INGENIERÍA INFORMÁTICA

Curso Académico 2006/2007

Proyecto Fin de Carrera

MUCODE: Código más utilizado en aplicaciones de software libre

Autor : Luis Cañas Díaz
Tutor : Jesús M. González Barahona

(c) 2007 Luis Cañas Díaz
This work is licensed under the
Creative Commons Attribution-ShareAlike 2.5 License.
To view a copy of this license, visit
http://creativecommons.org/licenses/by-sa/2.5/
or send a letter to
Creative Commons,
543 Howard Street, 5th Floor, San Francisco,
California, 94105, USA.
Ver apéndice G para más detalles.

A Juan y Adela.

Agradecimientos

Este proyecto no habría sido posible sin la colaboración y paciencia de mis compañeros del grupo de investigación GSyC/LibreSoft, especialmente de Juanjo que se presentó como conejillo de indias y de Carlos que tuvo paciencia para explicarme cosas del mundo GNOME.

Además me gustaría agradecer al resto de usuarios del proyecto que no siendo parte directamente relacionada con el grupo se animaron a ser partícipes del estudio.

Por último agradecerles a Gregorio y Jesús sus correcciones y consejos.

Índice general

1. Resumen **1**

2. Introducción **3**
2.1. Motivación . . . 3
2.2. Realización del estudio . . . 4
2.3. Resto del documento . . . 5

3. Entorno Tecnológico **7**
3.1. Software libre . . . 7
3.1.1. Principios básicos . . . 7
3.1.2. Estudio del desarrollo del software libre . . . 8
3.1.3. Herramientas de minería de software . . . 9
3.1.4. CVSAnalY, análisis de repositorios software . . . 9
3.2. Cooperative Bug Isolation project . . . 10
3.2.1. Un poco de historia . . . 10
3.2.2. Razones para su utilización . . . 11
3.3. Debian . . . 12
3.3.1. El sistema de paquetes en Debian GNU/Linux . . . 13
3.3.2. Partes y creación de paquetes Debian . . . 14
3.3.3. Ramas de desarrollos en Debian GNU/Linux . . . 15
3.3.4. La aparición de Ubuntu . . . 16
3.4. GNOME . . . 16
3.5. Python . . . 17

4. Objetivos **19**
4.1. Comprender el comportamiento de un programa . . . 19
4.2. Realizar el estudio en un entorno real . . . 20
4.2.1. Identificar partes críticas en aplicaciones GNOME . . . 20
4.2.2. Relacionar partes críticas con historia del código fuente . . . 20
4.2.3. Gráficas que relacionen informes con commits y medidas objetivas . . . 21
4.3. Aumentar el conocimiento sobre las herramientas de CBI . . . 21
4.4. Estudio de viabilidad sobre instrumentar distribuciones completas . . . 21

5. Requisitos **23**
5.1. Introducción . . . 23
5.1.1. Propósito . . . 23

5.1.2. Alcance . . . 23
5.1.3. Definiciones, acrónimos y abreviaturas . . . 23
5.1.4. Referencias . . . 24
5.1.5. Resumen del resto del documento . . . 24
5.2. Descripción general . . . 24
5.2.1. Perspectiva del producto . . . 24
5.2.2. Interfaces del sistema . . . 25
5.2.3. Funciones del producto . . . 25
5.2.4. Características del usuario . . . 25
5.2.5. Restricciones . . . 25
5.3. Requisitos específicos . . . 25
5.3.1. Interfaz de usuario . . . 25
5.3.2. Interfaz Hardware . . . 25
5.3.3. Fuentes de entrada . . . 26
5.3.4. Destino de la salida . . . 26
5.3.5. Rangos válidos . . . 26
5.3.6. Restricciones temporales . . . 26
5.3.7. Relaciones con otras entradas/salidas . . . 26
5.4. Requisitos funcionales . . . 26
5.4.1. Exportar los informes de uso a una base de datos . . . 26
5.4.2. Obtener el listado de funciones más utilizadas . . . 27
5.4.3. Obtener el listado de los ficheros más utilizados y sus desarrolladores . . . 28
5.4.4. Obtener el listado de los ficheros más utilizados y sus desarrolladores del último año . . . 29
5.5. Número máximo de usuarios . . . 30
5.6. Restricciones de diseño . . . 30

