UNIVERSIDAD
REY JUAN CARLOS

MÁSTER OFICIAL EN SISTEMAS TELEMÁTICOS E INFORMÁTICOS

Curso Académico 2013/2014

Trabajo Fin de Máster

FUNCTIONCOUNT: HERRAMIENTA PARA CONTAR FUNCIONES/MÉTODOS

Autor : Crisselda Copa Choque

Tutor : Gregorio Robles

Índice

1. Resumen

Con la aparición de software libre, la motivación para el desarrollo de software aumentó considerablemente y muchas herramientas de software de gran utilidad fueron desarrolladas gracias a este movimiento en los últimos años.

Hoy en día existen muchas herramientas desarrolladas a nuestro alcance, herramientas de software libre que podemos probarlos y experimentar su funcionalidad sin ningún compromiso. Como también tenemos acceso a su código fuente, con lo que podemos hacer modificaciones si así lo deseamos, sin tener que consultar al propietario del proyecto. A muchos programadores esto les ahorro mucho trabajo, ya que no tuvieron que empezar el proyecto desde cero. También podemos usar estos proyectos ya desarrollados, para poder realizar analíticas sobre ellos si es necesario.

Entre las herramientas comentadas anteriormente, existen herramientas que sirven para poder analizar el código fuente de un proyecto como ejemplo podemos citar a *SLOCCount*. Es una herramienta que cuenta la cantidad de líneas de código fuente que tiene un proyecto, con esta información también nos da resultados de los lenguajes en los que fue escrito el proyecto, sabiendo la cantidad de líneas realiza un cálculo de estimación de tiempo, coste y número de desarrolladores.

Herramientas como ésta son de gran utilidad a personas que están constantemente analizando, diseñando, desarrollando, en fin todo lo que tenga que ver con el desarrollo de software, ya que ayuda a tener una primera impresión del tamaño del proyecto. Este tipo de herramientas lo utilizan considerablemente las comunidades de software libre, las cuales están siempre en contacto con lo que se denomina software.

Las comunidades de software libre se encargan de colaborar y compartir experiencias con los usuarios y estos a la vez obtienen nuevas ideas según las consultas de los usuarios, existe una relación más cercana entre las comunidades y los usuarios ambos se ayudan mutuamente.

En base a lo que se menciono anteriormente surge la motivación de realizar esta herramienta de software *functioncount*, cuya funcionalidad sera contar las funciones de un proyecto. Con esta información pretendemos ayudar a tener una idea más clara y más precisa de la complejidad de un software a la hora de realizar una analítica de software. Con la cantidad de líneas puedo determinar con exactitud el tamaño de un proyecto pero no así su complejidad, con la ayuda de

esta herramienta y del SLOCCount los desarrolladores podrán tener una idea de la complejidad de un proyecto. Además con la cantidad de funciones de un software podemos determinar el tiempo, coste y los desarrolladores que intervinieron, Teniendo esta información podemos hacer comparativas con el SLOCCount y de esta manera nos ayudaría a tomar decisiones mas precisas en funcion a tiempo, coste y desarrolladores.

Esta herramienta pretende ser una ayuda a las comunidades de software libre y a todos los usuarios que quieran realizar analíticas de software. Sera de gran ayuda aquellos que tienen que manipular grandes volúmenes de código fuente.

2. Introducción

Actualmente estamos viviendo en un tiempo donde la tecnología esta teniendo un impacto impresionante en la sociedad.

Existen grandes avances tecnológicos y esto esta contribuyendo al usuario grandemente, como ejemplo de un avance tecnológico podemos nombrar **Internet**.

Con la aparición del Internet prácticamente todo el mundo puede tener acceso a todo tipo de información, por ejemplo las noticias, artículos, documentos, softwares (programas informáticos), y mucho más. Internet a contribuido también con el Software Libre para que sea expandido por todo el mundo.

Software Libre esta teniendo una grande evolución, una grande extensión a nivel mundial, porque nos ofrece fiabilidad y adaptabilidad. Ya que es plenamente operativo con cualquier tipo de plataforma y tiene una estructura modular, es decir, que ejecuta en cada momento únicamente aquellos procesos que necesitan por lo que es más eficiente y no necesita un hardware que tenga características muy potentes.

Existen muchas comunidades de Software Libre en diferentes lugares, que están en constante desarrollo, brindando apoyo a los usuarios. Gracias a Software Libre es posible desarrollar esta herramienta.

2.1. Software Libre

Se refiere a la libertad que tienen los usuarios y las comunidades para ejecutar, copiar, distribuir, estudiar o analizar, modificar y distribuir lo modificado de un software.

Un software se considera libre cuando el software garantiza estas 4 libertades que se describen a continuación:

- La libertad de ejecutar el programa, para cualquier propósito, esta libertad se la denominó como *libertad 0*, es decir, que tenemos la libertad de utilizar el programa como queramos y donde queramos sin necesidad de pedir permiso al propietario
- la libertad de estudiar como trabaja el programa, y realizar cambios al código fuente del programa de acuerdo a nuestras necesidades. A esto lo denominaron como *libertad 1*; para esta libertad necesitamos tener acceso al código fuente del programa.
- La libertad de redistribuir copias para que pueda ayudar al prójimo. A esto lo denominaron como *libertad 2*.
- Libertad de la redistribución de las mejoras al programa, es decir, publicar las versiones modificadas en general para que se beneficie toda la comunidad. A esto lo denominaron como *libertad 3*.

Un programa es software libre si otorga a los usuarios todas estas libertades de manera adecuada. De lo contrario no es libre.

2.1.1. Ingeniería de Software Libre

Los desarrolladores de software sabemos que, para que el producto (software) nos salga mínimamente bien, es necesario utilizar ingeniería de software. Para ello existen muchas metodologías, patrones que se pueden seguir para un desarrollo eficaz.

La ingeniería de Software Libre es una alternativa, permite que la metodología para el desarrollo de aplicaciones se lleve a cabo de una manera más amplia, ya sea utilizando un enfoque estructurado de análisis y diseño, un enfoque orientado a objetos o algún otro tipo de paradigma. Además no limita a los analistas y diseñadores a una técnica de modelado y diagramación como UML o el modelado estructurado.

Se fundamenta en que se debe trabajar en equipo, con el fin de que se debe fomentar una mayor participación de elementos para un desarrollo óptimo de aplicaciones, sin dejar de lado la utilización de técnicas y herramientas que se mencionaron anteriormente.

La ingeniería de software libre pretende ser una herramienta beneficiosa para la ingeniería de software tradicional y también para el software libre.

Si los cálculos de tiempo de desarrollo y de costes en los proyectos son difíciles de cuantificar, en la actualidad en el mundo del software libre ya existen herramientas que nos permiten analizar y tener una referencia sobre la complejidad del proyecto. *SLOCCount* puede ser una de esas herramientas, ya que nos permite analizar un proyecto: cuenta la cantidad de líneas de código fuente de un proyecto, y con esta información también nos permite determinar la complejidad del proyecto, el tiempo de desarrollo, y mucho más.

2.1.2. Desarrollo de software libre

Para el desarrollo de software se requieren personas, grupos que programen, que realicen análisis, que diseñen, que realicen pruebas. Generalmente se dice que el Software Libre es desarrollado por comunidades, aunque las comunidades son pequeñas en comparación con los usuarios o personas que aportan en el desarrollo de un proyecto.

El software Libre no se basa solamente en comunidades de desarrollo, también se toma en cuenta al usuario, la persona que prueba el producto, que encuentra errores y lo reporta a los encargados del proyecto, muestra a los demás cómo usar el programa. Esto forma parte de un movimiento mundial que defiende la libertad del software.

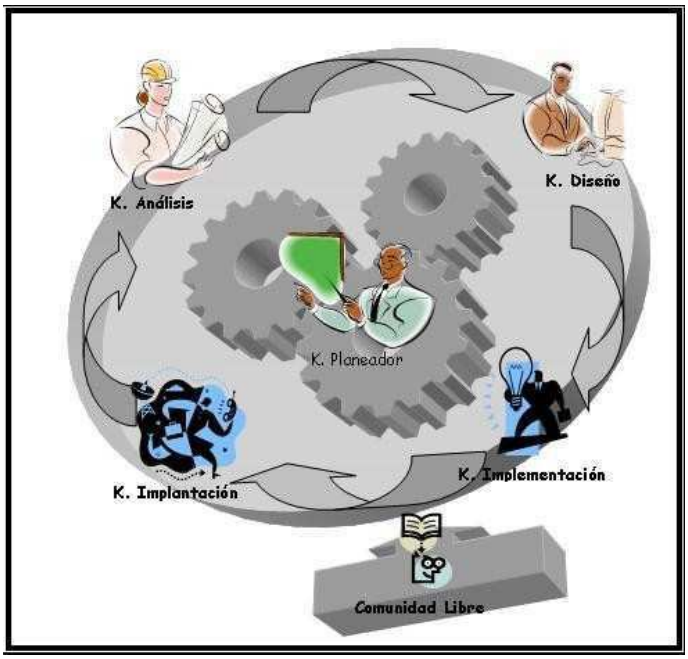


Figura 1: Esquema de trabajo en ingeniería de Software Libre

Bibliografía Mauro Callejas Cuervo. La Ingeniería de Software Libre y sus Herramientas Aplicadas a Proyectos Informáticos.

2.2. Software Analytics

Es un enfoque que permite a los profesionales de software explorar y analizar grandes cantidades de datos, permitiendo presentar una información detallada del proyecto. La cual puede ser útil a la hora de toma de decisiones del análisis, diseño o desarrollo de un proyecto.

Este enfoque esta teniendo un grande avance, más en el campo de software libre ya que el código fuente de los proyecto de software libre se puede obtener con facilidad a través de Internet, y se puede analizar bajo varios criterios. De hecho, hay herramientas libres que ya nos permiten realizar ciertos análisis cómo el *SLOCCount*, que es una herramienta de software libre.

