado el objeto de la ley, y aplicada a los desarrollos software, se considera los siguientes puntos como los esenciales para cumplir la presente ley:

- Facilitar el acceso a las políticas de privacidad
- Informar y solicitar permiso para el tratamiento de los datos, de carácter previo a la instalación de la herramienta.
- Permitir la supresión de los datos recogidos.
- Establecer un periodo de conservación de los datos.
- Ceñirse a lo estipulado en las políticas privacidad y avisar a los usuarios de los cambios en la misma, permitiendo a su vez, la aceptación o rechazo de estos cambios, con sus respectivas consecuencias.

Una vez han presentados los anteriores puntos, se pasa a comentar los aspectos que afectan a la herramienta actual presentada. Cada vez que se desee realizar un rediseño de la herramienta, con motivo de su mejora, deberá de ser contemplado esta ley, para seguir su cumplimiento.

Todos los aspectos comentados con anterioridad, son cumplidos por la herramienta, debido a los elementos que sean reescritos por parte del programa, son presentados en la propia herramienta, en la ayuda de cada una las acciones, advirtiendo al usuario de las acciones del mismo, quedando en la mano del desarrollador. A su vez, en el caso de que se

ejecute la herramienta con su correspondiente acción, se considera que se ha leído las acciones que implican.

A su vez, es necesario comentar que la acción incorporada en el actual TFG, realiza el almacenamiento en memoria RAM de los datos, siendo estos solo accesibles por la funcionalidad implementada por el desarrollador que la aplique, por lo que se cumple con dicha ley.

### Ley de Protección Intelectual

*“La propiedad intelectual está integrada por una serie de derechos de carácter personal y/o patrimonial que atribuyen al autor y a otros titulares la disposición y explotación de sus obras y prestaciones.” [15]*

En cuanto al hecho de que la actual herramienta haya utilizado software con derecho de autor, se puede afirmar que todo lo desarrollado cumple con la actual ley de propiedad intelectual, ya que se han utilizado herramientas que permite su utilización.

En el caso del uso de ANTLR, este tiene Licencia BSD [16], por lo tanto, se permite la utilización del mismo en software libre, como es este el caso.

### 3.3.- Metodología

Tras realizar comentar los aspectos que afectan al presente Trabajo Fin de Grado, se continúa con la descripción de la metodología, para asegurar la calidad del desarrollo y su entrega en los tiempos estipulados.

La metodología de la que se ha partido es Métrica v3 [17], concebida como una metodología de planificación, desarrollo y mantenimiento de sistemas software.

Esta metodología está compuesta por los siguientes procesos:

- Planificación de Sistemas de Información (PSI)
- Estudio de Viabilidad del Sistema (EVS)
- Análisis del Sistema de Información (ASI)
- Diseño del Sistema de Información (DSI)
- Construcción del Sistema de Información (CSI)
- Implementación y Aceptación del Sistema (IAS)
- Mantenimiento del Sistema de Información (MSI)

Una presentados los diferentes procesos que se realizan dentro de dicha metodología, se procede a mencionar el hecho de que ha sido aplicada de una manera simplificada, debido a que no todos los apartados de esta aplican ni son necesarios, ya el elemento desarrollado tiene una corta vida, ya que, como se ha mencionado anteriormente, el ciclo de vida es el prototipado, por lo que no tiene sentido realizar un gran esfuerzo en este sentido, si, en un corto periodo de tiempo, se va a desarrollar otra herramienta que extienda la presente.

En relación con el hecho que se acaba de comentar, se muestra a continuación la equivalencia de los apartados del presente documento con las tareas realizadas que componen los procesos de la metodología seleccionada.

TFG	Métrica V3 [18]
Capítulo 1 Introducción	EVS: Establecimiento del Alcance del sistema
Capítulo 2 Estado de la cuestión	EVS: Valoración del estudio de la situación actual
3.2.- Análisis	EVS: Identificación de Requisitos
3.3.- Diseño	EVS: <ul style="list-style-type: none"> <li>- Estudio de Alternativa de Solución</li> <li>- Valoración de las alternativas</li> <li>- Selección de Solución</li> </ul> DSI: <ul style="list-style-type: none"> <li>- Diseño de Casos de Uso Reales</li> </ul>
Capítulo 5 Planificación y Presupuesto	PSI: Definición del Plan de Acción

Tabla 1 Equivalencia TFG - Métrica V3

### 3.4.- Análisis

Una vez finalizada la introducción, se pasa a realizar una detalla explicación en lo que se refiere al análisis de requisitos que conforma la herramienta a desarrollar del TFG.

Cabe mencionar que cuando se refiere a acciones en los requisitos, se refiere a acciones que aplique reescritura de código.

Cada uno de los requisitos de software será redactado mediante una tabla que se detalla a continuación:

RX-XX								
Necesidad	Alta	Media	Baja	Claridad	Alta	Media	Baja	
Estabilidad	Esencial	Deseable	Opcional	Verificabilidad	Alta	Media	Baja	
Descripción								

Tabla 2 Ejemplo de especificación de requisito

La etiqueta que identificará cada uno de los requisitos funcionales y no funcionales, de manera inequívoca será la siguiente:

RX-YY

Donde:

- RX: Expresa el tipo de requisito al que se refiere la tabla
  - RF: Requisito funcional
  - RNF: Requisito no funcional
- YY: Expresa el número del requisito. Este valor se establece dentro de cada uno de los dos tipos de requisitos que se han definido:

Los atributos que definen un requisito son los siguientes:

- **Identificador:** código que identifica de forma única a cada requisito.
- **Necesidad:** medida del interés de los usuarios/clientes en que la aplicación lo realice. Puede ser Esencial en su mayor grado, deseable en término medio u opcional en un grado bajo.

- **Claridad:** medida de la ambigüedad del requisito y su correcta formulación lingüística. Puede ser alta, media o baja.
- **Estabilidad:** medida sobre el valor de permanencia del requisito a lo largo de la vida del software. Puede ser alta, media o baja.
- **Verificabilidad:** medida del nivel de comprobación de un requisito. Puede ser alta, media o baja.
- **Descripción:** Explicación y redacción del requisito.

### 3.4.1.- Requisitos Funcionales

Una vez se ha realizado la descripción de cada uno de los campos que conforman la tabla que se va a utilizar para describir cada uno de los requisitos de la herramienta, se procede a mencionar cada uno de los **requisitos software** que conforman esta herramienta:

RF 1								
Necesidad	Alta	Media	Baja	Claridad	Alta	Media	Baja	
Estabilidad	Esencial	Deseable	Opcional	Verificabilidad	Alta	Media	Baja	
Descripción	Se ofrecerá un listado de las diferentes opciones que ofrece la herramienta							

Tabla 3 RF 1

RF 2								
Necesidad	Alta	Media	Baja	Claridad	Alta	Media	Baja	
Estabilidad	Esencial	Deseable	Opcional	Verificabilidad	Alta	Media	Baja	
Descripción	Todos los comandos deberán ser únicos, y hacer referencia a una única acción							

Tabla 4 RF 2

RF 3								
Necesidad	Alta	Media	Baja	Claridad	Alta	Media	Baja	
Estabilidad	Esencial	Deseable	Opcional	Verificabilidad	Alta	Media	Baja	
Descripción	Se permitirá elegir solo una de las opciones que ofrece la herramienta							

Tabla 5 RF 3

RF 4								
Necesidad	Alta	Media	Baja	Claridad	Alta	Media	Baja	
Estabilidad	Esencial	Deseable	Opcional	Verificabilidad	Alta	Media	Baja	
Descripción	La herramienta deberá mostrar información adicional, al menos, referente a los desarrolladores de la misma y la licencia							

Tabla 6 RF 4

RF 5							
Necesidad	Alta	Media	Baja	Claridad	Alta	Media	Baja
Estabilidad	Esencial	Deseable	Opcional	Verificabilidad	Alta	Media	Baja
Descripción	La herramienta deberá mostrar todas las acciones que realiza la herramienta						

Tabla 7 RF 5

RF 6							
Necesidad	Alta	Media	Baja	Claridad	Alta	Media	Baja
Estabilidad	Esencial	Deseable	Opcional	Verificabilidad	Alta	Media	Baja
Descripción	Cada una de las acciones deberá tener asociado un comando						

Tabla 8 RF 6

RF 7							
Necesidad	Alta	Media	Baja	Claridad	Alta	Media	Baja
Estabilidad	Esencial	Deseable	Opcional	Verificabilidad	Alta	Media	Baja
Descripción	Cada una de las acciones deberá mostrar información respecto a sus argumentos y funcionalidad						

Tabla 9 RF 7

RF 8							
Necesidad	Alta	Media	Baja	Claridad	Alta	Media	Baja
Estabilidad	Esencial	Deseable	Opcional	Verificabilidad	Alta	Media	Baja
Descripción	La herramienta permite seleccionar qué funciones se desea interceptar para aplicar reescritura de código						

Tabla 10 RF 8

RF 9								
Necesidad	Alta	Media	Baja	Claridad	Alta	Media	Baja	
Estabilidad	Esencial	Deseable	Opcional	Verificabilidad	Alta	Media	Baja	
Descripción	Una acción disponible en la herramienta será la técnica de memorización. Esta utilizará reescritura de código para aplicarla. Se podrá elegir los métodos sobre los que se desea aplicar memorización, introduciendo, encima de la cabecera del método, la etiqueta "@memorization"							

Tabla 11 RF 9

RF 10								
Necesidad	Alta	Media	Baja	Claridad	Alta	Media	Baja	
Estabilidad	Esencial	Deseable	Opcional	Verificabilidad	Alta	Media	Baja	
Descripción	La estructura que almacene los argumentos y el resultado correspondiente debe de poder almacenar estos de manera independiente al número de los mismos.							

Tabla 12 RF 10

RF 11								
Necesidad	Alta	Media	Baja	Claridad	Alta	Media	Baja	
Estabilidad	Esencial	Deseable	Opcional	Verificabilidad	Alta	Media	Baja	
Descripción	El programa funciona con etiquetas añadidas al propio nombre del programa, las cuales, tendrán la siguiente funcionalidad y estructura: <ul style="list-style-type: none"> <li>• <b>La herramienta deberá mostrar las opciones</b>, con sus comandos correspondientes, mediante el uso del nombre de la herramienta, sin ninguna etiqueta, siguiendo el siguiente patrón:  [Nombre de la herramienta]</li> <li>• <b>La herramienta deberá mostrar las todas las acciones disponibles</b> por la herramienta con el comando -actions, con el comando que corresponda a cada una de las acciones.  [Nombre de la herramienta] -actions</li> <li>• <b>Todas las acciones deberán mostrarán su descripción y los argumentos necesarios</b> siguiendo el siguiente patrón.  [Nombre de la herramienta] -action - inf  siendo -inf la acción del comando para mostrar la información de la acción.</li> <li>• <b>La etiqueta -aboutus, mostrará, por lo menos, los desarrolladores de la herramienta.</b>  [Nombre de la herramienta] -aboutus</li> <li>• En caso de que se produzca algún error en los argumentos de entrada, se retornará “Por favor introduzca una sentencia válida”</li> <li>• Si se produce un error de ruta, se retorna “Error de Ruta”</li> </ul>							

Tabla 13 RF 11

RF 12							
Necesidad	Alta	Media	Baja	Claridad	Alta	Media	Baja
Estabilidad	Esencial	Deseable	Opcional	Verificabilidad	Alta	Media	Baja
<b>Descripción</b>							Los tipos soportados por dicha estructura serán:
							<ul style="list-style-type: none"> <li>• 'boolean'</li> <li>• 'char'</li> <li>• 'byte'</li> <li>• 'short'</li> <li>• 'int'</li> <li>• 'long'</li> <li>• 'float'</li> <li>• 'double'</li> </ul>

Tabla 14 RF 12

### 3.4.2.- Requisitos No Funcionales

Una vez presentados los requisitos funcionales, se continúa con el resultado del análisis, en cuanto a los requisitos no funcionales se refiere.

RNF 1							
Necesidad	Alta	Media	Baja	Claridad	Alta	Media	Baja
Estabilidad	Esencial	Deseable	Opcional	Verificabilidad	Alta	Media	Baja
<b>Descripción</b>							La herramienta deberá ser ejecutada desde consola de comandos

Tabla 15 RNF 1

RNF 2							
Necesidad	Alta	Media	Baja	Claridad	Alta	Media	Baja
Estabilidad	Esencial	Deseable	Opcional	Verificabilidad	Alta	Media	Baja
<b>Descripción</b>							La herramienta deberá estar desarrollada en Java

Tabla 16 RNF 2

RNF 3							
Necesidad	Alta	Media	Baja	Claridad	Alta	Media	Baja
Estabilidad	Esencial	Deseable	Opcional	Verificabilidad	Alta	Media	Baja
Descripción	Se podrá ejecutar en cualquier SO, que sea compatible con JRE 1.8.0_141-b15 o versiones compatibles con la misma						

Tabla 17 RNF 3

RNF 4							
Necesidad	Alta	Media	Baja	Claridad	Alta	Media	Baja
Estabilidad	Esencial	Deseable	Opcional	Verificabilidad	Alta	Media	Baja
Descripción	El análisis de los datos de entrada y la reescritura se realizará mediante el uso de un transpilador.						