6. Diseño e Implementación 31
6.1. Prototipos de aplicación instrumentada . . . 31
6.1.1. Aplicación instrumentada . . . 32
6.1.2. Aplicación empaquetada en Debian . . . 35
6.2. Selección de aplicaciones objetivo para el estudio . . . 37
6.2.1. EOG . . . 39
6.2.2. Evince . . . 40
6.2.3. Evolution . . . 41
6.2.4. Gaim . . . 42
6.2.5. GNOME-terminal . . . 44
6.2.6. Rhythmbox . . . 45
6.3. Clases utilizadas por los scripts de análisis . . . 45
6.3.1. Clase report_store . . . 46
6.3.2. Clase Analyzer . . . 46
6.4. Funciones de los scripts de análisis . . . 47
6.4.1. Exportar los informes de uso a una base de datos . . . 47
6.4.2. Obtener el listado de funciones más utilizadas . . . 47
6.4.3. Obtener el listado de los ficheros más utilizados y sus desarrolladores . . . 48

6.4.4. Obtener el listado de los ficheros más utilizados y sus desarrolladores del último año . . . 49
6.5. Bibliotecas utilizadas por los scripts de análisis . . . 49
6.5.1. MySQLdb . . . 49
6.5.2. Aplicaciones de la biblioteca DOM . . . 50

7. Estudio de los informes **51**
7.1. Entorno real del estudio . . . 51
7.2. Tipo de estudio realizado . . . 52
7.3. Resumen de los datos ofrecidos por el estudio . . . 52
7.4. Resultados sobre Eye of Gnome 2.18.1 . . . 53
7.4.1. Datos obtenidos . . . 53
7.4.2. Funciones más utilizadas . . . 53
7.4.3. Actividad alrededor de los ficheros más utilizados en año previo a la publicación . . . 54
7.4.4. Actividad alrededor de los ficheros más utilizados . . . 55
7.5. Resultados sobre Evince 0.4.0 . . . 57
7.5.1. Datos obtenidos . . . 57
7.5.2. Funciones más utilizadas . . . 57
7.5.3. Actividad alrededor de los ficheros más utilizados en el último año . . . 57
7.6. Resultados sobre Evince 0.8.1 . . . 60
7.6.1. Datos obtenidos . . . 60
7.6.2. Funciones más utilizadas . . . 60
7.6.3. Actividad alrededor de los ficheros más utilizados en el último año . . . 62
7.6.4. Actividad alrededor de los ficheros más utilizados . . . 62
7.7. Resultados sobre Evolution 2.6.3 . . . 65
7.7.1. Datos obtenidos . . . 65
7.7.2. Funciones más utilizadas . . . 65
7.7.3. Actividad alrededor de los ficheros más utilizados en el último año . . . 65
7.8. Resultados sobre Evolution 2.10.1 . . . 67
7.8.1. Datos obtenidos . . . 67
7.8.2. Funciones más utilizadas . . . 67
7.8.3. Actividad alrededor de los ficheros más utilizados en el último año . . . 67
7.8.4. Actividad alrededor de los ficheros más utilizados . . . 69
7.9. Resultados sobre Gedit 2.14.4 . . . 72
7.9.1. Datos obtenidos . . . 72
7.9.2. Funciones más utilizadas . . . 72
7.9.3. Actividad alrededor de los ficheros más utilizados en el último año . . . 73
7.10. Resultados sobre Gedit 2.18.1 . . . 73
7.10.1. Datos obtenidos . . . 73
7.10.2. Funciones más utilizadas . . . 73
7.10.3. Actividad alrededor de los ficheros más utilizados en el último año . . . 73
7.10.4. Actividad alrededor de los ficheros más utilizados . . . 75
7.11. Resultados sobre Gnome Terminal 2.14.2 . . . 78
7.11.1. Funciones más utilizadas . . . 78

7.11.2. Actividad alrededor de los ficheros más utilizados en el último año 79
7.11.3. Actividad alrededor de los ficheros más utilizados 79
7.12. Resultados sobre Rhythmbox 0.9.6 83
7.12.1. Datos obtenidos 83
7.12.2. Funciones más utilizadas 83
7.12.3. Actividad alrededor de los ficheros más utilizados en el último año 83
7.13. Resultados sobre Rhythmbox 0.10.0 85
7.13.1. Datos obtenidos 85
7.13.2. Funciones más utilizadas 85
7.13.3. Actividad alrededor de los ficheros más utilizados en el último año 85
7.13.4. Actividad alrededor de los ficheros más utilizados 87
7.14. Estimación de Error 87
7.15. Aparición de la biblioteca libegg 89
7.16. Conclusiones extraídas del estudio de informes 90