2.3. SLOCCount

Es una herramienta que analiza proyectos de software a través del conteo de líneas de código. Hace la estimación de coste, tiempo y número de desarrolladores, etc. Este software cuenta la cantidad de líneas de código fuente, independiente del lenguaje de desarrollo.

2.4. Python

Python es un lenguaje de programación de muy alto nivel, con una sintaxis muy clara. Sin duda es un lenguaje de programación muy versátil, fuertemente tipado, imperativo y orientado a objetos, aunque contiene también características que lo convierten en un lenguaje de paradigma funcional.

Python es un lenguaje interpretado. A diferencia de C, el código de Python no se ejecuta directamente en la máquina destino, sino que es ejecutado por un software intermedio (o intérprete). Sin embargo, al igual que Java, Python compila el código escrito en lenguaje de alto nivel para obtener un pseudo código máquina (bytecode) que es el que propiamente ejecuta el intérprete.

Python es un lenguaje de programación multiparadigma. Esto significa que más que forzar a los programadores a adoptar un estilo particular de programación, permite varios estilos: programación orientada a objetos, programación imperativa y programación funcional. Otros paradigmas están soportados mediante el uso de extensiones.

2.5. Funciones

Las funciones son la expresión de los algoritmos en un lenguaje de programación específico de la computadora. Se puede utilizar funciones cuando lo necesitemos. Las funciones son trozos de código que se escriben separadamente y que realizan cálculos y tareas específicas. Todos los lenguajes de programación tienen funciones incorporadas, es decir, funciones que realizan cálculos o tareas de uso habitual que han sido ya programados y están disponibles para el usuario como por ejemplo *readline()*, que es una función de Python. Pero, además, todos los lenguajes tienen la posibilidad de que el usuario defina sus propias funciones. Las funciones se crearon para evitar tener que repetir constantemente fragmentos de código fuente.

A continuación presentamos dos tipos de funciones:

Función que busca un elemento en una lista:

```
1. def buscarElemento(lista, elemento):
2.     for i in range(0,len(lista)):
3.        if(lista[i] == elemento):
4.            return i
```

```
Función  que imprime el mayor de dos número

1. def mayorNumero(n1,n2):
2.    if (n1 > n2):
3. print("El mayor es: %s" n1)
4.    else:
5.        print("El mayor es: %s" n2)
```

En estas dos funciones podemos ver que la segunda función tiene mayor cantidad de líneas de código. Pero si analizamos la complejidad de ambas funciones podremos ver que la función más compleja es la primera, ya que al momento de su ejecución la primera tardará más si tomamos el peor caso, que el elemento que se busca se encuentra al final de la lista. Con esto podemos ver que la cantidad de líneas no determina la complejidad.

2.6. MySQL

Es un sistema de gestión de base de datos relacional, multihilo y multiusuario.

MySQL como software libre tiene un esquema de licenciamiento dual. Por un lado se ofrece bajo la *GNU GPL* para cualquier uso compatible con esta licencia, pero para aquellas empresas que quieran incorporarlo en productos privativos deben comprar a la empresa una licencia específica que les permita este uso. *MySQL* esta desarrollado en su mayor parte en *ANSI C*.

Inicialmente *MySQL* carecía de elementos considerados esenciales en las bases de datos relacionales, tales como integridad referencial y transacciones.

A pesar de ello, atrajo a los desarrolladores de páginas web con contenido dinámico, justamente por su simplicidad. Poco a poco los elementos de los que carecía *MySQL* están siendo incorporados tanto por desarrollos internos, como por desarrolladores de software libre. Entre las características disponibles en las últimas versiones se puede destacar:

- Amplio subconjunto del lenguaje *SQL*. Algunas extensiones son incluidas igualmente.
- Disponibilidad en gran cantidad de plataformas y sistemas.

- Posibilidad de selección de mecanismos de almacenamiento que ofrecen diferentes velocidades de operación, soporte físico, capacidad, distribución geográfica, transacciones...
- Transacciones y claves foráneas.
- Conectividad segura.
- Replicación.
- Búsqueda e indexación de campos de texto.

2.7. Ctags

Es una herramienta de programación que genera etiquetas (tag) para los códigos fuente de un proyecto.

Genera un archivo texto con el nombre *tags* donde almacena las características de un programa, en función del lenguaje, las funciones, las variable, las clases , etc. Ctags soporta muchos lenguajes de programación.

3. Objetivos

3.1. Descripción del problema

A la hora de analizar proyectos, muchas herramientas son requeridas, ya que un proyecto puede ser analizado bajo varios criterios.

SLOCCount es una herramienta que analiza proyectos de software contando líneas de código fuente. Tomando en cuenta este proyecto se vio la necesidad de implementar una herramienta que permita analizar proyectos de software. Tomando en cuenta las funciones/ métodos.

Al analizar un proyecto de software la información que se tenga del proyecto es de gran importancia, cuanto más información se pueda obtener más eficiente serán las decisiones tomadas en criterios como por ejemplo el cosste, el tiempo y la complejidad.

3.2. Objetivo General

Para desarrollar una aplicación que permita obtener la cantidad de funciones de un proyecto cualquiera, se necesita tener acceso al código fuente. Es por ello que esta enfocado más a los proyectos de software libre porque su código fuente es más accesibles y en su gran mayoría podemos obtenerlo de forma gratuita.

3.3. Objetivos Específicos

En esta sección definimos algunos objetivos específicos. Los cuales nos ayudaran a cumplir con nuestro objetivo general, que es, desarrollar una aplicación para contar funciones de un proyecto culaquiera.

- Lanzar las Ctags para un proyecto.
- Analizar la estructura del fichero tags, generado al lanzar las ctags.
- Desarrollar un parser.
- Diseñar y crear una base de datos.
- Analizar la información almacenada en la base de datos.

- Realizar consultas y lanzar resultados.
- Construir un menú de opciones.

3.4. Requisitos

A continuación se detallan los requisitos más básicos que se pretende cumplir, en el desarrollo de este proyecto.

- Facilitar una herramienta para contar funciones.
- Ofrecer una aplicación de fácil manejo.
- Facilitar gráficas para tener mayor comprensión de los resultados.
- imprimir los resultados en diferentes formatos.

3.5. Modelo de desarrollo

En el proceso de desarrollo de un proyecto de software, se realiza una determinada cantidad de tareas, para ello es recomendable utilizar una metodología o modelo de desarrollo, para llevar un orden en la implementación de las tareas o actividades.

El modelo de desarrollo que se utilizó para el desarrollo de esta aplicación es el modelo en espiral.

Se optó por este modelo porque es un modelo de proceso de software evolutivo, es decir, que el proyecto va evolucionando con el pasar del tiempo, donde el esfuerzo del desarrollo es iterativo, tan pronto se termina con un esfuerzo del desarrollo. Es decir, tras un ciclo comienza otro, repitiendo así en forma de espiral hasta concluir con el proyecto. En cada iteración se analiza también los riesgos que se puedan presentar.

Cada ciclo es una vuelta que se divide en cuatro actividades, que se detallan a continuación;

- **Determinar los objetivos**. En esta fase del proyecto se definen los objetivos específicos, se identifican las limitaciones del proceso, se diseña un plan detallado de gestión y se identifican los riesgos.

- **Análisis de riesgo**. En esta fase se analiza detalladamente cada uno de los riesgos que se identifico en la anterior fase. Se definen los pasos que se van a seguir para reducir los riesgos.
- **Desarrollar y verificar**. En esta fase ya se pasa al Desarrollo del software.
- **Planificación**. En esta última fase se revisa el proyecto y se toma la decisión si se continuará con otro ciclo o no. Si se decide continuar, se desarrollaran los planes para la siguiente fase del proyecto.

4. Diseño e implementación

4.1. Arquitectura general

Lo primero que necesitamos para el desarrollo de este proyecto es el código fuente de un proyecto. Para ello se acudió a Internet y se buscó un proyecto de software libre cualquiera. Una vez descargado el proyecto, se comienza a lanzar las Ctags. Al realizar este proceso nos genera de manera automática un archivo tags donde almacena toda la información más relevante del proyecto.

Luego se analizó la estructura del fichero tags junto con los datos. Pasamos al proceso de parseo de los datos que se encuentran en el archivo tags. Una vez parseados los datos que nos interesan, los almacenamos en una base de datos.

A continuación presentamos un diagrama donde se puede apreciar el flujo de procesos de una manera general del proyecto

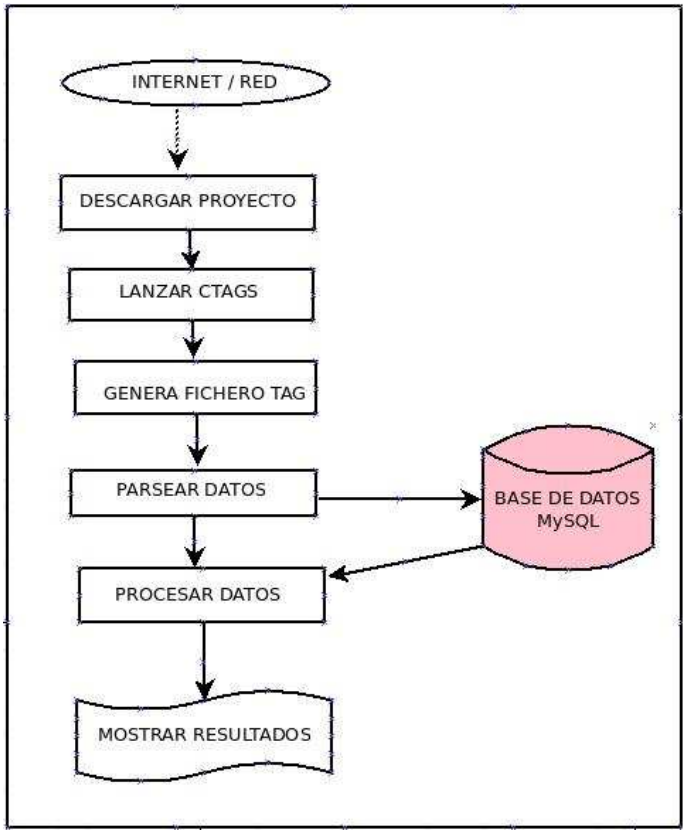


Figura 2: Estructura General del Proyecto

El funcionamiento de este proyecto consiste primero en lanzar ctags para el proyecto que se quiera analizar. Al realizar este primer paso, se genera automáticamente un archivo con el nombre tags. En este archivo se almacena toda la información del proyecto representados en etiquetas. Para continuar con el proyecto necesitamos abrir el fichero *tags*, para luego analizar las etiquetas. Creamos la base de datos con el nombre que el usuario elija al ejecutar el proyecto.