Tabla 18 RNF 4

RNF 5							
Necesidad	Alta	Media	Baja	Claridad	Alta	Media	Baja
Estabilidad	Esencial	Deseable	Opcional	Verificabilidad	Alta	Media	Baja
Descripción	La gramática que se utilice para el análisis léxico y semántico, debe ser del tipo LL.						

Tabla 19 RNF 5

### 3.4.3.- Matriz de Trazabilidad

Una vez se han mostrado todos los requisitos que conforman el diseño de la herramienta, se muestra la relación que tienen entre ellos, mediante la siguiente Matriz.

Cada uno de los cuadros ha sido referenciado, mediante una referencia cruzada para facilitar la lectura de la misma.



RNF 5								x	x									x	
-------	--	--	--	--	--	--	--	---	---	--	--	--	--	--	--	--	--	---	--

Tabla 20 Matriz de Trazabilidad

### 3.5.- Diseño

Tras realizar una detalla descripción de los diferentes requisitos que conforman la herramienta a desarrollar, se continúa el capítulo con las decisiones que conforman el diseño de la herramienta, en cuanto a la herramienta en la que apoyarse para realizar el análisis de los datos de entrada y, los casos de usos que se van a contemplar.

Siguiendo el orden establecido anteriormente, se comienza con el análisis de las diferentes herramientas que se han decidido analizar.

Para una mejor compresión de las diferentes propuestas, se ha decidido dividir esta sección en apartados, los cuáles, estarán dedicados cada uno de ellos a describir la propuesta que se ha valorado utilizar. A su vez, finalizando la sección, se incluirá un apartado en el que se valorará de forma conjunta todas las bondades y limitaciones de las herramientas para alcanzar una conclusión respecto a la herramienta a utilizar.

Cabe mencionar que, a pesar de ser un análisis para tomar una decisión, no se ha realizado una comparativa comparando los aspectos que abarca una herramienta frente a la otra, si no las bondades que ofrece una herramienta en sí, y que beneficios conllevaría su utilización respecto al problema planteado. Por ello, no se centra en que sean compatible con Java o no. En el caso de que una herramienta fuese lo sumamente beneficiosa para el proyecto y no fuera compatible con Java, se buscarías las soluciones oportunas para integrar la solución a Java.

#### 3.5.1.- JavaCC [19]

En primer lugar y tras una breve introducción del contenido que se va a tratar en el actual capítulo, se procede a comentar la primera de las herramientas que han sido propuestas para su utilización en el desarrollo de la herramienta propuesta.

Se trata de *JavaCC*, generador de analizadores léxico – sintáctico de código abierto, propietaria de Sun Microsystem. Permite el reconocimiento de secuencias y elementos basándose en una gramática libre de contexto y sus correspondientes reglas de producción.

**JavaCC integra las características tanto de analizador léxico como sintácticas en una sola herramienta**, debido a esto, solo es necesario un único fichero para las especificaciones léxicas y semánticas. Este hecho,

facilita en gran medida la lectura y mantenimiento del código, debido a que, de un solo vistazo, se pueden ver tanto las reglas de producción como la gramática libre de contexto.

A su vez, tanto la gramática como las reglas de producción, **utilizan especificaciones BNF extendida**, facilitando así la compatibilidad entre ficheros que hayan seguido la norma y su reutilización por parte de otros desarrolladores.

La parte del análisis sintáctico **utiliza un análisis sintáctico descendente de tipo LL (k)**. Además, se permite generar autómatas deterministas como no-deterministas.

A su vez, ofrece la flexibilidad de generar porciones de la gramática libre de contexto que sean tipo LL(k), con el fin de eliminar ambigüedades.

Entre otra de las características a destacar, **se permite la inclusión de bloques de código Java tras la identificación de un token**, permitiendo una gran cantidad de posibilidades a la hora del desarrollo. Cabe mencionar, que, si bien se permite esta inclusión, se ha de tener en cuenta que **el análisis del bloque de código no es exhaustivo**, pudiendo permitir la inclusión de errores sintácticos en el código Java, que una vez generados los archivos, lance algún tipo de error el código.

A parte de los bloques de código, se pueden añadir palabras clave como, por ejemplo, **skip**, entre otras, que añaden funcionalidades extra a las reglas. En el caso de **skip**, ignora el *token* que ha sido identificado.

JavaCC incluye **JJTree**, una herramienta de preprocesado que realiza el desarrollo de árboles. Esta herramienta permite mostrar la creación del árbol de análisis sintáctico, mediante la creación de un fichero aparte del utilizado para la gramática.

### 3.5.2.- Yacc [20] y Lex [21]

Tras realizar la descripción de las características más importantes que se han encontrado sobre JavaCC, se pasa a realizar la descripción sobre otras de las posibles soluciones que han sido planteadas para la resolución de los objetivos planteados.

En este caso, se trata de la **utilización de dos herramientas** [22], **Yacc y Lex**, las cuales, son un analizador sintáctico y léxico, correspondientemente. Estas dos herramientas están escritas en C, y a su vez, sólo son compatibles con C y C++ para poder ser utilizadas.

Con el fin de realizar un análisis y explicación lo más clara posible, se van a comentar, en primer lugar, las características de Lex, ya que es el analizador léxico y, en segundo lugar, las que corresponden con Yacc, respetando las principales etapas que sigue un proceso de interpretación de un lenguaje.

Con el objetivo de la escritura de la gramática y su reconocimiento por parte de la herramienta, se ha de generar un fichero “.l”, donde especificar la gramática libre de contexto.

Para el reconocimiento de *token*, se utiliza un autómata de estados finitos, el cual, sigue la siguiente estructura mostrada en la Ilustración 8 Autómata Finito de estados Lex:

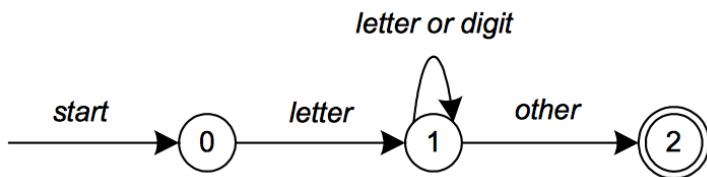


Ilustración 8 Autómata Finito de estados Lex

Debido a esta estructura, no es adecuado para manejar estructuras anidadas, como se pueden dar en un método, ya que, para este tipo de estructuras, se necesita una pila para manejar estas situaciones. Por esta situación, es necesario el apoyo en Yacc.

En esta integración, Yacc, recoge la salida de Lex y realiza el análisis sintáctico de los *token* correspondientes, utilizando para ello, las reglas de producción. Esta relación se ha de realizar utilizando ambas herramientas con el fin de enlazar la salida de Lex, con sus correspondientes toles, con la entrada de Yacc.

Yacc, con la misma metodología que Lex, requiere un fichero “.y”, en el cual, se declaran las reglas de producción, entre otras funciones que permite.

El analizador sintáctico realiza un análisis bottom-up [23], en el cual, parte de una expresión de la gramática completa, y la va reduciendo en función de los *token* que va recibiendo por la entrada, hasta terminar en un único no-terminal.

Para facilitar la compatibilidad entre gramáticas, Yacc utiliza la especificación BNF y, su vez, facilitando su uso, ya que esta se encuentra muy extendida.

Para la especificación de reglas, Yacc permite modificar las relaciones de asociatividad, permitiendo que sean tanto por la derecha como por la izquierda, aportando mayor flexibilidad a las reglas.

En la producción de reglas de producción, Yacc permite asociar acciones en código C o C++, cuando se realiza la asociación con los *token* de entrada. Estos *token* pueden ser referenciados desde la misma acción, siendo \$\$ el *token* que se desea detectar y \$n siendo n = {1...n} donde n es el número de *token* que se encuentran formando parte de la regla.

También se considera necesario comentar que se permite añadir funciones que sean llamadas desde la acción de la regla de producción, permitiendo una mayor limpieza y orden.

Se permite el control de errores que se puedan producir, como caracteres no permitidos que sean detectados por Lex, pero estos deben de ser implementados por el desarrollador, en cualquier caso, a través de la función yyerror.

Tanto Lex como Yacc son herramientas que solo pueden ser utilizadas desde la terminal. Todos los resultados son mostrados a partir de esta y el enlazado de los ficheros es necesario que sea realizado desde la misma, utilizando una serie de comandos específicos.

### 3.5.3.- Bison [24] y Flex [25]

Una vez se han comentado todas las características Lex y Yacc, se continúa con la comparativa, mostrando otras dos herramientas que siguen la misma organización que las mostradas, ya que, Flex, es el generador de analizadores léxicos y Bison, el generador de analizadores sintácticos. [26]

Tal y como se realizó en el anterior apartado, se describirán en primer lugar, las características más destacables de Flex y, posteriormente, se realizará lo propio con Bison.

En lo que se refiere a Flex, partiendo de un fichero en extensión “.l” realiza la generación de un fichero con la extensión “.c”, en el cual, se encuentra la función yylex (), encargada de realizar la relación de los elementos de entrada con los *token* que posee. Este fichero se encuentra escrito en código C.

A pesar de que anteriormente no se hayan descrito la estructura de los ficheros, en este caso, si se va a realizar con el objetivo de poder ver con

mayor claridad cómo funciona Flex, ya que una estructura un poco compleja.

Este fichero se divide 3 zonas:

- **Definiciones:** En esta sección se especifican *token* con su definición, es decir, la expresión regular que sigue. Estos *token* deben de encontrarse definidos en el fichero de Bison correspondiente.
- **Reglas:** El actual apartado es dedicado para establecer las expresiones regulares con sus correspondientes acciones. A su vez, se pueden referenciar *token* que hayan sido declarados en la sección de definiciones, con el fin de crear expresiones regulares más complejas.
- **Código de Usuario:** En esta última sección se permiten escribir código C, que será escrito tal cual en el archivo resultado tras ejecutar Flex.

Flex, permite modificar tanto la salida como la entrada del programa, permitiendo así realizar las redirecciones en función de las necesidades del desarrollador.

En cuanto a su funcionamiento, se utiliza la función `yylex()`, que debe de ser llamada para realizar el reconocimiento de los elementos. Esta función, finaliza o bien cuando se ha alcanzado el final del fichero o bien cuando alguna de las acciones realiza un `return`, acción que se realiza de forma cuando se ha detectado una expresión regular o *Token*, siendo devuelto un *Token* declarado en Bison.

Para poder realizar pruebas sobre la correcta implementación de los elementos comentados con anterioridad, Flex permite la posibilidad de activar un *flag* a la hora de generar su fichero, que realiza la impresión de los elementos detectados de la siguiente forma mostrada en la Ilustración 9 Resultado Modo Debug Flex:

```
accepting rule at line 53 ("the matched text")
```

Ilustración 9 Resultado Modo Debug Flex

Esto facilita en gran medida el desarrollo de herramientas ya que se puede de ver de forma rápida donde se pueden encontrar errores.

Finalmente, se considera necesario comentar la **incompatibilidad que existe entre ciertos elementos de Flex con Lex**, debido a que Flex, es

una reescritura de este, siendo añadidas diferentes mejoras que no se encontraban anteriormente.

Una vez comentados todos los aspectos destacables en lo que se refiere a Flex, se pasa a realizar lo propio con Bison.

**Bison**, tal y como se ha comentado anteriormente, es un generador de analizadores sintácticos, el cual, partiendo de un fichero en extensión “.y”, es capaz de generar un fichero en extensión .c, que realice el análisis sintáctico. Este fichero “.c”, para que realice la función completa, debe de ser compilado junto con “.c” correspondiente generado por la herramienta Flex, para tener el análisis completo que se busca.

Bison, permite realizar la generación de sus ficheros tanto en código C como C++, así como es compatible totalmente con los ficheros generados para Yacc, lo cual, facilita en gran medida el mantenimiento de herramientas creadas para Yacc, que se deseen portar a esta herramienta.

En cuanto a las reglas de producción, igual que todas las anteriores, utiliza la notación BNF, y este analizador, utiliza las gramáticas LALR, para crear el analizador sintáctico.

Una característica interesante de Bison, es el hecho de que permite especificar el tipo de variable que se espera que devuelva un Token. De esta forma, se puede obtener el dato de una manera correcta sin necesidad de realizar transformaciones, si se devolviese el tipo de dato en formato String, o perdida de información, en el caso de que por defecto fuese int. Por contra, se debe de mencionar que, al especificar la existencia de más de un tipo de dato, se debe de especificar en cada token el tipo de dato que se trata.

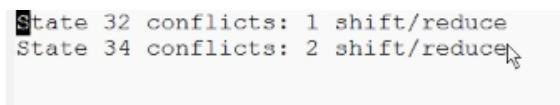
Se permite, además, asociar a cada producción, una acción, siendo esto código en C y C++. Dentro de cada acción, se puede obtener el contenido de los token que componen las reglas de producción. Para ello, se sigue la misma metodología seguida en Yacc.

Finalmente, se considera necesario comentar el funcionamiento de la detección de errores en Bison.

Existen dos tipos de errores posibles que pueden ocurrir:

- **Error de diseño:** La gramática que ha sido implementada no ha sido de la manera correcta y no puede ser compilado el fichero.
- **Error funcional:** la gramática es compilada de forma correcta pero no realiza el análisis sintáctico tal y como queremos.

**En cuanto a los errores de diseño,** Bison permite activar un *flag*, “-v”, el cual, muestra si existe algún tipo de problema en el diseño de la gramática. Para mostrarlo, genera un fichero “.output”, donde realiza una descripción de las acciones que va realizando en cada estado, Ilustración 10 Detección de Errores de Diseño.



```
State 32 conflicts: 1 shift/reduce
State 34 conflicts: 2 shift/reduce
```

Ilustración 10 Detección de Errores de Diseño

Se considera necesario aclarar que **cuando se habla de estado, se refiere al estado del autómata de estados finitos** que se utiliza para realizar el análisis sintáctico.

En lo que respecta a los errores funcionales, se puede activar una serie de *flags*, dentro del fichero de la gramática, que permiten mostrar todos los pasos que se realizan a la hora de analizar la gramática. De esta forma, se puede observar en que paso se produce el funcionamiento no deseado, Ilustración 11 Resultado Autómata Bison.



```
Stack now @ 1
Entering state 7
Next token is token end_T 
Shifting token end_T <>
Entering state 20
Reducing stack by rule 1 (line 6):
    $1 = token begin_T <>
    $2 = nterm statements <>
    $3 = token end_T <>
-> $$ = nterm program <>
```

Ilustración 11 Resultado Autómata Bison

Tal y como se puede observar, todo lo relacionado con Bison y Flex, se realiza a través de consola, como las herramientas del apartado anterior, sin existir ningún para que realice cierto apoyo al desarrollador.

### 3.5.4.- ANTLR [27] [28]

Tras realizar la descripción de las características de Flex y Bison, finalmente se llega a la última herramienta dentro de la comparativa. **Antlr, Another Tool for Language Recognition**, es una herramienta que englobe tanto el análisis léxico como el sintáctico.

Esta herramienta se caracteriza por encontrarse integrada en algunos de los más usados IDE como Eclipse o NetBeans, teniendo a su vez su propio entorno de desarrollo, ANTLRWorks2, aunque esta se encuentra un poco desactualizada, ya que la salida de su última versión data de 28/8/2013.

Antes de continuar con la descripción de dicha herramienta, se ha de comentar que se ha hecho el análisis utilizando esta herramienta junto con el IDE Eclipse, que facilita en gran medida su uso.

A su vez, existen una gran variedad de lenguajes compatibles con esta herramienta, los cuales, son los siguientes:

- Java
- C#
- Python (2 y 3)
- JavaScript
- Go
- C++
- Swift

Otra característica de esta herramienta, es el hecho de que solo necesita un fichero tanto para el analizador léxico como semántico, en el cual, se encuentran tantas las reglas léxicas como semánticas, facilitando así la lectura del mismo por parte del desarrollador, tal y como se ha podido observar en JavaCC.

A su vez, hay que comentar que las gramáticas que se construyen son del tipo BNF, siendo a su vez, del tipo LL.

Al igual que herramientas anteriores, se permite añadir acciones a tanto a las reglas sintácticas como a las léxicas, teniendo en cuenta que se encuentran en diferentes *scopes*. Para solventar este problema, se puede declarar una variable global, en la clase main, permitiendo así pasar datos entre los dos tipos que podemos encontrar.

Estas acciones, tienen un importante detalle, es el hecho de que no se realiza la detección de errores en las acciones, es decir, en el archivo donde se encuentra la gramática, si no en el archivo que se genera en consecuencia al mismo, el Parser o el Lexer. Estos errores, además, no se ven de forma clara, ya que difiere el código escrito que el que se puede encontrar en la clase. Esto puede resultar muy molesto, y complica la resolución de errores, ya que, además, estos errores no se detectan al instante, si no cada vez que se ejecuta la herramienta para actualizar el fichero, con un espacio de tiempo establecido.

Dentro de las acciones, se permite referenciar los *token* que pertenecen tanto al *parser* como al *lexer*, utilizando \$ y el nombre del *token*, a diferencia de Yacc y Bison, en los cuales, se referencia por posición.

En cuanto a la depuración de errores en cuanto al diseño del analizador léxico y sintáctico, reconoce los errores de sintaxis que se pueden producir en tiempo real, lo cual, es de gran ayuda, ya que no es necesario ejecutar la herramienta por consola, como se ha visto anteriormente, para detectar errores, si no que se muestran directamente en el archivo de la gramática.

A su vez, ANTLR ofrece una herramienta, para el diseño de la sintaxis muy interesante. Es el hecho de poder observar cómo se crea el árbol de sintaxis de la gramática, permitiendo, al desarrollar, ver si el comportamiento es el que desea, introduciendo en el recuadro correspondiente el código que se desee probar, Ilustración 12 Ejemplo Depuración Gramática.



Ilustración 12 Ejemplo Depuración Gramática

A su vez, si el árbol generado es muy grande, se permite seleccionar a partir de que regla se desea partir, introduciendo el código que corresponda para realizar las pruebas pertinentes. En este caso, se ha señalado el *token* expresión, para poder ver cómo se comporta ante la entrada que se muestra en la Ilustración 13 Ejemplo de Selección de Reglas a Analizar.

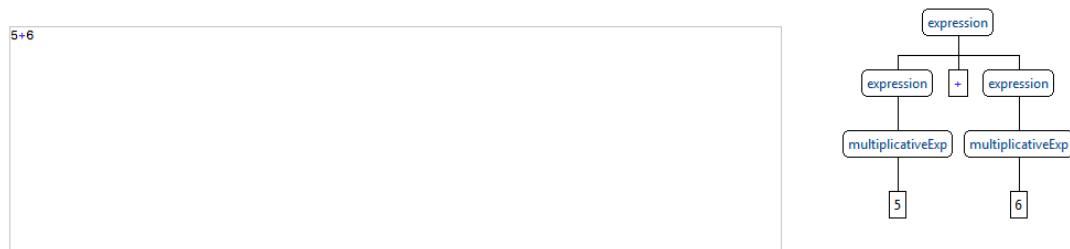


Ilustración 13 Ejemplo de Selección de Reglas a Analizar

### 3.5.5.- Comparativa con las distintas soluciones y la propuesta

Una vez han sido expuestas las características más importantes de las herramientas que han sido seleccionadas, se debe pasar a realizar la explicación de la herramienta escogida para el desarrollo.

Antes de comenzar, se considera necesario comentar que cualquiera de las herramientas que han sido expuestas, tienen la capacidad de poder cumplir con los objetivos expuestos, pero se ha buscado primar la facilidad de uso para que la herramienta sea atractiva para que siga siendo desarrollada con posterioridad.

Por lo tanto, siguiendo la premisa expuesta, se han descartado en primer lugar las herramientas **Lex** y **Yacc**, debido a que Lex ha sido reescrito creando **Flex**, añadiendo nuevas funcionalidades, por lo que no tiene sentido utilizar una herramienta desfasada y sustituida ya por otra.

A su vez, **Yacc** ha sido descartado en favor de **Bison** debido a las siguientes razones:

- **Existen mayores funcionalidades en Bison:** Todas funciones que implementa Yacc son ofrecidas por Bison, y este, a su vez, ofrece un abanico más grande de posibilidades, como por ejemplo, las acciones a media regla, que son acciones que son posibles de ejecutar, antes de que se haya completado la detección de una regla.
- **Mayor soporte de lenguajes:** Además de C, Bison puede generar código en C++, por lo que, en el caso de que la herramienta quiera ser portada a este lenguaje, no sería necesario cambiar la herramienta.

A continuación, para finalizar la comparativa, se mostrará un análisis del resto de herramientas restantes, el cual, se ha basado en las siguientes premisas:

- **Tipo de gramática que acepta**
- **El número de lenguajes sobre los que se permite desarrollar**
- **Control de errores**

En cuanto respecta al tipo de gramática que utilizan [29], se pueden agrupar en LL y LR, englobando en este grupo todas sus variantes, las cuales, tienen diferentes matices que no se consideran necesarios comentar.

Existen grandes diferencias entre estas gramáticas, de las cuales, se ha de comentar el hecho de su forma de recorrer el árbol de análisis semántico, construido para poder realizar el análisis.

En el caso de las gramáticas tipo LL, el árbol se recorre desde el nodo raíz hasta las hojas, donde se encuentran los elementos obtenidos por los *token*.

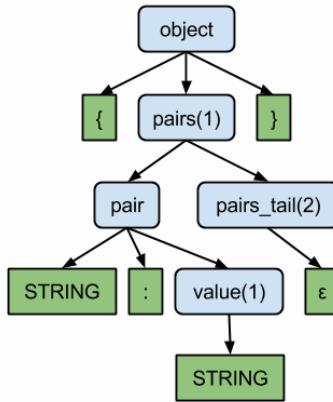


Ilustración 14 Árbol sintáctico generado por una gramática LL [29]

Para poder crear dicho árbol, se ha de apoyar en una acción conocida como *lookahead*.

Esta técnica se utiliza debido al hecho, de que se puede dar el caso en el que sea necesario observar los siguientes *token* para poder tomar la decisión de si aplicar una regla, en el caso de que sea así, cuál. En el caso de este tipo de gramática, debido a que solo tienen una parte de la regla antes de aplicar, debido a la estrategia que sigue a la hora de construir el árbol, como mínimo, se debe de realizar la lectura de 1 *token* para poder aplicar una regla, ya que comienza la lectura al principio de los *token* de la regla. Debido a esto, se permite conocer el contexto de cada instante del análisis.

Por otro lado, las gramáticas del tipo LR, realiza el recorrido del árbol de las hojas al nodo raíz, aplicando las reglas para ir reduciendo los elementos de la entrada, hasta llegar al nodo raíz, comenzando el análisis por las hojas del árbol.

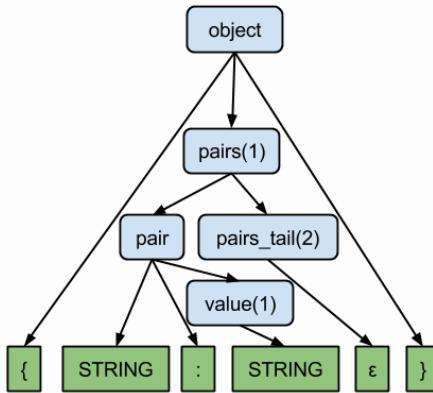


Ilustración 15 Árbol sintáctico generado por una gramática LR [29]

Esta estrategia reduce en gran medida el *lookahead*, debido a que conocen todos los elementos de la regla antes de tener que tomar una decisión, reduciendo así el número de errores que se producen al aplicar las reglas de producción.

Una vez expuesto el concepto en el que se basan los tipos de gramáticas que utilizan los analizadores sintácticos, se consideran preferentes los analizadores que aceptan las gramáticas tipo LL, ya que tal y como se exponen en los objetivos, se busca implementar un transpilador, que analice las expresiones para aplicar diferentes reescrituras en el código.

Para realizar la reescritura en el código pertinente, es necesario un contexto para poder encontrar los casos en los que se deben aplicar o no, por lo que el tipo de gramática comentada facilita en gran medida esta implementación.

En consecuencia, las herramientas de Bison y Flex son descartadas, al utilizar un derivado de la gramática LR.

Finalmente, como herramientas válidas quedan JavaCC y ANTLR. Ante esta disyuntiva, se ha decidido escoger ANTLR como herramienta sobre la que desarrollar los objetivos planteados.

Para realizar dicha decisión se ha realizado la siguiente argumentación:

- **La gran cantidad de lenguajes soportados por ANTLR:** Uno de los deseos a la hora de desarrollar este trabajo fin de carrera es que sirva de base sobre la que desarrollar otros ejemplos y herramientas que permitan facilitar cada vez más el mantenimiento del código por parte de los desarrolladores, el cual, podría utilizar en toda la herramienta en su conjunto, o partes de ella. Por ello, para facilitar la portabilidad de la gramática desarrollada hasta el momento y todas las acciones que hayan sido implementadas, un argumento

válido para que un desarrollador quisiera portar la herramienta que se va a implementar en el actual TFG, es el hecho de la gran compatibilidad de lenguajes de ANTLR, facilitando dicha tarea.

- **La compatibilidad de ANTLR con los IDE:** A pesar del hecho de que se ha podido encontrar compatibilidad de JavaCC con IDE, estos *plugins* que se ofrecen se encuentran totalmente indocumentados, por lo que no se puede realizar una comparativa entre los *plugins*. Sólo se ha podido encontrar información respecto a su uso por terminal.
- **El control de errores y su integración:** Tal y como se ha comentado con anterioridad, ANTLR ofrece facilidades a la hora de implementar la gramática, mostrando el árbol creado durante el análisis.

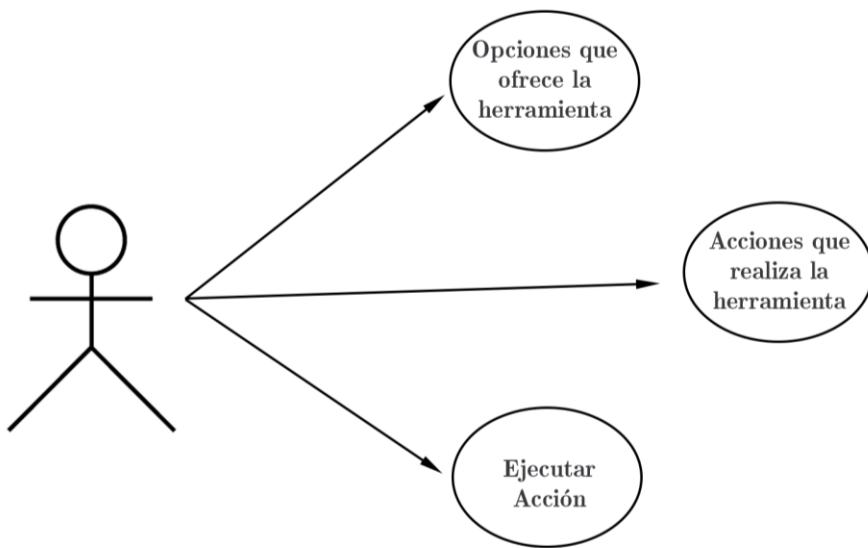
En contraposición, JavaCC sólo se puede usar desde Terminal, y a la hora de realizar el análisis de la gramática, es necesaria la creación de un nuevo fichero, para poder realizar la creación del árbol del análisis sintáctico utilizando JTree, entorpeciendo la agilidad de desarrollo.