En el proceso de ejecución se crea primeramente la base de datos y como segundo paso se crea las tablas de la base de datos, luego se introduce los datos a las tablas creadas con los datos obtenidos del fichero tags. Este proceso tarda un poco; el tiempo de ejecución depende al volumen de código fuente del proyecto que se quiera analizar; cuanto más grande es el proyecto más tardará el proceso de población de los datos a las tablas.

Luego de crear y poblar las tablas, pasamos a analizar los datos que se almacenan en la base de datos, realizando consultas de diferentes tipos. Finalmente podemos lanzar los resultados al usuario.

La estructura de la base de datos ha sido diseñada en base al análisis detallado que se realizó a los datos almacenados en el archivo tags. Se analizó a detalle las columnas y las etiquetas impresas en cada columna para comprender la estructura del fichero. Una vez analizado el contenido del archivo tags, se buscó las etiquetas de interés para el proyecto, y en base a ello se crearon las tablas de la base de datos.

4.2. Iteración 0: Estudio Previo

En esta iteración se estudió minuciosamente las herramientas que se utilizaron para este proyecto.

4.2.1. Estudio de las Ctags

Lo primero que se estudió son las ctags. Las ctags generan un archivo con etiquetas del código fuente de un proyecto . Para nuestro análisis dividimos el análisis en 3 partes los cuales los detallamos más adelante.

Para ejecutar las ctags hay muchas maneras de hacerlo. Hay muchas opciones que se pueden ocupar, esto sirve para filtran las etiquetas de acuerdo a las necesidades que tengamos, podemos elegir una opción y escribir en una terminal de Linux las líneas de comando de la opciones

elegida y luego ejecutarlas. Esto nos genera un archivo texto plano con el nombre tags.

Para comenzar a estudiar las ctags se utilizó primeramente código fuente con un volumen muy pequeño. Para ello utilizamos solamente código escrito en el lenguaje C++. La opción que se eligió para lanzar las ctags fue *(ctags *.c *.h)*. Con esta opción se pudo ver que las ctags solo toma en cuenta las funciones que están desarrolladas en el fichero .c y no así los que están declaradas en el archivo .h, con esto nos despreocupamos de la duplicidad de funciones en nuestro conteo de funciones.

Luego fuimos aumentando el volumen del código fuente para nuestro análisis. Analizamos un directorio completo el cual tenia varios archivos dentro de diferentes tipos de lenguajes. Como teníamos diferentes lenguajes de programación, la opción que utilizamos para lanzar las ctags fue *(ctags *)*. Con esta opción nos ejecuta todos los archivos sin importar las extensiones del archivo, toma en cuenta todos los archivos que están dentro del directorio en el que nos encontremos.

Finalmente aumentamos el volumen del código fuente para nuestro análisis y ejecutamos todo un proyecto completo con todos sus directorios y archivos que contenga el proyecto. La opción que utilizamos para lanzar las ctags de todo un proyecto fue *(ctags -R *)*, esta opción nos ejecuta todos los archivos de un árbol de directorio completo de manera recursiva sin importar el tipo de archivo, es decir sin importar el tipo de lenguaje del código fuente. Ya que un proyecto puede estar desarrollado en un tipo de lenguaje o en muchos tipos de lenguaje.

A continuación se muestra una parte del contenido del archivo tags que se generó para el proyecto *evince*:

```
!_TAG_FILE_FORMAT 2 /extended format; --format=1 will not append ;" to lines/
!_TAG_FILE_SORTED 1 /0=unsorted, 1=sorted, 2=foldcase/
!_TAG_PROGRAM_AUTHOR Darren Hiebert /dhiebert@users.sourceforge.net/
!_TAG_PROGRAM_NAME Exuberant Ctags //
!_TAG_PROGRAM_URL http://ctags.sourceforge.net /official site/
!_TAG_PROGRAM_VERSION 5.9~svn20110310 //
A85BREAKCNTR backend/tiff/tiff2ps.c 1727;" d file:
A85BREAKLEN backend/tiff/tiff2ps.c 1728;" d file:
ACTION_SCROLL_DOWN libview/ev-view-accessible.c /^ ACTION_SCROLL_DOWN,$/;" e enum:__anon101 file:
ACTION_SCROLL_UP libview/ev-view-accessible.c /^ ACTION_SCROLL_UP,$/;" e enum:__anon101 file:
ACTIVATED shell/ev-history-action.c /^        ACTIVATED,$/;" e enum:__anon159 file:
ACTIVATED shell/ev-zoom-action.c /^        ACTIVATED,$/;" e enum:__anon165 file:
ACTIVATE_LINK libmisc/ev-page-action.c /^ ACTIVATE_LINK,$/;" e enum:__anon100 file:
ACTIVATE_LINK shell/ev-history.c /^        ACTIVATE_LINK,$/;" e enum:__anon177 file:
ADD_BOOKMARK shell/ev-sidebar-bookmarks.c /^        ADD_BOOKMARK,$/;" e enum:__anon145 file:
```

```
AFM2TFM backend/dvi/mdvi-lib/tfmfile.c 71;" d file:
ANGLE backend/dvi/mdvi-lib/sp-epsf.c 50;" d file:
ANNOTS_SIDEBAR_ID shell/ev-window.c 263;" d file:
ANNOT_ACTIVATED shell/ev-sidebar-annotations.c /^ ANNOT_ACTIVATED,$/;" e enum:__anon170 file:
ANNOT_ADD_CANCELLED shell/ev-sidebar-annotations.c /^ ANNOT_ADD_CANCELLED,$/;" e enum:__anon170
file:
APPLICATION_DBUS_INTERFACE shell/ev-application.c 77;" d file:
APPLICATION_DBUS_OBJECT_PATH shell/ev-application.c 76;" d file:
ASCENDER backend/dvi/mdvi-lib/afmparse.c /^  ASCENDER, CHARBBOX, CODE, COMPCHAR, CAPHEIGHT, COMMENT, $/;"
e enum:parseKey file:
ASSERT backend/dvi/mdvi-lib/common.h 184;" d
ASSERT backend/dvi/mdvi-lib/common.h 195;" d
ASSERT_VALUE backend/dvi/mdvi-lib/common.h 188;" d
ASSERT_VALUE backend/dvi/mdvi-lib/common.h 196;" d
ATTACHMENTS_SIDEBAR_ID shell/ev-window.c 261;" d file:
AUTHOR_PROPERTY properties/ev-properties-view.c /^ AUTHOR_PROPERTY,$/;" e enum:__anon140 file:
Ascii85Encode backend/tiff/tiff2ps.c /^Ascii85Encode(unsigned char* raw, char *buf)$/;" f file:
Ascii85EncodeBlock backend/tiff/tiff2ps.c /^int Ascii85EncodeBlock( TIFF2PSContext *ctx, uint8 * ascii85_p,$/;
f
Ascii85Flush backend/tiff/tiff2ps.c /^Ascii85Flush(TIFF2PSContext* ctx)$/;" f
Ascii85Init backend/tiff/tiff2ps.c /^Ascii85Init(TIFF2PSContext *ctx)$/;" f file:
Ascii85Put backend/tiff/tiff2ps.c /^Ascii85Put(TIFF2PSContext *ctx, unsigned char code)$/;" f
AsyncData thumbnailer/evince-thumbnailer.c /^struct AsyncData {$/;" s file:
AutoScrollInfo libview/ev-view-private.h /^} AutoScrollInfo;$/;" t typeref:struct:__anon127
AvoidDeadZonePreamble backend/tiff/tiff2ps.c /^static char AvoidDeadZonePreamble[] = "\\$/;" v file:
```

Los ficheros *tags* tienen un tipo de formato, y están divididos en:

- **tagname**. Nombre de cualquier identificador sin espacios.
- **tagfile**. Nombre del archivo donde el nombre de la etiqueta ”tagname”se define en relación con el directorio actual.
- **tagaddress**. Campo que tiene la descripción para la ubicación de las etiquetas.
- **tagfield**. Es un campo de extensión, donde se almacenan clave y valor separados por tabuladores.

4.2.2. Estructura de base de datos

EL gestor que utilizamos para la base de datos es MySQL. Como el proyecto se desarrollo en Python para la conexión entre el proyecto y el gestor, utilizamos el modulo MySQLdb. Con

este módulo podemos registrar y leer información de la base de datos mediante las funciones que están desarrolladas en *functioncount.py*. A continuación se muestra una gráfica del flujo de conexión de la base de datos.

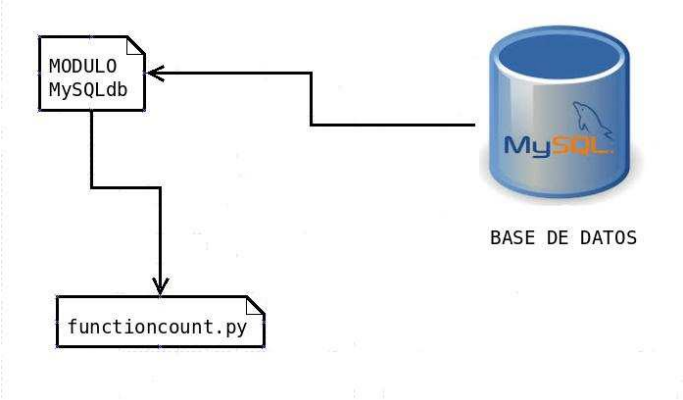


Figura 3: Conexión a la base de datos

Luego de enterarnos de cómo funciona las ctags y de cómo va su estructura, pudimos diseñar la base de datos que esta compuesta por cuatro tablas las cuales se muestran a continuación en la siguiente gráfica:

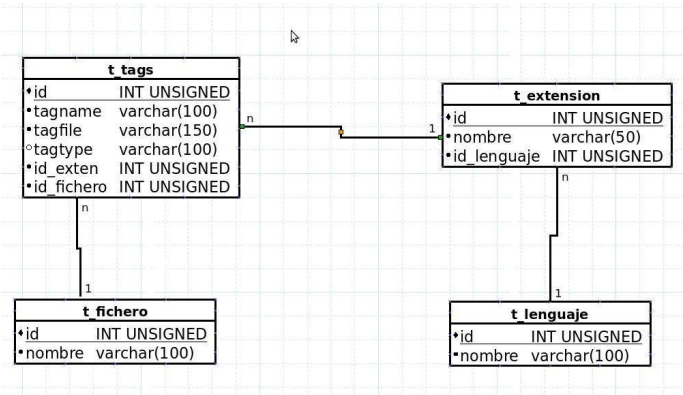


Figura 4: Estructura de la base de datos

4.3. Iteración 1: Desarrollo del modulo parser y almacenamiento de los datos

En esta iteración empezamos a interactuar con nuestra base de datos creando así nuestro parser para almacenar los datos que nos interesan en sus tablas correspondiente.

4.3.1. Diseño

Para el diseño de nuestro parser continuamos analizando las ctags, sus propiedades, las etiquetas que utiliza y sus ventajas. Ctags ofrece muchas opciones para configurar, para ejecutar el código fuente de un proyecto, dependiendo de lo que se quiera analizar puedes utilizar una opción u otras o varias para su ejecución, de acuerdo a los parámetros que se le asignen para ejecutar se genera un archivo llamado tags con la información asociada a los parámetros asignados.

```
--fields=[+|-]flags

a Access (or export) of class members
f File-restricted scoping [enabled]
i Inheritance information
k Kind of tag as a single letter [enabled]
K Kind of tag as full name
l Language of source file containing tag
m Implementation information
n Line number of tag definition
s Scope of tag definition [enabled]
S Signature of routine (e.g. prototype or parameter list)
z Include the "kind:" key in kind field
t Type and name of a variable or typedef as "typeref:"
```

Para este proyecto no se cambió la configuración, se trabajó con la configuración que viene por defecto, pero se puede cambiar de acuerdo a nuestras necesidades.

También hay opciones que se pueden ocupar dependiendo el lenguaje de programación con que el proyecto este desarrollado. Pero en nuestro caso tomaremos en cuenta todos los lenguajes de programación que soporte las ctags. Porque nuestra herramienta analizara proyectos desarrollados en diferentes lenguajes de programación.

A continuación presentamos una lista de etiquetas que utiliza las ctags para identificar las clases, funciones, variables, etc de un lenguaje de programación. Dependiendo del tipo de lenguaje de programación como se muestra a continuación de algunos lenguajes conocidos.

```
   --list-kinds

   ....
C
    c  classes
    d  macro definitions
    e  enumerators (values inside an enumeration)
    f  function definitions
    g  enumeration names
    l  local variables [off]
    m  class, struct, and union members
    n  namespaces
    p  function prototypes [off]
    s  structure names
    t  typedefs
    u  union names
    v  variable definitions
    x  external and forward variable declarations [off]
C++
    c  classes
    d  macro definitions
    e  enumerators (values inside an enumeration)
    f  function definitions
    g  enumeration names
    l  local variables [off]
    m  class, struct, and union members
    n  namespaces
    p  function prototypes [off]
    s  structure names
    t  typedefs
    u  union names
    v  variable definitions
    x  external and forward variable declarations [off]

  ...
Python
    c  classes
    f  functions
    m  class members
    v  variables
    i  imports [off]

  ....
```

4.3.2. Inplementación

Al inicio de la implementación, primeramente creamos la base de datos. Para crear la base de datos, antes de ejecutar el proyecto le pasamos un nombre como parámetro. Este parámetro se refiere al nombre que se le estaría asignado a la base de datos, la cual sera creada en el momento de ejecución del proyecto.

Luego pasamos a conectarnos con la base de datos creada. Luego comenzamos a crear las tablas que formaran parte de esta base de datos. Las tablas son creadas también en el proceso de ejecución del proyecto.

Como las tablas están relacionadas entre sí, para evitar que nos genere error en el momento de su creación, empezamos creando las tablas más independientes y terminamos con la tabla más dependiente de nuestro proyecto, que es la *tabla tags*. Si creamos las tablas sin respetar este criterio nos generaría error.

Una vez terminado el proceso de creación de las tablas, pasamos poblar las tablas con la información correspondiente a cada una.

Para almacenar datos en las *tablas lenguaje y extensión*, implementamos una función para cada tabla, para la cual leemos un archivo de texto plano para cada tabla (*lenguajes.txt* y *extensiónes.txt*). Se desarrolló de esta manera para automatizar el proceso.

Mientras que las *tablas fichero y tags* son pobladas con los datos que salen como resultado del proceso de parseo del fichero tags el cual se genera al ejecutar las ctags.

Las funciones que tenemos implementadas en este proyecto son las siguientes:

- **poblarLenguajes():** Esta función sirve para poblar la tabla lenguaje, lee los datos que están en el fichero lenguajes.txt y los almacena en la tabla de la base de datos lenguaje.
- **poblarExtensiones():** Esta función sirve para poblar de datos la tabla extensión de la base de datos, lo mismo que la otra función lee los datos almacenados en el fichero extensiones.txt y mediante un ciclo va almacenando los datos linia por linia a la tabla extensión.
- **totalFunciones():** Esta función cuenta y saca el total de funciones que el proyecto analizado tiene en su código fuente.
- **funcionesLenguaje():** Este proceso realiza el cálculo de la cantidad de funciones por lenguaje de programación que tiene el proyecto analizado, saca una lista de lenguajes con

las cantidades de funciones correspondientes a cada lenguaje.

- **contarTotalLenguaje(idlen):** Esta función es un proceso auxiliar que cuenta las funciones según el parámetro que se le envíe. El parámetro que se le puede enviar es el identificador de un lenguaje cualquiera. Según este parámetro cuenta la cantidad de funciones que existen desarrolladas del tipo de lenguaje que se le envía como parámetro

- **buscarCodigoExtension(extension):** Este proceso también es una función auxiliar que busca el código de un lenguaje de programación que tenga la extensión que se le envía por parámetros. Un lenguaje de programación puede tener ficheros con diferentes extensiones como por ejemplo .cp o .h, los dos tipos de extensiones pertenecen al lenguaje C++.

- **buscarFichero(fichero):** Este proceso busca el identificador del fichero que se le manda como parámetro.

- **poblarFicheros(ruta):** Esta función sirve para poblar la tabla de la base de datos fichero, se le envía un fragmento de código del archivo tags de la cual especifica la ruta donde se encuentra la línea de código fuente. Parseamos los datos sacando los datos de interés del fichero. Luego se almacena a la tabla fichero de la base de datos.

- **parseoDatos():** Este proceso lee el fichero tags para parsear sus datos. Luego los almacena a la tabla tags de la base de datos

- **contarLenguajeFichero(idlen):** Cuenta la cantidad de funciones que existen de un tipo de lenguaje especifico, el tipo que se le mande como parámetro. Con este id sacamos primeramente todos los tipos de extensiones que puede tener este lenguaje y luego con esta información a través de un ciclo vamos sacando la cantidad parcialmente, luego contamos todas las cantidades parciales, sacando así una cantidad total del tipo de lenguaje.

- **funcionesFichero():** Esta es una función general que engloba las funciones que existen en un fichero de todos los tipos de lenguajes de programación que existan en el fichero.

- **eliminarBaseDatos():** Esta función elimina la base de datos que creamos al inicio del proyecto.

- **imprimirResultados():** Esta función nos ayuda a imprimir los resultados en el formato [.odt, Texto OpenDocument] , documento de texto abierto.

- **imprimirResultadosHtml():** Nos ayuda a imprimir los resultados en el formato [.html, HyperText Markup Language], lenguaje de marcas de hipertexto.
- **imprimirPdf():** Nos ayuda a imprimir los resultados en el formato [.pdf, portable document format], formato de documento portátil.
- **graficaBarras():** Este proceso nos permite realizar una gráfica con los resultados en formato de barras.
- **grafreicaTorta():** De la misma forma que el anterior proceso, nos permite realizar una gráfica con los resultados en formato de tarta.
- **menu():** Esta función nos permite tener acceso a los resultados en sus diferentes formatos, también a mostrar las gráficas en sus dos formatos. A través de este menú el usuario también puede eliminar la base de datos.

4.4. Iteración 2: Desarrollo de un menú en una interfaz gráfica

A medida que se fue desarrollando esta herramienta surgió la idea de construir una interfaz gráfica, donde el usuario pueda tener acceso a un menú de opciones.

4.4.1. Diseño

Para el diseño de nuestro menú, primeramente creamos una interfaz gráfica luego creamos una barra de menú en la cual se contienen tres menús que son:

- **Menú Archivo:** En este menú tenemos dos opciones. La primera opción nos permite eliminar la base de datos creada en el proceso la segunda opción nos permite salir del programa.
- **Menú Resultados:** En este menú mostramos los resultados en tres formatos que son .odt, .html y .pdf.
- **Menú Gráficas:** Aquí presentamos los resultados en forma de gráfica, tenemos dos tipos de gráficas que son: en barras y tartas.