### 3.5.6.- Casos de Uso

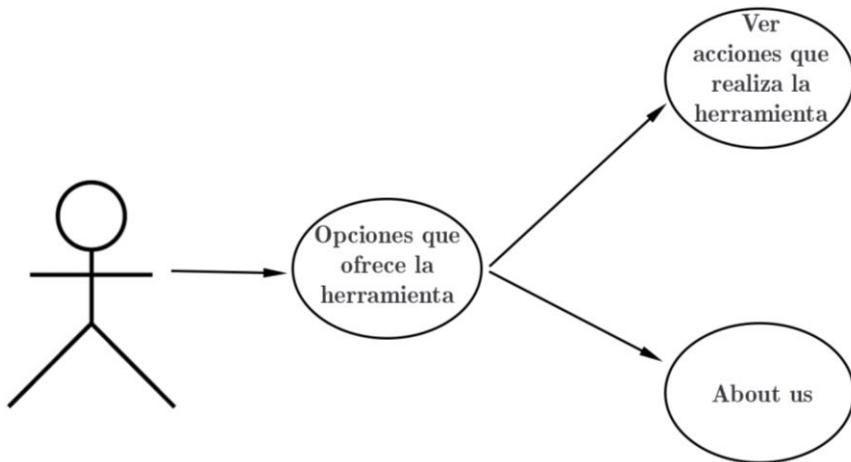
Tras realizar una extensa comparativa sobre las diferentes herramientas que se pueden encontrar para poder implementar el transpilador, y tras elegir de ellas se ha considerado más correcta para este cometido, se continúa presentando los casos de uso de la herramienta.

En este apartado, se describirán los casos de uso, describiendo el proceso que seguiría un usuario al utilizar la herramienta por primera vez.

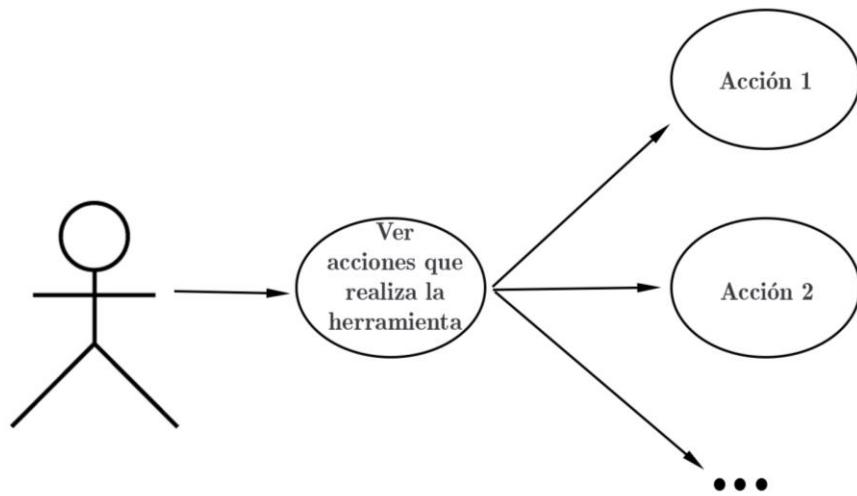
En primer lugar, **las acciones disponibles que se presentan por parte de la herramienta**, en el caso de que se ejecute sin ninguna opción es:

*Caso de Uso 1*

**En el caso que el usuario quiera ver las opciones de la herramienta,** se podrá observar lo siguiente, mostrando a su vez, el comando correspondiente para realizar cada una de las acciones:

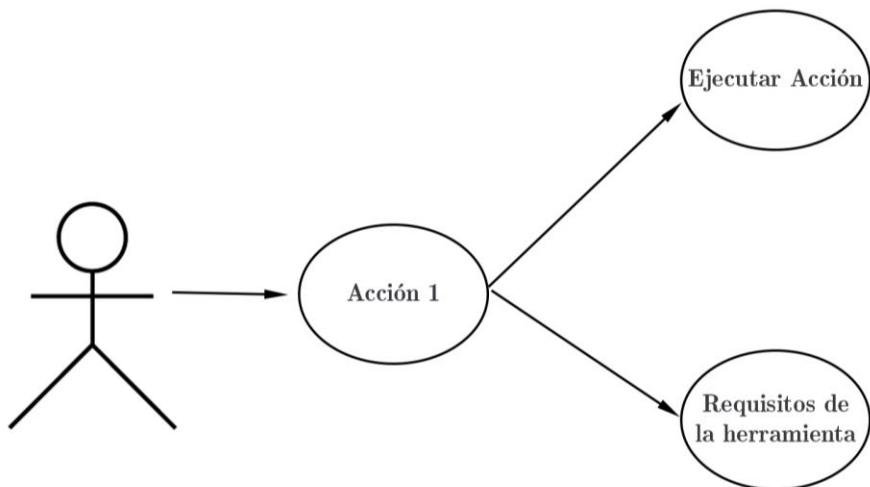
*Caso de Uso 2*

Continuando con el flujo de acciones, se procede a mostrar el caso que se deseen ver las acciones que es posible realizar por parte de la herramienta. Al igual que anteriormente, a cada uno de las acciones estará asociado un comando que permita ejecutar la acción deseada.



*Caso de Uso 3*

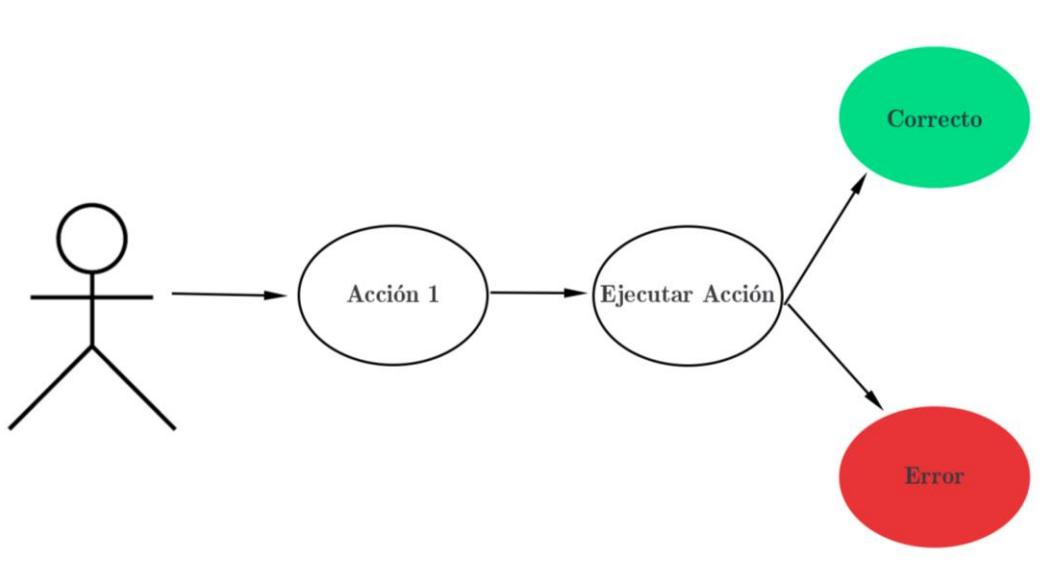
Una vez son conocidas todas las acciones que puede realizar la herramienta, y continuando el flujo de las posibles acciones por parte del usuario, se procede a mostrar las posibilidades que se ofrecen cuando se desee utilizar la acción.



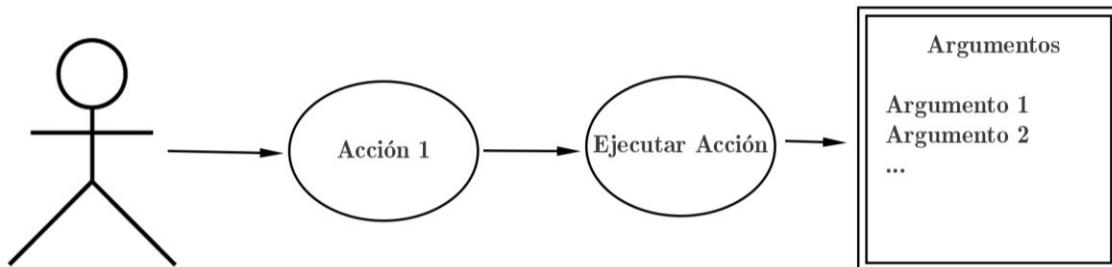
*Caso de Uso 4*

Por un lado, en el caso de que se ejecute una acción, se podrán ofrecer 2 resultados:

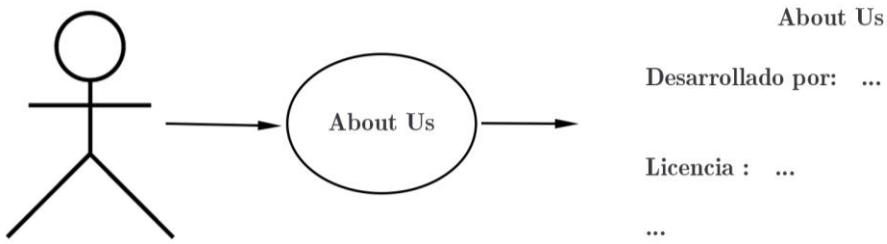
- Correcto
- Error, apareciendo, en este caso, el motivo del error producido.

*Caso de Uso 5*

Por otro lado, si el usuario desea conocer los argumentos necesarios para realizar la acción, se le mostrarán los diferentes argumentos, su orden y las diferentes opciones de cada uno de ellos con su significado.

*Caso de Uso 6*

Finalmente, en el caso de que se desee conocer información adicional de la herramienta, se accederá al apartado *about us*, donde se mostrará dicha información.



En caso de que se produzca algún error, se retornará:

- “Introduzca una sentencia válida” si se introduce una sentencia incorrecta
- “Error de Ruta” si el path de la ruta es incorrecto

### 3.6.- Implementación

Una vez se ha presentado todo el proceso referente a su diseño, herramienta a utilizar, etc. Se procede a realizar una descripción de los instantes más complicados que se han pasado en el proceso de implementación de la herramienta de desarrollo.

En primer lugar, se debe comentar que la gramática utilizada para realizar el análisis ha sido obtenida de un repositorio [30], la cual, ha sido desarrollada por el creador de ANTLR, y, por lo tanto, es compatible con las características de la herramienta.

Esta decisión se basa en el hecho de que crear una propia gramática no es necesaria, ya que no se está inventando un lenguaje, si no se quiere utilizar la sintaxis de uno ya existente y probada por lo que sería un esfuerzo innecesario.

Una vez comentado este hecho, y siguiendo hablando sobre la gramática, se debe mencionar **que se realizó un análisis costoso sobre esta misma para poder conocer que elementos no terminales eran necesarios modificar o añadir** para poder realizar la reescritura del método, aplicando memorización.

Para facilitar esta tarea, se ha utilizado el mencionado recuadro anteriormente en la comparativa de herramientas, el cual, permite ver, utilizando una serie de ejemplos, que flujo sigue el árbol para analizar la entrada de los datos correspondientes, como se puede ver a continuación.

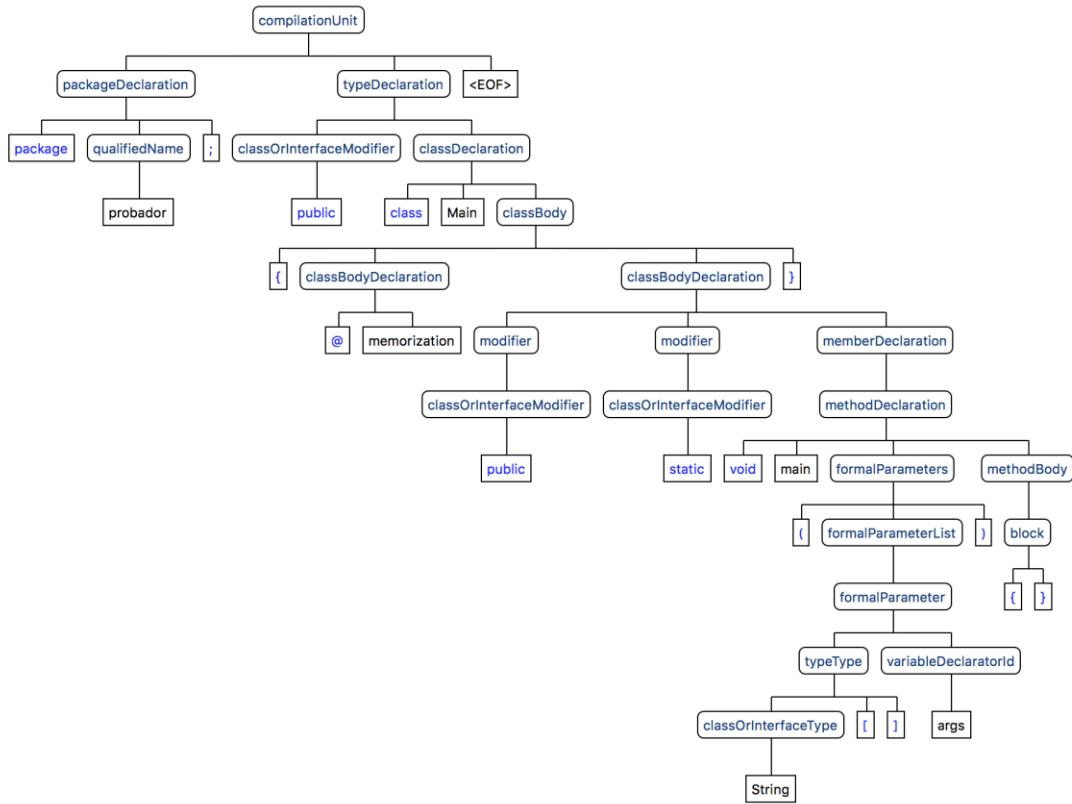


Ilustración 16 Árbol generado por ANTLR

Antes de comenzar a realizar la explicación del funcionamiento generado en torno a la reescritura y con el fin de una mejor compresión, se intuye necesario comentar que las reglas semánticas que han añadido para utilizar código Java, se ejecutan cuando se ha generado toda la parte del árbol sintáctico correspondiente.

A este árbol, Ilustración 16 Árbol generado por ANTLR, han sido necesario añadir una serie de *token* para poder adaptarlo al objetivo mencionado, como, por ejemplo, la detección de la etiqueta “memorization”.

```

classBodyDeclaration
:   ';'
|   'static'? block
|   modifier* memberDeclaration
|   memorizationDeclaration
;
  
```

Ilustración 17 Ejemplo de Modificación de la Gramática

```

memorizationDeclaration
:
startMemorization modifier* memberDeclaration
{
    if (finishDetection == true){
        finishDetection = false;
        System.out.println ("finishDetection = " + finishDetection);
        System.out.println ("Hola");
        reader.modificarMetodoMemorizacion(nombreMetodo, identificadorMetodo, $memberDeclaration.text, tipoDeRetorno, Argumentos);
        nombreMetodo = "";
        identificadorMetodo = "";
        tipoDeRetorno = "";
        Argumentos.clear();
    }
}
;

```

Ilustración 18 Ejemplo de Modificación de la Gramática

```

startMemorization:
AT Identifier
|
AT typeOfLabels
{
    String etiqueta = $typeOfLabels.text;
    if (etiqueta.equals("Memorization") || etiqueta.equals("memorization")) {
        labelMemorizationDetected = true;
        System.out.println ("labelMemorizationDetected = " + labelMemorizationDetected);
    }
}
;

```

Ilustración 19 Ejemplo de Modificación de la Gramática

En este ejemplo, se ha añadido el no terminal “`startMemorization`”, Ilustración 19 Ejemplo de Modificación de la Gramática, y el resto de no terminales que derivan del mismo.

Este conjunto: Ilustración 17 Ejemplo de Modificación de la Gramática , Ilustración 18 Ejemplo de Modificación de la Gramática Ilustración 19 Ejemplo de Modificación de la Gramática; es el encargado de generar la estructura correspondiente al árbol sintáctico que se corresponde con la detección de la estructura que conforma dicha etiqueta, Ilustración 20 Detección de la etiqueta correspondiente a la memorización.

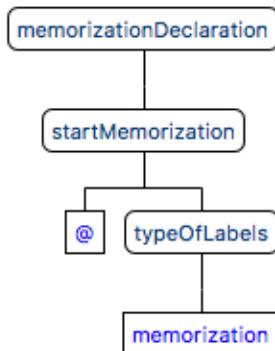


Ilustración 20 Detección de la etiqueta correspondiente a la memorización

A su vez, y añadidas todas las etiquetas necesarias, se muestra el árbol resultado, Ilustración 21 Ejemplo de Cabecera de Método.

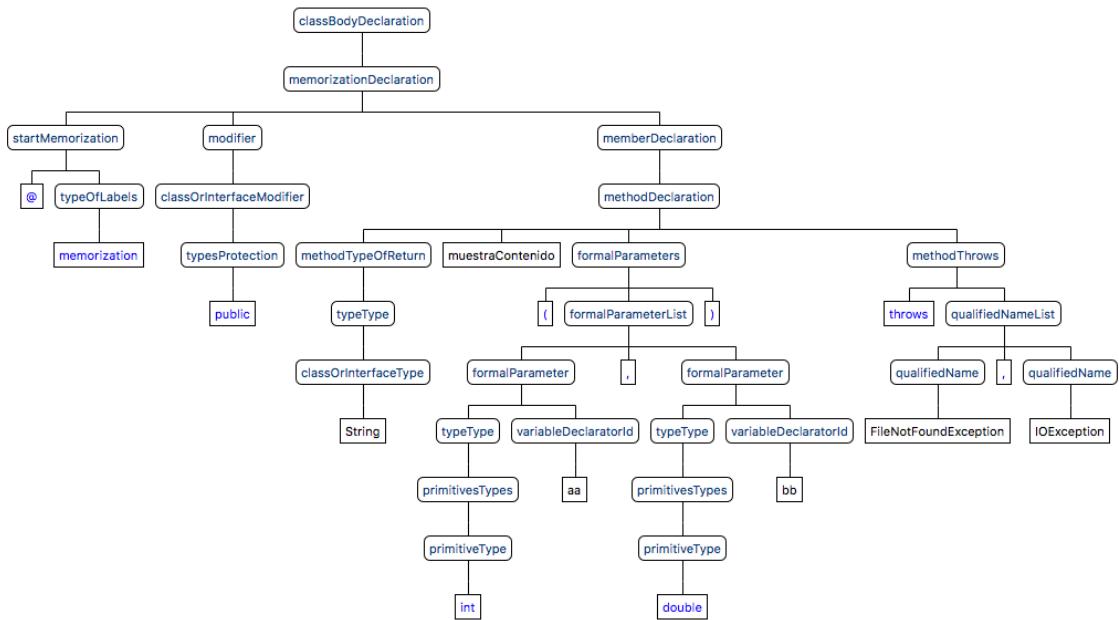
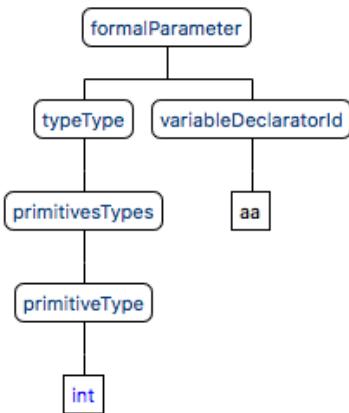


Ilustración 21 Ejemplo de Cabecera de Método

El análisis sintáctico detecta que el contenido es correcto, pero no se puede realizar nada con el mismo, ya que no tiene el contexto para conocer que este resultado es el buscado. Por ello, se añaden las reglas semánticas en código Java, que analizan el contenido de los *Token*.

En relación con las reglas semánticas, otro de los problemas encontrados, es el hecho de que ANTLR no se ha encontrado una interfaz en la documentación (ni en el código), que de la posibilidad de acceder al árbol sintáctico al mismo tiempo que este se está generando. Por ello, se han tenido que ser necesarios el uso de *boolean*, para poder detectar el contexto de las reglas.

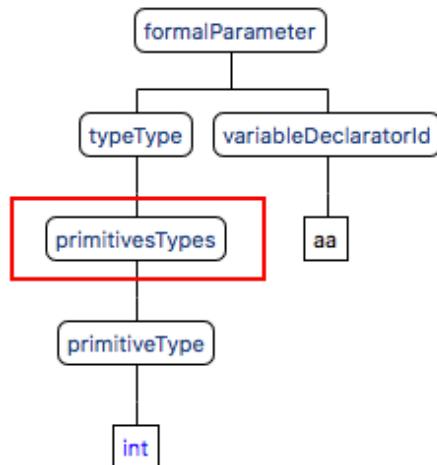
Para conocer de una manera más fiel, la utilización de los *boolean*, se muestra en siguiente ejemplo de árbol semántico, Ilustración 22 Ejemplo de funcionamiento boolean, simplificado, para que sea simple y claro.



*Ilustración 22 Ejemplo de funcionamiento boolean*

Se muestra en primer lugar, un trozo del árbol sintáctico que se genera cuando se detecta la cabecera de un método. Estas etiquetas pertenecen únicamente a la detección de la cabecera de los métodos.

Una vez observado el árbol, se muestra el proceso que corresponde con la detección del argumento que lo compone.



*Ilustración 23 Paso 1 en la detección de los argumentos*

En primer lugar, Ilustración 23 Paso 1 en la detección de los argumentos, se comprueba el tipo del argumento. Si el argumento es de tipo primitivo ('boolean', 'char', 'byte', 'short', 'int', 'long', 'float', 'double'), se sigue realizando comprobaciones. En caso contrario, se finalizan las comprobaciones, no siendo compatible el método con lo soportado.

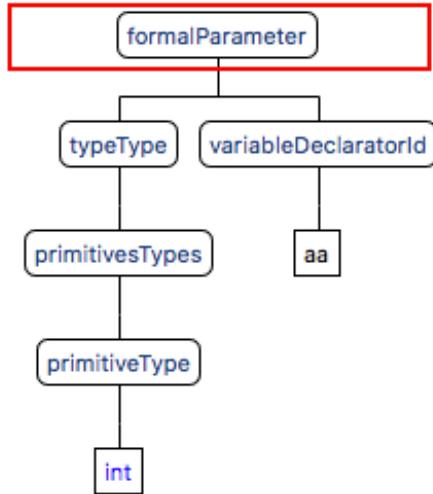


Ilustración 24 Paso 2 en la detección de los argumentos

A continuación, Ilustración 24 Paso 2 en la detección de los argumentos, cuando termina de comprobarse la regla `formalParameter`, se comprueba si se ha detectado un tipo primitivo. Si es así, se devuelve el valor de dicho boolean al valor falso, y se comprueba a su vez, si se ha iniciado la detección de un método compatible con la acción de memorización. Si es así, dicho argumento es almacenado.

Siguiendo este ejemplo, se puede observar como las acciones se ejecutan de abajo a arriba del árbol y el funcionamiento de los boolean en estas acciones asociadas.

Finalmente, cabe comentar un aspecto más que ha resultado complicado a la hora de poder conseguir el objetivo planteado. Es lo referente a los argumentos de la función que se desee reescribir.

El problema reside en el hecho de que una `HashMap` ha sido la estructura elegida para almacenar los resultados de las funciones, debido a su velocidad de acceso. **El problema de esta es que realiza un almacenamiento clave-valor**, por lo que, cuando los argumentos son más de uno, no se podría utilizar dicha estructura, en principio.

Para solucionar este problema, y en relación con lo mostrado anteriormente referente al funcionamiento de los boolean, se ha modificado la gramática, añadiendo elementos no terminales y acciones para detectar los argumentos y su tipo.

Una vez han sido detectado todos los argumentos de dicha función, **se crea una clase con todos estos argumentos, y se asigna como clave dicha clase**. De esta manera, se soluciona dicho problema.

Cabe mencionar que no se pueden utilizar cualquier tipo de argumento, solo son válidos los argumentos de tipo primitivo. Esto es debido a que, los tipos primitivos, pueden realizar una igualación sin necesidad de ser implementada. Por el contrario, las clases por ejemplo tipo String, necesitan una llamada al método `equals`, complicando la implementación y obligando un esfuerzo de tiempo extra que no es posible asumir, dejándose como un objetivo futuro.

### 3.7.- Implantación

Una vez se ha terminado de mencionar los aspectos relacionados con el desarrollo que más han costado al desarrollados, se finaliza el capítulo con los aspectos necesarios para la utilización de la herramienta desarrollada.

**La herramienta ha sido desarrollada con la siguiente versión de Java,**  
Ilustración 25 Versión Java:

```
java version "1.8.0_141"
Java(TM) SE Runtime Environment (build 1.8.0_141-b15)
```

*Ilustración 25 Versión Java*

Por ello, para la utilización de la herramienta será necesario la versión mostrada o compatibles con la misma.

A su vez, y tal y como se muestra en el apartado 3.4.- Análisis, **la herramienta deberá ser ejecutada por consola.**



# Capítulo 4

## Evaluación

Tras presentar todos los aspectos referentes a la aplicación, y de realizar su correspondiente desarrollo, se procede a mostrar las pruebas correspondientes, para poder comprobar si se han cumplido todos los aspectos diseñados para esta.

Para poder realizar dichas pruebas, se utilizarán los requisitos presentados en el apartado 3.4.- Análisis.

Posteriormente, se presentará un análisis de resultados respecto a la memorización, para mostrar los efectos que provoca la inclusión un entorno real, con una función matemática pura.

### 4.1.- Plan de Pruebas

Una vez presentado el capítulo, se continúa con la descripción del plan de pruebas diseñado para comprobar que la implementación cumple con los requisitos descritos, de una manera correcta.