A continuación presentamos una gráfica que muestra nuestro menú en la interfaz gráfica:

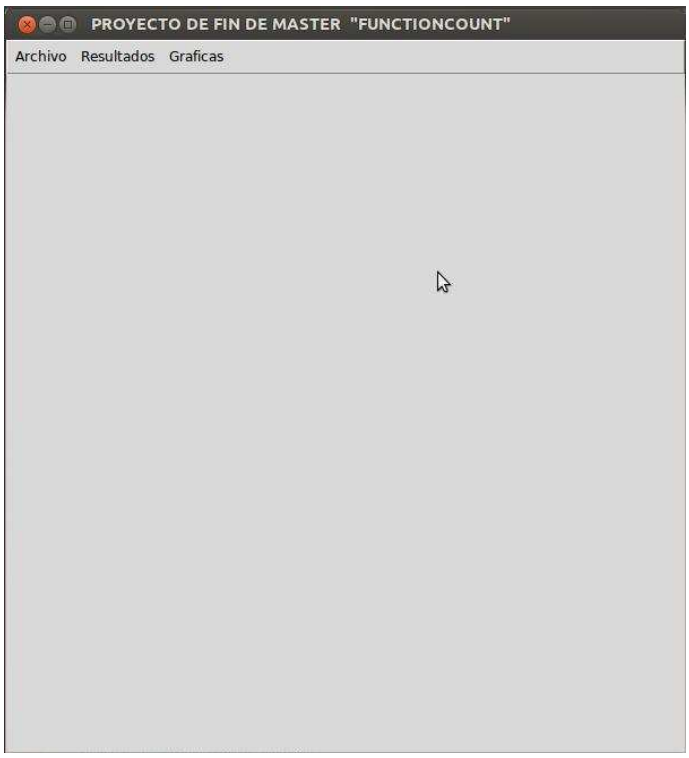


Figura 5: Interfaz gráfica que contiene un menú de opciones

4.5. Iteración 3: Presentación de Resultados Obtenidos

Después del proceso de creación de la base de datos, de sus tablas, construcción del parser y almacenamiento la información obtenida con las ctags a la base de datos, pasamos analizar los datos que se guardaron en la base de datos. Realizamos diferentes consultas a la base de datos, los cuales nos presentan resultados con diferentes criterios. Esta herramienta presenta los resultados en diferentes formatos:

4.5.1. Presentación de resultados en formato de texto del proyecto Python

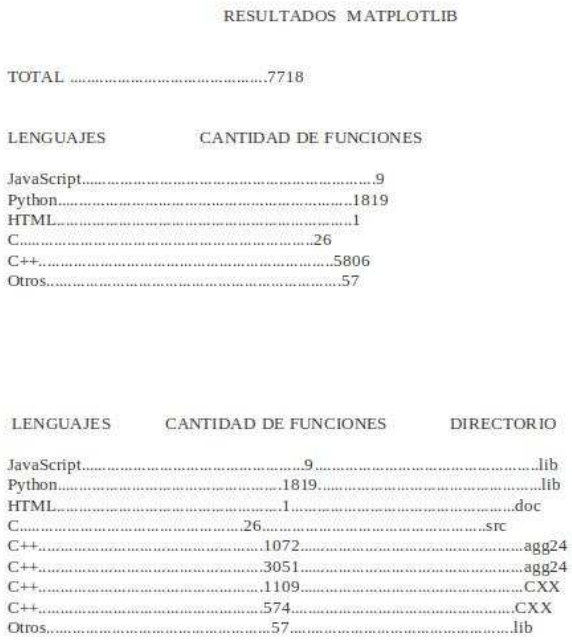
La primera opción determinada para la salida de información de la base de datos, fue imprimir los resultados en la la terminal donde la herramienta seria ejecutada. Esta opción resultaría más fácil para el usuario para ver los resultados, sin tener que contener ninguna interfaz gráfica. Además también fue la manera más rápida de ver los resultados de la herramienta. A continuación presentamos los resultados obtenidos del análisis del proyecto *Python*:

```
cris12@cris:~$ python functioncount.py python

================================RESULTADOS============================


TOTAL .........................21915


LENGUAJES        CANTIDAD DE FUNCIONES
----------------------------------------
JavaScript ..................... 1
Python ..................... 8297
HTML ..................... 2
C ..................... 12782
C++ ..................... 255
Sh ..................... 104
Otros ..................... 474


 LENGUAJES         CANTIDAD DE FUNCIONES          DIRECTORIO
-----------------------------------------------------------------
JavaScript .....................1 .....................Doc
Python .......................8297 ..................Demo
HTML ....................... ..2 .....................Lib
C ......................... 12782 ....................Demo
C++ .........................255 ......................Python
Sh ..........................104 ......................Tools
Otros .......................474 ..................... Mac
```

```
 cris12@cris:~$ python functioncount.py evince


================================RESULTADOS============================


TOTAL ........................2793


LENGUAJES        CANTIDAD DE FUNCIONES
---------------------------------------
Python ..................... 2
C ..................... 2680
C++ ..................... 110
Sh ..................... 1


 LENGUAJES         CANTIDAD DE FUNCIONES          DIRECTORIO
-------------------------------------------------------------
Python ...................... 2 ..................... test
C ...................... 2680 ..................... backend
C++ ...................... 3 ..................... libdocument
C++ ...................... 107 ..................... backend
Sh ...................... 1 ..................... cut-n-paste
```

4.5.2. Presentación de resultados en formato .odt del proyecto Matplotlib

La segunda opción para mostrar los resultados es mediante una interfaz gráfica que desarrollamos la cual contiene un menú de opciones. El usuario puede tener acceso al resultado a través de una opción en el menú. Los resultados son guardados en un fichero cuyo nombre es *resultado.odt* a través de la función *imprimirResultados()*. A continuación mostramos los resultados en este formato del proyecto *Matplotlib*.



RESULTADOS MATPLOTLIB

TOTAL 7718

LENGUAJES	CANTIDAD DE FUNCIONES
JavaScript	9
Python	1819
HTML	1
C	26
C++	5806
Otros	57

LENGUAJES	CANTIDAD DE FUNCIONES	DIRECTORIO
JavaScript	9	lib
Python	1819	lib
HTML	1	doc
C	26	src
C++	1072	agg24
C++	3051	agg24
C++	1109	CXX
C++	574	CXX
Otros	57	lib

Figura 6: Gráfica: Resultados en formato .odt

4.5.3. Presentación de resultados en formato .html del proyecto Linux-3.4.4

Otra de las opciones que nos presenta en el menú de opciones es la salida de resultados en formato HTML. Para la salida de estos resultados preparamos la información a través de la función *imprimirResultadosHtml()*. A continuación presentamos a través de una captura de pantalla, los resultados en formato HTML del proyecto *Linux*.

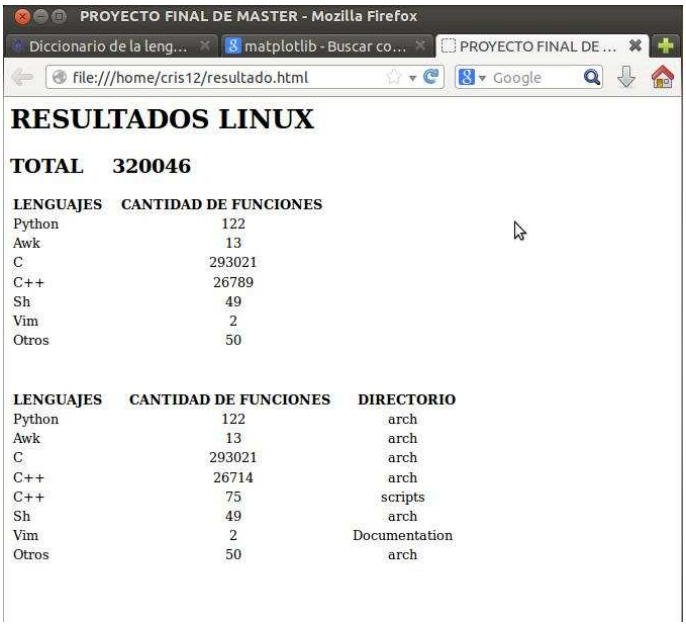
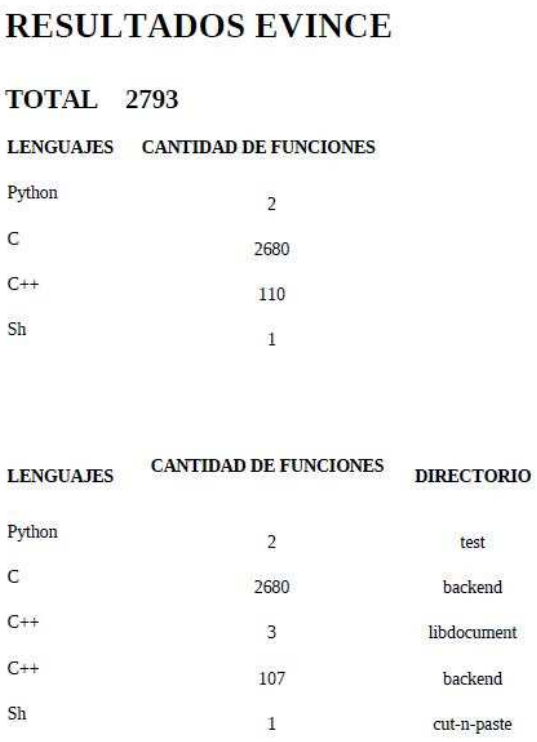


Figura 7: Gráfica: Resultados en formato .html

4.5.4. Presentación de resultados en formato .pdf del proyecto Evince

En nuestro menú también nos ofrece la opción de mostrar los resultados en formato PDF. Para generar esta opción lo que hacemos es una comversión del formato HTML a PDF en la función *imprimirPdf*, en la cual ocupamos el comando *unoconv* para dicha conversión. A continuación presentamos los resultados en formato PDF del proyecto *Evince*



RESULTADOS EVINCE

TOTAL 2793

LENGUAJES	CANTIDAD DE FUNCIONES
Python	2
C	2680
C++	110
Sh	1

LENGUAJES	CANTIDAD DE FUNCIONES	DIRECTORIO
Python	2	test
C	2680	backend
C++	3	libdocument
C++	107	backend
Sh	1	cut-n-paste

Figura 8: Gráfica: Resultados en formato .pdf

4.5.5. Presentación de gráficas

Para presentar las gráficas utilizamos la librería *matplotlib*, con esta librería y con los datos que presenta nuestra herramienta, pudimos crear dos tipos de gráficas que son: barras y torta. Para mostrar estas gráficas en nuestro menú de opciones que contiene nuestra interfaz gráfica, nos ofrece dos opciones que son: *barras y torta*. Para ver uno de de estos tipos el usuario únicamente tiene que darle un clic en una de las opciones. A continuación presentaremos algunas gráficas.