Siguiendo con la descripción del Plan de Pruebas, se continúa especificando el entorno sobre el que se van a llevar a cabo las pruebas.

Entorno de Pruebas	
Dispositivo	Macbook Pro 2017 Touch Bar
Procesador	3,1 GHz Intel Core i5
Memoria RAM	16 GB 2133 MHz LPDDR3
Disco Duro	512 GB SSD
SO	macOS Sierra 10.12.6
Versión de Java	1.8.0_141-b15

Tabla 21 Entorno de Pruebas

Una vez descritas las características del entorno sobre el que se van a desarrollar las pruebas, se procede a describir los apartados de la plantilla que sobre las que se van a definir las pruebas mencionadas.

PXX	
Objetivo	
Precondiciones	
Proceso	
Postcondiciones	
Resultado	
Trazabilidad	

Tabla 22 Tabla de Ejemplo Pruebas

Los atributos que componen la prueba, son descritos a continuación:

- **PXX:** Es el identificador de la prueba, siendo este único en el ámbito de las pruebas de requisitos.
    - XX Indica el número de la prueba que se trata
  - **Objetivo:** Presenta el objetivo establecido de la prueba en cuestión.
  - **Precondiciones:** Establece las precondiciones necesarias para poder ejecutar la prueba
  - **Proceso:** Indica los pasos necesarios para realizar la prueba.
  - **Postcondiciones:** Condiciones que deben cumplirse después de realizarse la prueba.
- 
- **Resultado:** Indica el estado de la prueba. Los estados posibles son:

- Verificado
- No Verificado
- **Trazabilidad:** Establece la relación entre los requisitos y la prueba.

Finalmente, se presentan las pruebas diseñadas para el plan actual.

Prueba 1	
Objetivo	Se comprueba que la herramienta ha sido desarrollada y se ejecuta en Java, mostrando las opciones que ofrece, con el comando especificado.
Precondiciones	El path de la consola debe situarse en el mismo directorio donde se encuentra la herramienta
Proceso	1. Se ejecuta la herramienta mediante el siguiente comando: java -jar “nombredelaherramienta”
Postcondiciones	Se muestran las opciones de la herramienta
Resultado	Verificado
Trazabilidad	RF 1, RF 3, RF 11, RNF 1, RNF 2, RNF 3

Tabla 23 Prueba 01

Prueba 2	
Objetivo	Se comprueba que la herramienta muestra la información adicional de la herramienta, con el comando especificado
Precondiciones	El path de la consola debe situarse en el mismo directorio donde se encuentra la herramienta Se debe de conocer con anterioridad la opción de mostrar la información adicional, siguiendo los pasos pertinentes
Proceso	1. Se ejecuta la herramienta mediante el comando: java -jar “nombredelaherramienta” -aboutus
Postcondiciones	Se debe mostrar en la consola, por lo menos, los desarrolladores que han contribuido en la herramienta
Resultado	Verificado
Trazabilidad	RF 4, RF 11, RNF 1, RNF 2, RNF 3

Tabla 24 Prueba 02

Prueba 3	
<b>Objetivo</b>	Se comprueba que la herramienta muestra todas las acciones que se encuentran disponibles, con una etiqueta asociada a cada una distinta entre sí.  A su vez, se comprueba que estas son únicas y se refieren a una única acción.
<b>Precondiciones</b>	El path de la consola debe situarse en el mismo directorio donde se encuentra la herramienta.  Se debe de haber comprobado con anterioridad la etiqueta que corresponde a la opción de mostrar las acciones disponibles
<b>Proceso</b>	1. Se ejecuta la herramienta mediante el comando: java -jar “nombredelaherramienta” –actions
<b>Postcondiciones</b>	Se muestran todas las acciones disponibles con todas las etiquetas correspondientes
<b>Resultado</b>	Verificado
<b>Trazabilidad</b>	RF 5, RF 6, RNF 1, RNF 2, RNF 3

Tabla 25 Prueba 03

Prueba 4	
<b>Objetivo</b>	Se comprueba que todas las acciones disponibles, muestran mediante la etiqueta correspondiente, la descripción de esta misma, junto con los argumentos que requiere para funcionar correctamente.
	A su vez, se comprueba que una de las acciones describe que aplica la técnica de memorización a una función matemática pura, mediante reescritura de código utilizando un transpilador y con el uso de una gramática LL.
	Comprobar el control de errores de la herramienta
<b>Precondiciones</b>	<p>El path de la consola debe situarse en el mismo directorio donde se encuentra la herramienta</p> <p>Se debe de haber comprobado con anterioridad la etiqueta que corresponde a la opción de mostrar las acciones disponibles y haber escogido la que realiza memorización</p>
<b>Proceso</b>	<ol style="list-style-type: none"> <li>1. Se ejecuta el comando correspondiente a la memorización con la etiqueta <code>-help</code></li> <li>2. Se comprueba que una de las acciones describe que aplica la técnica de memorización a una función matemática pura, mediante reescritura de código utilizando un transpilador y con el uso de una gramática LL.</li> <li>3. Se prueba el control de errores introduciendo: <ul style="list-style-type: none"> <li>- 1 etiqueta incorrecta</li> <li>- 2 etiquetas incorrectas</li> <li>- La primera etiqueta correcta y la segunda incorrecta</li> <li>- La primera incorrecta y la segunda correcta</li> </ul> </li> </ol>
<b>Postcondiciones</b>	<p>Se muestran todas las características todas las acciones por pantalla, según el orden que se hayan ejecutado los comandos correspondientes.</p> <p>Todas las opciones erróneas coinciden con los errores especificados.</p>
<b>Resultado</b>	Verificado
<b>Trazabilidad</b>	RF 5, RF 6, RF 7, RF 9, RF 11, RNF 1, RNF 2, RNF 3, RNF 4, RNF 5

Tabla 26 Prueba 04

Prueba 5	
<b>Objetivo</b>	Se comprueba que la selección de los métodos, a los cuales se los quiere aplicar memorización, se realiza mediante la colocación de la etiqueta “@memorization” en su cabecera
<b>Precondiciones</b>	El path de la consola debe situarse en el mismo directorio donde se encuentra la herramienta  Se debe de haber comprobado con anterioridad la etiqueta que corresponde a la opción de mostrar las acciones disponibles y haber escogido la que realiza memorización
<b>Proceso</b>	1.- Se escoge un fichero que contenga la gramática java y se indica en la cabecera de un método “@memorization”  2.- Se selecciona la acción “memorization” y se convierten los datos  3.- Se comprueba que se ha reescrito el método indicado
<b>Postcondiciones</b>	Se ha creado un fichero nuevo, llamado nombredelfichero_memorization.java, que contiene el método reescrito
<b>Resultado</b>	Verificado
<b>Trazabilidad</b>	RF 8, RF 9, RF 11, RNF 1, RNF 2, RNF 3

Tabla 27 Prueba 05

Prueba 6	
<b>Objetivo</b>	Se comprueba que la acción de memorización se puede aplicar a funciones con varios argumentos
<b>Precondiciones</b>	El path de la consola debe situarse en el mismo directorio donde se encuentra la herramienta Se debe de haber comprobado con anterioridad la etiqueta que corresponde a la opción de mostrar las acciones disponibles y haber escogido la que realiza memorización
<b>Proceso</b>	1.- Se escoge un fichero que contenga la gramática java y se indica en la cabecera de un método “@memorization”. A su vez, el método deberá tener más de un argumento. 2.- Se selecciona la acción “memorization” y se convierten los datos 3.- Se comprueba que se ha reescrito el método indicado
<b>Postcondiciones</b>	Se ha creado un fichero nuevo, llamado nombredelfichero_memorization.java, que contiene el método reescrito
<b>Resultado</b>	Verificado
<b>Trazabilidad</b>	RF 9, RF 10, RF 11, RNF 1, RNF 2, RNF 3

Tabla 28 Prueba 06

Prueba 7	
<b>Objetivo</b>	Se comprueba que la acción de memorización, es compatible con métodos con argumentos múltiples y diferentes tipos primitivos
<b>Precondiciones</b>	El path de la consola debe situarse en el mismo directorio donde se encuentra la herramienta  Se debe de haber comprobado con anterioridad la etiqueta que corresponde a la opción de mostrar las acciones disponibles y haber escogido la que realiza memorización
<b>Proceso</b>	1.- Se escoge un fichero que contenga la gramática java y se indica en la cabecera de un método “@memorization”. A su vez, el método deberá tener más de un argumento.  2.- Se selecciona la acción “memorization” y se convierten los datos  3.- Se comprueba que se ha reescrito el método indicado  4.- Se repite el proceso hasta hayan sido probados todos los tipos de argumentos primitivos
<b>Postcondiciones</b>	Se ha creado un fichero nuevo, llamado nombredelarchivo_memorization.java, que contiene el método reescrito
<b>Resultado</b>	Verificado
<b>Trazabilidad</b>	RF 9, RF 10, RF 11, RF 12, RNF 1, RNF 2, RNF 3

Tabla 29 Prueba 07

# Capítulo 5

## Planificación y Presupuesto

Una vez verificados todas las pruebas diseñadas para la herramienta, y continuando con el guion establecido en el apartado 1.3.- Resto del documento.

Por un lado, se presenta la planificación diseñada al principio del proyecto y la planificación resultante una vez ha sido finalizado el proyecto, así como unas conclusiones de porqué se ha producido este desfase.

Por otro lado, basándose en la planificación expuesta, se presenta el presupuesto calculado para la realización del proyecto.

### 5.1.- Planificación

Una vez presentados todos los aspectos relacionados con la herramienta y con el objetivo de poder presentar esta herramienta como una solución sólida ante los problemas planteados, se continúa con la planificación de las diferentes tareas que han sido necesarias en el proceso de desarrollo de la herramienta.

Para realizar una vista simple y explicativa de la planificación realizada, se ha decidido utilizar un Diagrama de Gantt, el cuál, es uno de los más comunes en estos, ya que no se tiene un número superior a unas 25 tareas, a partir de las cuales, se aconsejan buscar otras soluciones.

A continuación, se muestran 2 diagramas:

- **Planificación estimada**, la cual, muestra la planificación que se realizó antes de comenzar el proyecto, con el objetivo de llegar con el suficiente margen a la fecha final
- **Planificación Real**, realizadas con fecha posterior a la finalización del trabajo fin de carrera, mostrando el tiempo real utilizado para el mismo.

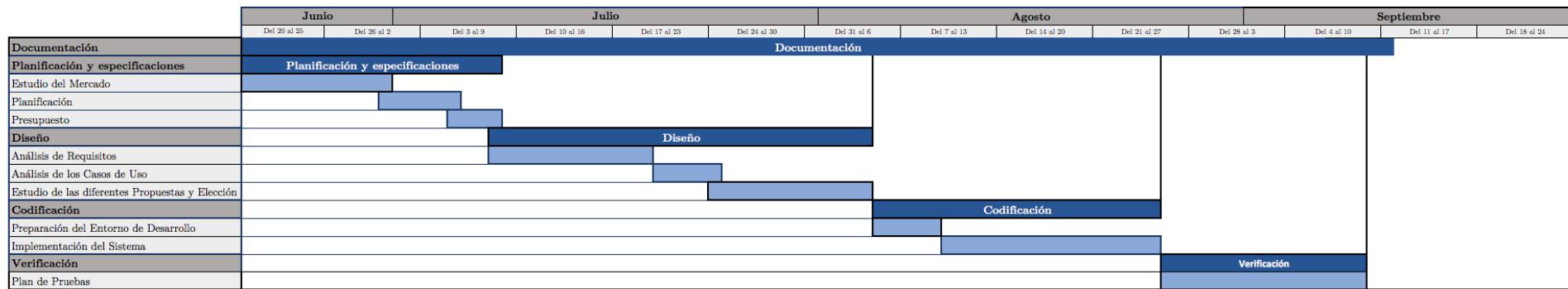


Ilustración 26 Planificación Estimada

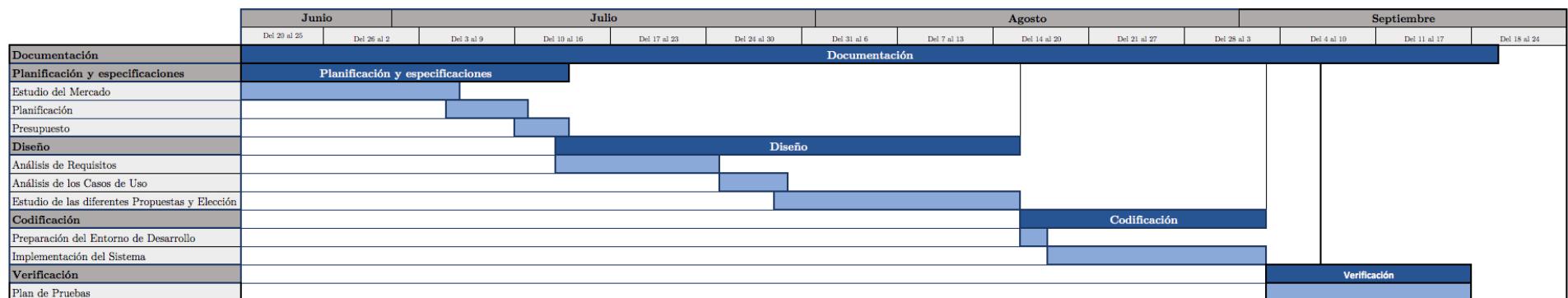


Ilustración 27 Planificación Real

Una vez observados los dos diagramas, se puede ver se ha producido un retraso en la fecha final del proyecto, produciéndose variaciones en el tiempo estimado para cada una de las tareas.

La justificación a este retraso se debe al hecho a la dificultad para entender los términos referentes a la teoría del lenguaje que se encuentra relacionado este TFG, así como la documentación de las herramientas, ya que esta, es muy compleja, ya que ofrece un vasto catálogo de posibilidades que no son fácil de comprender la ventaja que puede obtenerse con su uso.

A su vez, se puede observar una mejora en el tiempo estimado para la preparación del sistema, debido a la fortuna de encontrar un repositorio con un arquetipo Maven [31], que facilitó la instalación de las herramientas necesarias, dependientes de ANTLR, así como su posterior configuración. Lo referente a la configuración del mismo se encuentra explicado con mayor detalle en el Anexo I: Manual de Instalación del Entorno de Desarrollo.

## 5.2.- Presupuesto

Una presentada la planificación, se procede a comentar todo lo referente al aspecto económico referentes al proyecto. Para ello, se van presentar varios subapartados con el fin de facilitar la comprensión de cada uno de ellos.

### 5.2.1.- Alcance del Proyecto

En primer lugar, y para poder realizar una estimación lo más fiel posible a la realidad, se considera necesario conocer el alcance del proyecto, tanto en horas como en semanas. Para ello, es necesario apoyarse en el Diagrama de Gantt que muestra el tiempo estimado que ocupan las tareas.

Planificación del proyecto	
Duración del proyecto en horas	327
Duración del proyecto en semanas	12,111111
Horas del trabajo por semana	27
Horas de trabajo al mes	117,45

Tabla 30 Planificación del Proyecto

### 5.2.2.- Coste del Personal

Posteriormente, y una vez conocido el alcance del proyecto, se procede al desglose por rol, de los costes del personal que es necesario para el proyecto, así como lo referente a los impuestos que deben ser abonados por parte del contratante.

Salarios		
Rol	Salario Bruto Anual	Salario Bruto Mensual
Jefe de Proyecto	35.000	2.916,67
Analista	28.000	2.333,33
Programador	24.000	2.000,00
Gestor de Pruebas	24.000	2.000,00

Tabla 31 Salarios por Rol al Mes

Cuota Patronal Mensual							
Rol	Seguridad Social		Desempleo		Formación Profesional		Total Cuota Patronal
	%	€	%	€	%	€	
Jefe de Proyecto	23,6	688,333333	5,5	160,416667	0,6	17,5	0,2 5,83333333
Analista	23,6	550,666667	5,5	128,333333	0,6	14	0,2 4,66666667
Programador	23,6	472	5,5	110	0,6	12	0,2 4
Gestor de Pruebas	23,6	472	5,5	110	0,6	12	0,2 4

Tabla 32 Cuota Patronal

Una vez se ha mostrado los costes mensuales, se muestran los costes que supondría para el proyecto, siendo asignado a cada rol las horas correspondientes a cada etapa del proyecto:

Coste total del personal				
Rol	Coste Mensual del Trabajador	Horas Asignadas a cada Rol	Coste de la hora trabajada	Coste Total de Horas Persona
Jefe de Proyecto	3.788,75	74	32,3	2387,1
Analista	3.031,00	108	25,8	2787,1
Programador	2.598,00	81	22,1	1791,7
Gestión de Pruebas	2.598,00	60	22,1	1327,2
<b>Coste Total del Personal</b>				<b>8293,2</b>

Tabla 33 Coste total del Personal

### 5.2.3.- Coste de los Equipos de Desarrollo

Tras mostrar un detallado desglose de los aspectos referentes al coste del personal, se continúa mostrando los costes de los equipos necesarios para poder poner en marcha el proyecto, con su correspondiente amortización en el tiempo que confiere el proyecto.

Equipos de desarrollo y documentación					
Producto	Amort. Anual	Amort. Semanal	Semanas	Total	
Apple Macbook Pro Retina 13' 2017 16 Gb Ram	1985,0	992,5	18,4	12,1	222,6
Asus Monitor VX239	134,5	67,2	1,2	12,1	15,1
Ratón Asus WT425	15,0	7,5	0,1	12,1	1,7
Adaptador Apple HDMI a Type - C	134,5	67,2	1,2	12,1	15,1
Belkin - Mini Hub USB-C de 4 puertos (tipo C)	35,5	17,8	0,3	12,1	4,0
<b>Coste Total de los Equipos</b>					<b>258,42 €</b>

Tabla 34 Costes de los Equipos y Amortización

### 5.2.4.- Coste de las licencias y otros costes

Finalmente, se muestran los costes referentes tanto a las licencias necesarias como otros costes que no se han clasificado en ningún tipo.

<b>Coste de las licencias de software</b>	
SteelMouse	15,00 €
Coste de las licencias de software	0,00 €
<b>Total de Coste los equipos del proyecto</b>	<b>15,00 €</b>

Tabla 35 Coste de las Licencias Software

<b>Otros Costes</b>	
Alquiler del Local (Luz y Agua)	800,00 €
Consumibles de Ofimática	20,00 €
<b>Total de Coste los equipos del proyecto</b>	<b>820,00 €</b>

Tabla 36 Otros Costes

### 5.2.5.- Coste Total del Proyecto

Finalmente, se suman todos los costes anteriormente mostrados y se aplican tanto el margen de riesgo, como el de beneficio y el IVA, Tabla 37 Coste Total del Proyecto IVA Incluido.

<b>Total</b>		
Coste Total		9.386,60 €
Margen de Riesgo	10%	938,66 €
Margen de beneficio	17%	1.595,72 €
Coste de proyecto sin IVA		11.920,98 €
IVA	21%	2.503,41 €
<b>COSTE TOTAL DEL PROYECTO IVA INCLUIDO</b>		<b>14.424,38 €</b>

Tabla 37 Coste Total del Proyecto IVA Incluido

Cabe mencionar, que debido al hecho de que el proyecto ha sufrido un retraso en su entrega, se deberá restar, al margen de riesgo y, en el caso de que sea necesario, al margen de beneficio, los costes asociados a este retraso.

### 5.3.- Impacto Socio-Económico

Una vez comentado el presupuesto, se finaliza el capítulo con el Impacto Socio-Económico que se espera en el caso de que se llegue a desarrollar este proyecto, tal y cómo se plantea.

Tal y cómo se puede observar en la actualidad, la corrección de errores que son descubiertos en diferentes tecnologías puede ser crucial para disminuir el impacto que puede producirse.

En estos últimos días se ha presentado un error en la tecnología Bluetooth [32], el cual permite tomar el control de los dispositivos vulnerables.

La velocidad con la que estos errores deben de ser resueltos es crucial para reducir el número de posibles infectados por este error.

A su vez, la actualización de librerías con una gran dependencia, como, por ejemplo, API Rest, puede ser un verdadero problema para el mantenimiento de las aplicaciones, ya que un cambio en una interfaz, pueden suponer miles de líneas de código.

Por todo ello, **se considera como un Impacto muy positivo el desarrollo de esta herramienta**, ya que permitiría ahorrar mucho tiempo y agilizar procesos de actualización de código rutinarios, pero de gran importancia para el mundo actual.

# Capítulo 6

## Conclusiones y Trabajos Futuros

Una vez finalizada la explicación referente al Presupuesto y Planificación del Proyecto, se concluye el presente documento con una serie de conclusiones, tanto del producto como de la experiencia personal, en el proceso llevado a cabo.

A su vez, se incluyen una serie de trabajos futuros que se han considerado interesantes para continuar el desarrollo de la herramienta.

### 6.1.- Conclusiones

Tras una breve explicación del contenido del capítulo actual, se procede a comentar las conclusiones extraídas del proceso llevado a cabo para conseguir desarrollar la herramienta.

Con el objetivo de presentar unas conclusiones ordenadas, explicativas y claras, se ha dividido el actual apartado en dos subapartados:

- **Conclusiones de Producto:** Se comentan todos los aspectos relacionados con la herramienta.
- **Conclusiones Personales:** Se comentan todos los aspectos relacionadas con la experiencia personal del alumno que ha llevado a cabo este TFG.

### 6.1.1.- Producto

Tal y como se ha comentado, se comienzan las conclusiones obtenidas del producto desarrollado, así como todos aspectos relacionados con la misma.

En primer lugar, se ha podido observar la potencia y la gran cantidad de posibilidades que ofrece la reescritura de código. Tener la posibilidad de que una herramienta permita un ahorro de tiempo para tareas rutinarias y mecánicas facilita mucho el trabajo.

Si este concepto de herramienta, se consigue que se vaya popularizando, se podría conseguir un gran ahorro en tareas rutinarias.

En segundo lugar, y para finalizar, comentar que el uso de ANTLR, y, de manera ligada, el uso de gramáticas tipo LL, puede facilitar en gran medida el crecimiento de la utilidad presentada, ya que, por un lado, ANTLR ofrece grandes facilidades, como es el hecho de poder ver como se genera el árbol sintáctico y la integración con Eclipse. Dichos hechos, fueron comentados con anterioridad en el apartado 3.5.5.- Comparativa con las distintas soluciones y la propuesta, siendo uno de los motivos por los que se escogió desarrollar esta herramienta y, una vez utilizada, se reafirman dichos hechos.

Por otro lado, el uso de gramáticas de tipo LL, permite la utilización de acciones a media regla gracias la manera que se aplican las reglas sintácticas. Estas se han mostrado muy útiles e interesantes en diferentes escenarios planteados, aunque finalmente no se utilizaron. Además, cabe comentar que una vez se comenzó a utilizar dicha gramática, siendo comparadas con las gramáticas tipo LR, se mostraron más comprensibles y fáciles de entender. Capítulo 0

### 6.1.3.- Personales

Una vez comentadas las conclusiones que se refieren a los aspectos del producto, se continúa con las conclusiones extraídas por el alumno.

En primer, merece la pena comentar el hecho de que el presente alumno no cursó la especialidad de “Ciencias de la Computación”, sino “Ingeniería de Computadores”, por lo que este TFG se puede catalogar como reto, obligando a salir de la zona de confort de los conceptos aprendidos en dicha mención.

A su vez, el hecho comentado de la mención seleccionada, más que un impedimento, fue una motivación, ya que, ha permitido conocer el

funcionamiento de un compilador y las diferentes fases que lo conforman.

Además de los compiladores, se ha conocido el concepto de Transpilador y el potencial que presenta a la presente situación actual, donde las líneas de código de las herramientas no paran de crecer, y realizar un cambio que resulte recurrente por un fallo de seguridad, por ejemplo, puede volverse muy complicado y crítico, ya que una persona, ante tantas líneas, puede confundirse y no modificar alguna parte del código.

A su vez, y para finalizar estas conclusiones, se reforzaron los conocimientos adquiridos sobre gramáticas, permitiendo conocer más en profundidad las diferentes características que lo conforman, así como los diferentes casos que puede ser interesante para un parser, utilizar una gramática y qué tipo puede ser más interesante.

## 6.2.- Trabajos Futuros

Una vez se han comentados las conclusiones más destacables que se han ido encontrado en el proceso de creación del presente TFG, con el objetivo de motivar la continuación de la herramienta presentada, así como el concepto de la reescritura de código.

Para ello, se van a presentar diferentes ideas que se consideran interesantes de aplicar en la presente herramienta.

- **Soportar todo tipo de argumentos.** Tal y como se comenta en el apartado 3.6.- Implementación, no se incluye soporte a variables no primitivas, por tener que manejar la llamada al método *equals*, por lo que se considera importante añadir este soporte a la herramienta.
- **Una Red de Neuronas que decida qué métodos aplicar memorización.** Tal y como se comenta en el apartado 2.2.- Estudio de la Actualidad, el desarrollador debe de seleccionar qué métodos desea que se aplique memorización, por lo que cabe la posibilidad que la técnica se aplique de una manera incorrecta. Por ello, sería muy interesante que la herramienta directamente seleccionará por si misma qué funciones se les aplicaría, utilizando diferentes análisis, como test de rendimiento etc.
- **Reescritura de llamadas a función y variables.** Actualmente, las API se encuentran en constante evolución, debido en parte a los

problemas de seguridad que se encuentran constantemente en las mismas. Es muy frecuente que un simple argumento, con un valor u otro, implique un fallo de seguridad en una aplicación. En base a esto, se considera interesante implementar el cambio de valor de los argumentos indicando a qué función pertenecen.

A su vez, que se pudiera cambiar una estructura de código, es decir, una función con sus argumentos correspondientes que se encontrara obsoleta, por ejemplo, por otra con sus respectivos argumentos.

- **Integración con los IDE más populares.** Para una mayor facilidad de uso y popularidad, es interesante que la herramienta presentada fuese un *plugin* de diferentes IDE, Eclipse, por ejemplo. Así estaría más al alcance de los usuarios.

# Anexo I: Manual de Instalación del Entorno de Desarrollo.

Con el objetivo de que todo desarrollador interesado en aumentar las funcionalidades presentadas en el actual Trabajo Fin de Carrera, se presenta el siguiente repositorio de GitHub [33], en el cual, se encuentran un archivo .zip, que contiene todo lo necesario para la instalación, así como un tutorial guiado para realizar la instalación:

<https://github.com/jpavlich/antlr4-tutorial/blob/master/doc/instalacion.md>



## Anexo II: Tablas para Impresión

En el presente Anexo se presentan las tablas, que, por motivos de tamaño, ha sido necesaria su rotación para una lectura adecuada en Formato PDF.