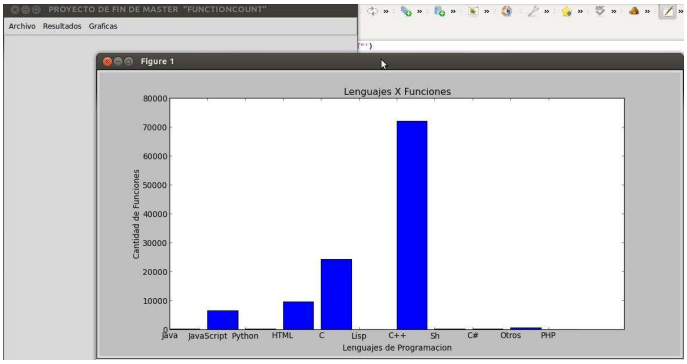


Figura 9: Gráfica en barras del proyecto Mozilla

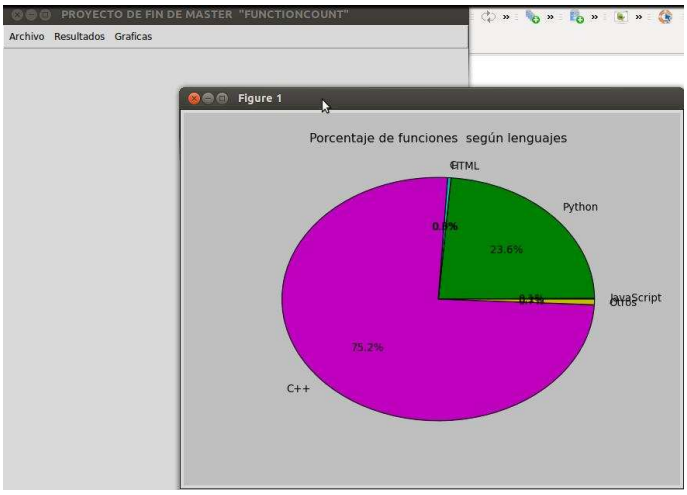


Figura 10: Gráfica en forma de torta del proyecto Matplotlib

En la gráfica de torta, podemos ver que algunos nombres de los lenguajes de programación se sobreponen uno sobre otro, esto es porque los porcentajes de esos resultados son muy pequeños, y es por eso que los trozos de torta para estos tipos de lenguajes son pequeños, casi insignificantes, pero la librería Matplotlib nos permite realizar un zoom a los trozos que queramos verlos con más detalle. A continuación mostramos una gráfica que muestra el trozo de tarta donde los nombres se sobreponen y de esta manera podemos verlos mejor.

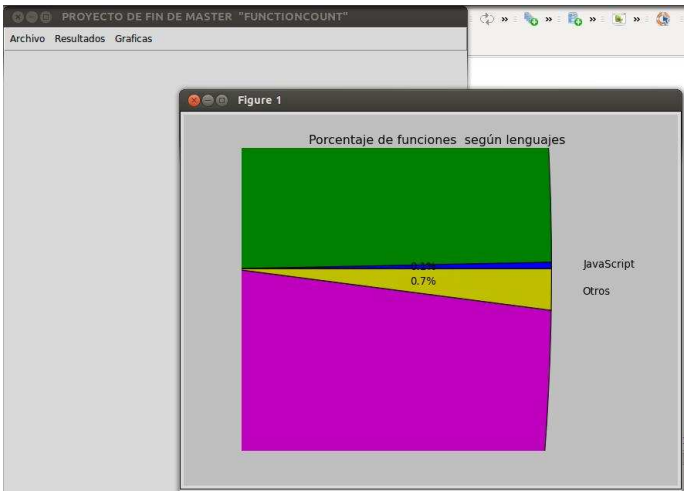


Figura 11: Gráfica de un trozo de la tarta del proyecto Matplotlib

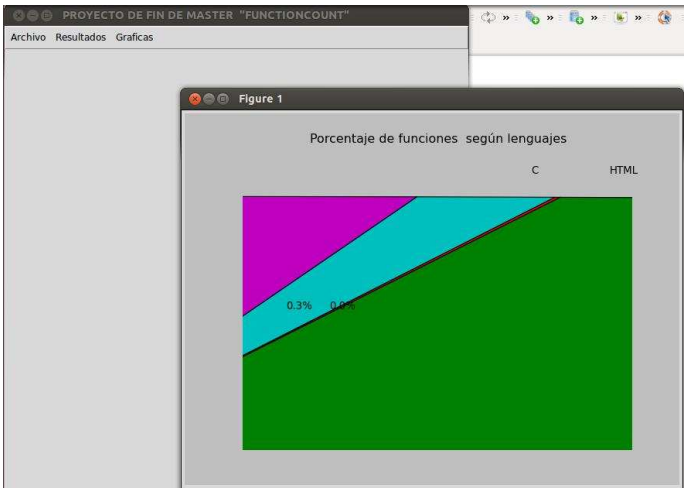


Figura 12: Gráfica de un segundo trozo de la tarta del proyecto Matplotlib

4.6. Análisis de los resultados

4.6.1. Análisis del proyecto linux-3.4.4

Para el análisis de este proyecto utilizamos la herramienta que desarrollamos *functioncount* y procesamos el código fuente del proyecto Linux. Los resultados que obtuvimos los pasamos a una tabla. Luego pasamos a ejecutar SLOCCount para este mismo proyecto, de la misma manera los resultados obtenidos lo pasamos a nuestra tabla la cual mostraremos a continuación.

PROYECTO LINUX				
	LINEAS DE CODIGO		FUNCIONES	
LENGUAJES	CANTIDAD	PORCENTAJE	CANTIDAD	PORCENTAJE
Python	2812	0.03%	122	0.04%
Awk	710	0.01%	13	0.00%
C	9922723	96.83%	293021	91.55%
C++	4761	0.05%	26789	8.37%
Sh	4575	0.04%	49	0.02%
Vim	0	0.00%	2	0.00%
Otros	2552855	3.04%	50	0.02%
TOTAL	10247436		320046	

Figura 13: Tabla de resultados del proyecto Linux

En esta tabla tenemos cinco columnas, la primera columna muestra la lista de lenguajes de programación en la que el proyecto Linux está implementado. La segunda columna se refiere a la cantidad de líneas de código fuente que tiene el proyecto Linux por cada lenguaje de programación. La tercera columna muestra los porcentajes correspondientes a cada lenguaje de programación según el cien por ciento que seria el total de líneas de código fuente. Estos datos los obtenemos a través de la herramienta SLOCCount. En la cuarta fila mostramos la cantidad de funciones que tiene el proyecto Linux por cada lenguaje de programación. Y la última línea, que es la quinta línea, mostramos los porcentajes correspondientes a cada cantidad de funciones por cada lenguaje de programación. Los datos de la cuarta columnas los obtenemos a través de la herramienta functioncount. Y los datos de la última columna realizamos el cálculo de forma manual utilizando los datos que nos proporciono la herramienta functioncount.

Con estos datos podemos analizar y ver el comportamiento de las líneas vs funciones para ello presentamos una gráfica en la que podemos visualizar con mayor detalle este comportamiento.

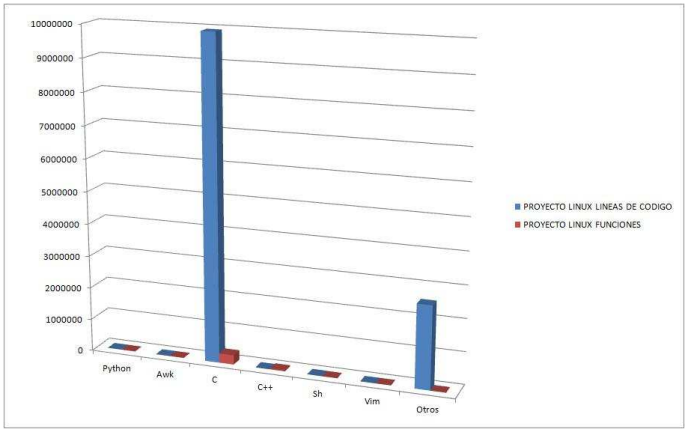


Figura 14: Gráfica en barras de líneas y funciones del proyecto linux-3.4.4

Si tomamos en cuenta las cantidades para nuestro análisis, podemos visualizar que hay

mayor cantidad de líneas, la diferencia entre las líneas y funciones, en función de la cantidad es muy grande. Este comportamiento es normal ya que una función engloba muchas cantidades de líneas.

Es por ello que presentamos esta otra gráfica donde ya no se toma en cuenta las cantidades, si no mas bien los porcentajes. Aquí ya podemos ver que la diferencia no es mucha. Podemos ver que en función a los porcentajes los resultados se asemejan mucho. Si tomamos como ejemplo el pico de esta gráfica, que lo representa el lenguaje de programación C, y los porcentajes son 96,88 de líneas de código y 91,55 de funciones. Podemos observar que hay una diferencia de un 5,33 por ciento. Esta diferencia es mínima. A continuación mostramos la gráfica que nos muestra esta diferencia.