En el caso de que el presente documento haya sido imprimido, se presenta este Anexo para facilitar la lectura de las tablas en dicho formato.

RF / RNF	RNF 5	RNF 4	RNF 3	RNF 2	RNF 1	RF 12	RF 11	RF 10	RF 9	RF 8	RF 7	RF 6	RF 5	RF 4	RF 3	RF 2	RF 1	RF	RF 2
RF 2																			
RF 1	x																		
RF 3																			
RF 4																			
RF 5					x														
RF 6	x			x		x													
RF 7						x		x										x	
RF 8				x	x			x									x	x	
RF 9	x					x			x	x	x								
RF 10						x		x				x							
RF 11	x	x		x	x		x		x				x						
RF 12								x			x								
RNF 1	x	x	x	x	x	x					x			x					
RNF 2											x			x			x		
RNF 3												x			x				
RNF 4							x											x	
RNF 5							x	x									x		x

Tabla 38 Matriz de Trazabilidad para Impresión

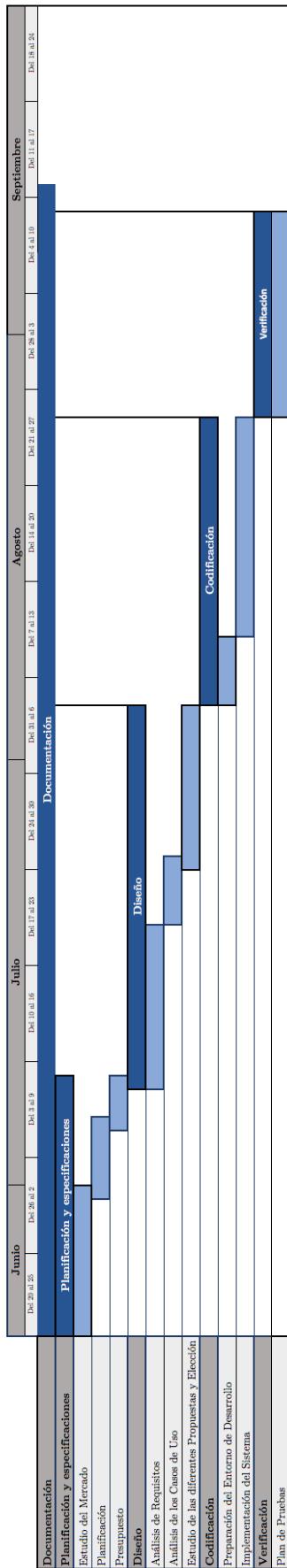


Ilustración 28 Planificación Estimada para Impresión

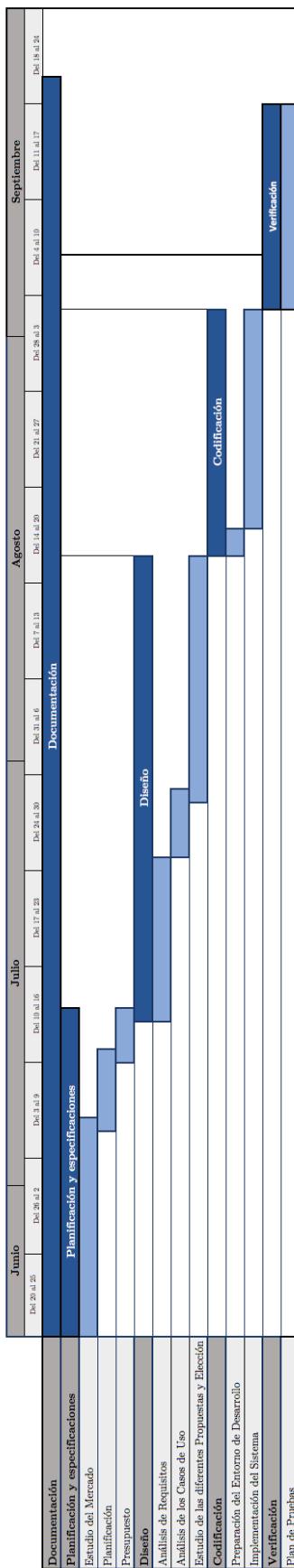


Ilustración 29 Planificación Real para Impresión





## Anexo III: Repositorio.

Con el objetivo de presentar todo el desarrollo hecho y mostrar el código del proyecto, por si el actual lector está interesado en el proyecto, puede consultar el siguiente enlace:

<https://github.com/estole95/TFG>



# Bibliografía

- [1] S. Fernández, «<https://www.youtube.com/>,» 06 04 2014. [En línea]. Available: <https://www.youtube.com/watch?v=5vtf8k3Ibgo>. [Último acceso: 15 09 2017].
- [2] GitHub, «<http://githut.info/>,» 2017. [En línea]. Available: <http://githut.info/>. [Último acceso: 05 09 2017].
- [3] Tiobec, «<https://www.tiobe.com/>,» 09 2017. [En línea]. Available: <https://www.tiobe.com/tiobe-index/>. [Último acceso: 06 09 2017].
- [4] J. L. Bentley, Writing Efficient Programs, Prentice-Hall Software Series, 1982.
- [5] J. Titcomb, «[telegraph](http://www.telegraph.co.uk/technology/2017/06/29/iphone-turns-10-ways-changed-world1/),» 29 Junio 2017. [En línea]. Available: <http://www.telegraph.co.uk/technology/2017/06/29/iphone-turns-10-ways-changed-world1/>. [Último acceso: 2017 Agosto 7].
- [6] AppleInsider Staff, «[Apple Insider](http://appleinsider.com/articles/08/07/10/apples_app_store_launches_with_more_than_500_apps),» 2008 Julio 10. [En línea]. Available: [http://appleinsider.com/articles/08/07/10/apples\\_app\\_store\\_launches\\_with\\_more\\_than\\_500\\_apps](http://appleinsider.com/articles/08/07/10/apples_app_store_launches_with_more_than_500_apps). [Último acceso: 2017 Agosto 7].
- [7] D. Pérez, «[El Androide Libre](https://elandroidelibre.elespanol.com/2015/01/google-play-supera-la-appstore-en-cantidad-de-aplicaciones-y-desarrolladores.html),» 14 Enero 2015. [En línea]. Available: <https://elandroidelibre.elespanol.com/2015/01/google-play-supera-la-appstore-en-cantidad-de-aplicaciones-y-desarrolladores.html>. [Último acceso: 2017 Agosto 7].
- [8] «<https://ruslanspivak.com/>,» 27 04 2017. [En línea]. Available: <https://ruslanspivak.com/lsbasi-part13/>. [Último acceso: 07 09 2017].
- [9] Gartner, «<http://www.gartner.com/>,» 04 01 2017. [En línea]. Available: <http://www.gartner.com/newsroom/id/3560517>. [Último acceso: 06 09 2017].
- [10] xda-developers, «<https://www.xda-developers.com/i/>,» 02 09 2016. [En línea]. Available: <https://www.xda-developers.com/infograph>-

- how-the-xda-community-views-phone-costs-screen-size-storage-space-and-more/. [Último acceso: 06 09 2017].
- [11] GameBench, «<https://blog.gamebench.net/>,» 15 12 2014. [En línea]. Available: <https://blog.gamebench.net/samsung-galaxy-note-4-beats-every-other-phone-asphalt-8>. [Último acceso: 06 09 2017].
- [12] OnePlus, «<https://oneplus.net/>,» [En línea]. Available: <https://oneplus.net/es/5/specs>. [Último acceso: 07 09 2017].
- [13] P. Neumann, «<http://pages.cpsc.ucalgary.ca/>,» 2017 10 2004. [En línea]. Available: <http://pages.cpsc.ucalgary.ca/~saul/pmwiki/uploads/Main/topic-neumann.pdf>. [Último acceso: 11 9 2017].
- [14] Agencia Española de Protección de Datos, «<http://www.agpd.es/>,» 05 03 2011. [En línea]. Available: [http://www.agpd.es/portalwebAGPD/canaldocumentacion/legislacion/estatal/common/pdfs/2014/Ley\\_Organica\\_15-1999\\_de\\_13\\_de\\_diciembre\\_de\\_Proteccion\\_de\\_Datos\\_Consolidado.pdf](http://www.agpd.es/portalwebAGPD/canaldocumentacion/legislacion/estatal/common/pdfs/2014/Ley_Organica_15-1999_de_13_de_diciembre_de_Proteccion_de_Datos_Consolidado.pdf). [Último acceso: 11 09 2017].
- [15] Ministerio de Educación, Cultura y Deport, «<https://www.mecd.gob.es/>,» [En línea]. Available: <https://www.mecd.gob.es/cultura-mecd/areas-cultura/propiedadintelectual/la-propiedad-intelectual/preguntas-mas-frecuentes/la-propiedad-intelectual.html>. [Último acceso: 11 09 2017].
- [16] Open Source Initiative, «<https://opensource.org/>,» [En línea]. Available: <https://opensource.org/licenses/BSD-3-Clause>. [Último acceso: 12 09 2017].
- [17] Ministerio de Hacienda y Administraciones Públicas, «<https://administracionelectronica.gob.es/>,» [En línea]. Available: [https://administracionelectronica.gob.es/pae\\_Home/pae\\_Documentacion/pae\\_Metodolog/pae\\_Metrica\\_v3.html#.Wbq2SK0ryi4](https://administracionelectronica.gob.es/pae_Home/pae_Documentacion/pae_Metodolog/pae_Metrica_v3.html#.Wbq2SK0ryi4). [Último acceso: 14 09 2017].
- [18] M. Cillero, «<https://manuel.cillero.es/>,» [En línea]. Available: <https://manuel.cillero.es/doc/metrica-3/>. [Último acceso: 14 09 2017].

- 
- [19] JavaCC Developers, «GitHub Inc,» 4 Agosto 2017. [En línea]. Available: <https://github.com/javacc/javacc>. [Último acceso: 21 Agosto 2017].
  - [20] S. C. Johnson, «<http://dinosaur.compilertools.net/>,» 9 07 1974. [En línea]. Available: <http://dinosaur.compilertools.net/yacc/index.html>. [Último acceso: 25 08 2017].
  - [21] M. . E. Lesk y E. S. , «<http://dinosaur.compilertools.net/>,» Desconocida, - - -. [En línea]. Available: <http://dinosaur.compilertools.net/lex/index.html>. [Último acceso: 07 09 2017].
  - [22] T. Niemann, «<http://epaperpress.com/>,» Desconocida, 07 08 2017. [En línea]. Available: <http://epaperpress.com/lexandyacc/download/LexAndYaccTutorial.pdf>. [Último acceso: 25 08 2017].
  - [23] D. Grune y C. J.H., Parsing Techniques, Segunda ed., Springer, 2008.
  - [24] C. D. y R. S. , «<http://dinosaur.compilertools.net/>,» Desconocida, 11 1995. [En línea]. Available: <http://dinosaur.compilertools.net/bison/index.html>. [Último acceso: 2017 09 7].
  - [25] V. Paxson, «<http://dinosaur.compilertools.net/>,» Desconocida, - 03 1995. [En línea]. Available: <http://dinosaur.compilertools.net/flex/index.html>.
  - [26] R. B. Hernández, «<http://webdiis.unizar.es/>,» 2004. [En línea]. Available: [http://webdiis.unizar.es/asignaturas/LGA/material\\_2004\\_2005/Intro\\_Flex\\_Bison.pdf](http://webdiis.unizar.es/asignaturas/LGA/material_2004_2005/Intro_Flex_Bison.pdf). [Último acceso: 7 09 2017].
  - [27] G. Tomassetti, «<https://tomassetti.me/>,» Desconocida, 08 Marzo 2017. [En línea]. Available: <https://tomassetti.me/antlr-mega-tutorial/>. [Último acceso: 2017 09 07].
  - [28] «<https://github.com/>,» 04 2017. [En línea]. Available: <https://github.com/antlr/antlr4/blob/master/doc/index.md>. [Último acceso: 07 09 2017].

- [29] J. Haberman, «<http://blog.reverberate.org/>,» 22 07 2013. [En línea]. Available: <http://blog.reverberate.org/2013/07/ll-and-lr-parsing-demystified.html>. [Último acceso: 27 08 2017].
- [30] T. Par y S. Harwell, «<https://github.com/antlr/>,» 07 2017. [En línea]. Available: <https://github.com/antlr/grammars-v4>. [Último acceso: 12 09 2017].
- [31] J. Pavlich-Mariscal, «<https://github.com/jpavlich/>,» 2016. [En línea]. Available: <https://github.com/jpavlich/antlr4-tutorial>. [Último acceso: 15 08 2017].
- [32] Oficina de Seguridad del Internauta, «<https://www.osi.es/>,» 13 09 2017. [En línea]. Available: <https://www.osi.es/es/actualidad/avisos/2017/09/fallos-de-seguridad-en-bluetooth-pueden-ser-utilizados-para-infectar>. [Último acceso: 13 09 2017].
- [33] J. P. Mariscal, «<https://github.com/jpavlich/>,» 13 Abril 2016. [En línea]. Available: <https://github.com/jpavlich/antlr4-tutorial/blob/master/doc/installacion.md>. [Último acceso: 15 09 2017].