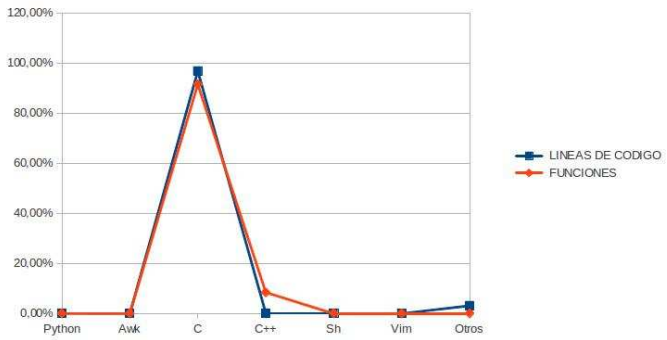


Figura 15: Gráfica en líneas proyecto Linux-3.4.4

4.6.2. Análisis del proyecto Mozilla Firefox 3.0.4

PROYECTO MOZILLA				
	LINEAS DE CODIGO		FUNCIONES	
LENGUAJES	CANTIDAD	PORCENTAJE	CANTIDAD	PORCENTAJE
Java	5189	0.20%	190	0.17%
JavaScript	0	0.00%	6582	5.78%
Python	6692	0.26%	199	0.17%
HTML	0	0.00%	9531	8.36%
C	944435	36.32%	24335	21.36%
Lisp	256	0.01%	15	0.01%
C++	1469601	56.52%	72040	63.23%
Sh	28678	1.10%	236	0.20%
C#	6232	0.24%	277	0.24%
PHP	244	0.01%	7	0.01%
Otros	59968	2.30%	525	0.46%
xml	85122	3.27%	0	0.0%
TOTAL	2600185		113937	

Figura 16: Tabla de resultados del proyecto Mozilla Firefox 3.0.4

En la tabla que contiene los resultados del proyecto Mozilla podemos ver que la cantidad total de líneas de código es de 2,600,185 y 113,937 de funciones la diferencia en este proyecto tomando en cuenta las cantidades también son muy grandes, teniendo una mayor cantidad de líneas de código pero si tomamos en cuenta los porcentajes la diferencia no es tan grande. A continuación presentamos una gráfica que nos muestra la diferencia en porcentajes de las líneas de código vs funciones.

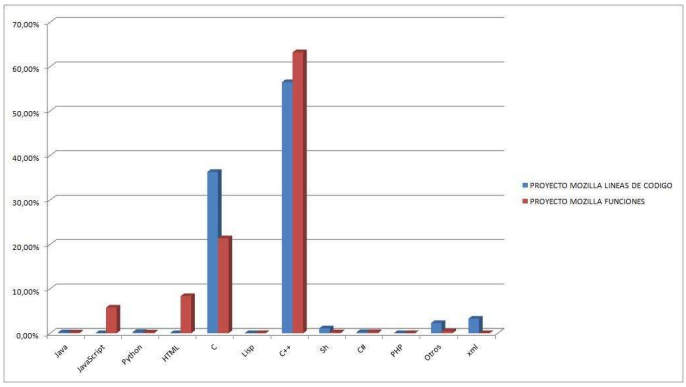


Figura 17: Gráfica en líneas proyecto Mozilla Firefox 3.0.4

En esta gráfica podemos ver dos barras que superan el 10 por ciento, en una de ellas que es la de lenguaje C el mayor porcentaje son las líneas de código mientras que en la otra barra es del lenguaje C++ y el mayor porcentaje en esta barra lo lleva las funciones. Existen dos barras mas pequeñas que nos llamo la atención. En estas dos barrras pequeñas tenemos un porcentaje considerablemente de funciones y no así de líneas de código a continuación detallaremos mas estos lenguajes de código.

- **JavaScript**. En este lenguaje tenemos 5,78 por ciento de funciones y 0 por ciento de líneas de código. Este resultado nos alarmó al principio porque es algo incoherente. Como voy a tener una porcentaje de funciones y no así ningún por ciento de líneas de código, siendo que una función esta echa a base de líneas de código. Pero insertándonos en el paquete de este proyecto pudimos comprobar que efectivamente existen código escrito en JavaScript. Y que además existen mucha funciones desarrolladas con este tipo de lenguaje.

- **HTML**. Es un lenguaje de marcado y su código se escribe en forma de etiquetas. Por lo que podemos decir que no podemos construir funciones con este tipo de lenguaje. Pero el porcentaje que tenemos de funciones en este tipo de lenguaje es de 8,36 por ciento y

de líneas de código un 0 por ciento. Esto es otro caso curioso. Pero de la misma forma que actuamos con el anterior caso, con JavaScript actuamos en este. Nos introducimos en los directorios de este proyecto y analizamos los directorios y buscamos archivos .html y pudimos comprobar que efectivamente este tipo de lenguaje existían y que dentro de del código de estos archivos existen funciones de tipo JavaScript las que toma en cuenta nuestra herramienta para el conteo de funciones. Y es por ello que efectivamente tenemos funciones en este tipo de ficheros.

- **C**. En este lenguaje de programación tenemos 36,32 por ciento de líneas de código y 21,36 por ciento de funciones. Teniendo mas líneas de código que funciones, pero la diferencia es de un 15 por ciento. Podemos entender de que existen funciones implementadas en este lenguaje con muchas líneas de código.

- **C++**. Este lenguaje es el que mas por ciento de líneas de código tiene y por ende el que mas funciones tiene. Tiene un 56,52 por ciento de líneas de código y un 63,23 por ciento de funciones. Con lo que podemos decir que hay funciones con pocas líneas de código en este tipo de lenguaje

4.6.3. Análisis del proyecto Python

En la gráfica siguientes mostramos los resultados obtenidos del proyecto Python al pasar por las herramientas de functioncount y SLOCCount. mostrando las cantidades por lenguaje de programación con sus porcentajes correspondientes a cada cantidad en función a las cantidades totales.

PROYECTO PYTHON				
	LINEAS DE CODIGO		FUNCIONES	
LENGUAJES	CANTIDAD	PORCENTAJE	CANTIDAD	PORCENTAJE
C	431422	48.32%	12782	58.32%
C++	172	0.02%	255	1.16%
Python	430622	48.23%	8297	37,86%
Sh	15629	1.75%	104	0.47%
JavaScript	0	0.00%	1	0.00%
HTML	0	0.00%	2	0.01%
Otros	14922	1.67%	474	2.16%
TOTAL	892767		21915	

Figura 18: Tabla de resultados del proyecto Python

Con esta información podemos ver que el proyecto Python esta desarrollado en seis tipos de lenguajes de programación diferentes, esto sin contar los lenguajes que no son tan conocidos a

los cuales les asignamos el nombre de otros. En la gráfica siguiente podemos ver dos picos que sobresalen, son los lenguajes de programación que mayor porcentaje tienen de este proyecto. El primer pico corresponde al lenguaje C, marcando así con un 48,32 por ciento de líneas de código y un 58,32 por ciento de funciones. Con lo que podemos ver que en este tipo de lenguaje de programación hay un mayor porcentaje de funciones. Por lo que podemos decir también que las funciones de este tipo de lenguaje tienen menor cantidad de línas de código.

El otro pico corresponde al lenguaje de programación de Python que marca un porcentaje 48,23 de líneas de código y un 37,86 por ciento de funciones. Con lo que podemos decir que las funciones de este tipo de lenguaje contienen mayor cantidad de líneas de código fuente.

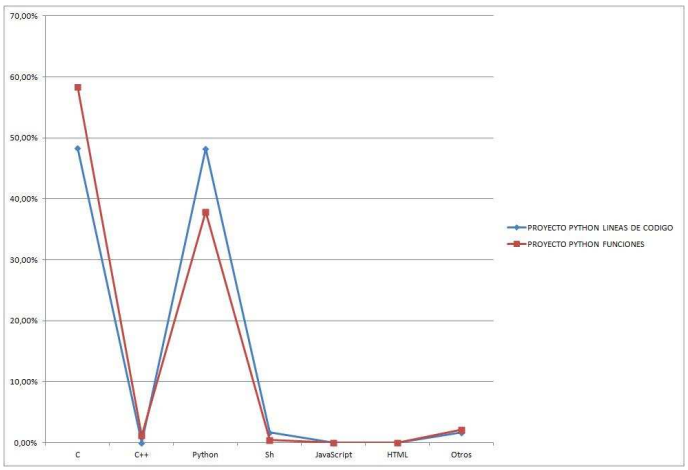


Figura 19: Gráfica en líneas proyecto Python

4.6.4. Análisis del proyecto evince

PROYECTO EVINCE				
	LINEAS DE CODIGO		FUNCIONES	
LENGUAJES	CANTIDAD	PORCENTAJE	CANTIDAD	PORCENTAJE
C	65312	94.69%	2680	95.95%
C++	2793	4.05%	110	3.93%
Python	207	0.30%	2	0.07%
Sh	59	0.09%	1	0.03%
xml	604	0.88%	0	0.0%
TOTAL	68965		2793	

Figura 20: Tabla de resultados del proyecto evince

Los resultados del análisis de este proyecto reflejan que hay un equilibrio entre funciones y líneas de código, si tomamos en cuenta sus porcentajes.

Pero si tomamos en cuenta las cantidades desde luego que el que lleva la delantera son las líneas de código. Si observamos la gráfica que presentamos a continuación podemos ver que

las barras casi van juntas. Hay un caso que nos llama la atención y es lo que sucede con el lenguaje de marcado XML ya que existen 604 líneas de código fuente tomando en cuenta las cantidades y 0 de funciones. Esto es porque XML es un lenguaje de marcas que funciona a través de etiquetas, y dentro de su código no podemos encontrar una función implementada. Y es por ello que en los archivos .xml siempre encontraremos una cantidad de líneas de código fuente pero no así de funciones.

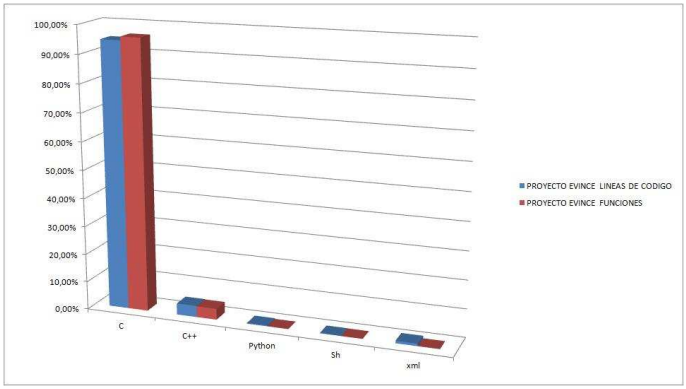


Figura 21: Gráfica en líneas proyecto evince

4.6.5. Tiempo de ejecución

En esta sección trataremos sobre el tiempo de ejecución de nuestra herramienta. A continuación mostramos una gráfica donde mostramos los cuatro proyectos que analizamos. En este caso la gráfica lo realizamos tomando en cuenta el total de las líneas de código y el total de las funciones. Estos datos los obtuvimos de los resultados mostrados anteriormente.

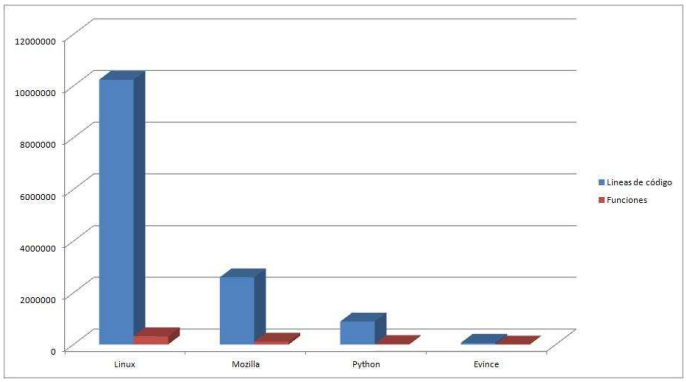


Figura 22: Gráfica del total de los proyectos

En esta gráfica podemos ver que el proyecto con mayor volumen de código es el proyecto de

Linux. El que le sigue es Mozilla y el siguiente a este es Python y al último esta evince. Evince podemos decir que es el proyecto mas pequeños en comparación de los cuatro proyectos. Para el análisis del tiempo tomamos como ejemplo los dos proyectos mas pequeños, para no que esperar tanto el proceso, ya que con el proyecto Linux se tarda mucho. El tiempo de ejecución que tarda al ejecutar Python es de 260 segundos.

- **Tiempo de ejecución del proyecto Python** . Este tiempo es de *260 segundos*. Es el tiempo que tarda el proyecto en ejecutarse con la herramienta functioncount, con la herramienta SLOCCount tarda *21 segundos.*

- **Tiempo de ejecución del proyecto evince** . Este proyecto tarda *36 segundos* en ejecutarse con la herramienta functioncount. Mientras que con SLOCCount tarda *3 segundos.*

Comparando el tiempo que tarda nuestra herramienta en ejecutar estos proyectos con el tiempo que tarda ejecutando SLOCCount. Podemos ver que nuestra herramienta demora mucho en ejecutarse, ya que los tiempos de SLOCCount son mas pequeños. Tendríamos que optimizar más el tiempo de ejecución esto queda como una tarea pendiente.

4.7. Conclusiones

El desarrollo de este proyecto tuvo sus complicaciones ya que las herramientas que utilizamos para el desarrollo de este proyecto en su mayoría eran nuevas, teniendo primero que enterarnos de su funcionamiento. Pero podemos concluir que se término satisfactoriamente. Cumpliendo con los objetivos propuestos al principio de este proyecto.

- Realizamos un estudió sobre las ctags, lanzando así para cualquier proyecto.
- Analizamos el fichero generado por las ctags, entendiendo su estructura.
- Desarrollamos un parser, para la obtención de los datos de interés del fichero tags.
- Diseñamos y creamos una base de datos, que contiene los datos parseados.
- Analizamos los datos almacenada en la base de datos, para realizar las consultas correspondientes al caso.
- imprimimos resultados.
- Construimos un menú de opciones para que el usuario pueda tener acceso a los diferentes formatos de resultados que imprime nuestra aplicación, para que tenga acceso también a las gráficas y tener la opción de eliminar la base de datos.

Podemos decir que concluimos con todos nuestros objetivos específicos propuestos al inicio de este proyecto. De esta manera concluimos también con nuestro objetos general que fue:

Desarrollar una herramienta que cuente las funciones de un proyecto cualquiera. Ayudandonos a Presentar resultados totales, parciales según los directorios y los tipos de lenguaje en que el proyecto fue desarrollado.

Este proyecto presenta los resultados en formato de texto, ODT, HTML y PDF y además nos muestra gráficas para tener una mayor comprensión de los resultados. Podemos destacar que para el desarrollo de este proyecto, se utilizo solamente herramientas de Software Libre por lo que se pudo comprobar la eficiencia de las herramientas.

4.8. Lecciones aprendidas

El proceso de desarrollo de este proyecto fue enriquecedor, porque me permitió aprender muchas herramientas nuevas de las cuales no tenía conocimientos y adquirir más experiencia con algunas herramientas de las que ya tenía conocimientos básicos.

- Entender lo beneficioso que es tener acceso al código fuente de un proyecto, porque si no fuera así no se podría desarrollar este proyecto. Puedo concluir que las libertades que nos ofrece el software libre son muy beneficiosas.
- El uso de las ctags que para mí esto fue nuevo, no tenía conocimiento de ellas. Pero ahora puedo decir que sí tengo y que son de gran utilidad.
- Desarrollar con el lenguaje Python, fue algo novedoso para mí, conociendo anteriormente solamente de nombre. Pudiendo comprobar así las funcionalidades de este lenguaje y lo legible que es el código en este lenguaje, se puede incorporar muchas librerías, las que necesitemos.
- Se aprendió a utilizar la librería Matplotlib, la cual presenta muchas funciones que nos permiten realizar gráfica en distintos formatos.
- Se aprendió a utilizar *MySQL* a través de la terminal. Y para la conexión el manejo de la librería *MySQLdb*.
- Para la documentación se aprendió el manejo de la LaTeX . El uso de este sistema de procesamiento de documentos fue novedoso también, comprender su sintaxis fue a principio complicado. Pero al final se logró realizar esta memoria.

4.9. Trabajos futuros

Una vez finalizado este proyecto, podemos plantear algunos campos de trabajo que supondrían aportar para la extensión de su funcionalidad y contribuirían a consolidarlo como una herramienta eficaz.

El defecto de esta herramienta está en el tiempo de ejecución de los proyectos ya que si ejecutamos proyectos que tengan un volumen muy grande de código fuente, esto se torna incómodo para el usuario, porque tendría que esperar demasiado tiempo. Esto sería una tarea que dejamos pendiente para mejorar

Otro tarea pendiente seria incorporar a la herramienta el proceso de ejecución de las ctags, ya que actualmente para que nuestra herramienta funcione primeramente tenemos que ejecutar las ctags, para que nos genere el fichero tags, para el proyecto que queramos analizar. Ahora mismo tenemos que tener el fichero tags ya generado para que nuestra herramienta funcione de lo contrario no funcionaria. De esta manera nuestra herramienta se tornaría mas independiente.

Todos estos nuevos objetivos (y los que pueda plantear una tercera persona), podrán implementarse para potenciar la herramienta, que quedará amparada por una licencia de software libre.

A. Instalación y uso

Para poner en marcha nuestra aplicación, antes tenemos que tener generado el fichero tags del proyecto que queramos analizar. Para obtener este fichero, primeramente nos introducimos a través de línea de comandos en el directorio que engloba el proyecto que queremos analizar, luego ejecutamos la siguiente instrucción:

```
cris12@cris:~/evince$ ctags -R *
```

Luego una vez obtenido el fichero tags, ponemos en marcha este proyecto para ello ejecuta el archivo functioncount.py acompañado por un nombre, recomendamos que el nombre sea el del proyecto que se esta analizando, ya que es para crear la base de datos con este nombre, como se muestra a continuación.

```
cris12@cris:~$ python functioncount.py Linux;
```

Donde Linux es el nombre a la base de datos que se creara para el proyecto que se analizara, ese nombre el usuario lo elige.

A.1. Requisitos

Los requisitos previos a la puesta en marcha de las herramientas son:

- *Intérprete de Python*.
  Debería funcionar con la versión 2.7.3 o posterior. Durante el desarrollo se uso la versión 2.7.3

- *Sistema gestor de base de datos MySQL*.
  Debería funcionar con la versión 5.5.31 en adelante. Durante el desarrollo se uso la versión 5.5.31.

- *Módulo MySQLdb para Python*.
  Durante el desarrollo se uso la versión 1.2.1 Escoger la versión de este módulo compatible con las del intérprete Python y MySQL empleados.

- *Herramientas Ctags*.

- *Matplotlib*.
  Herramientas que sirve para realizar las gráficas.

Bibliografía

[1] Web del proyecto MySQL for Python.

`http://dev.mysql.com/downloads/connector/python/`

[2] Web wiki Software libre.

`http://es.wikipedia.org/wiki/Software_libre`

[3] Ingenieria de Software libre.

`http://www.crice.org/files/Ingenieria-de-Software-Libre-y-Herramientas-Aplicadas.pdf`

[4] Web wiki MySQL.

`http://es.wikipedia.org/wiki/MySQL`

[5] Web del proyecto SLOCCount.

`http://www.dwheeler.com/sloccount/`

[6] Web slideshare.

`http://www.slideshare.net/byronoleas/herramientas-de-software-libre-aplicados-a-la-`

[7] Web plagatux ctags.

`http://plagatux.es/2009/02/generar-etiquetas-con-ctags-y-usarlas-en-vim/`

[8] Web oficial matplotlib.

`http://matplotlib.org/`

[9] Web oficial python.

`https://www.python.org/`