8. Conclusiones y trabajos futuros 93

A. Proceso de instrumentación con sampler-cc 97

B. Modificación de Makefiles de Debian 101

C. Informes de uso 105

D. Medidas calculadas para las funciones 107

E. Medidas calculadas para los ficheros 109

F. Ejemplo de uso 111

G. Licencia Reconocimiento-CompartirIgual 2.5 España 113

Índice de figuras

3.1. Ciclo de vida de los paquetes Debian. 14

6.1. Modelo de desarrollo en espiral 31
6.2. SLOCCount aplicado a GNOME-terminal-2.14.2 32
6.3. Modificación aplicada al fichero control 36
6.4. Modificación aplicada al fichero changelog 37
6.5. Modificación aplicada al fichero rules 38
6.6. Clase report_store 46
6.7. Clase analyzer 47
6.8. Base de datos de los informes de uso. 48

7.1. Componentes del proyecto MUCode 52
7.2. Actividad en el año previo a la publicación de EOG 2.18.1 55
7.3. Actividad hasta la publicación de EOG 2.18.1 57
7.4. Actividad en el año previo a la publicación de Evince 0.4.0 58
7.5. Actividad en el año previo a la publicación de Evince 0.8.1 62
7.6. Actividad hasta la publicación de Evince 0.8.1 64
7.7. Actividad en el año previo a la publicación de Evolution 2.6.3 66
7.8. Actividad en el año previo a la publicación de Evolution 2.10.1 69
7.9. Actividad hasta la publicación de Evolution 2.10.1 72
7.10. Actividad en el año previo a la publicación de Gedit 2.14.4 74
7.11. Actividad en el año previo a la publicación de Gedit 2.18.1 76
7.12. Gráfica de uso contra actividad sobre Gedit 2.18.1 78
7.13. Actividad en el año previo a la publicación de GNOME Terminal 2.14.2 81
7.14. Gráfica de uso contra actividad sobre GNOME Terminal 2.14.2 82
7.15. Actividad en el año previo a la publicación de Rhythmbox 0.9.6 84
7.16. Actividad en el año previo a la publicación de Rhythmbox 0.10.0 86
7.17. Actividad hasta la publicación de Rhythmbox 0.10.0 87

B.1. Modificación aplicada al fichero debhelper.mk 101
B.2. Modificación aplicada al fichero GNOME.mk 102
B.3. Error en la variable de entorno CFLAGS 103

C.1. El fichero environment 105
C.2. Ejemplo de sección del informe de uso dedicado a un fichero y sus funciones . . . 106
C.3. Ejemplo de información sobre las unidades de compilación 106

Índice de cuadros

3.1. Los diez lenguajes más utilizados en Debian 4.0 12

7.1. Datos recolectados para EOG 2.18.1 53
7.2. 10 funciones más utilizadas en EOG 2.18.1 53
7.3. Actividad en el año previo a la publicación de EOG 2.18.1 54
7.4. Actividad hasta la publicación de EOG 2.18.1 56
7.5. Datos recolectados para Evince 0.4.0 57
7.6. 10 funciones más utilizadas en Evince 0.4.0 58
7.7. Actividad en el año previo a la publicación de Evince 0.4.0 59
7.8. Datos recolectados para Evince 0.8.1 60
7.9. 10 funciones más utilizadas para Evince 0.8.1 60
7.10. Actividad en el año previo a la publicación de Evince 0.8.1 61
7.11. Actividad hasta la publicación de Evince 0.8.1 63
7.12. Datos recolectados para Evolution 2.6.3 65
7.13. 10 funciones más utilizadas para Evolution 2.6.3 65
7.14. Actividad en el año previo a la publicación de Evolution 2.6.3 66
7.15. Datos recolectados para Evolution 2.10.1 67
7.16. 10 funciones más utilizadas para Evolution 2.10.1 67
7.17. Actividad en el año previo a la publicación de Evolution 2.10.1 68
7.18. Actividad hasta la publicación de Evolution 2.10.1 71
7.19. Datos recolectados para Gedit 2.14.4 72
7.20. 10 funciones más utilizadas en Gedit 2.14.4 73
7.21. Actividad en el año previo a la publicación de Gedit 2.14.4 74
7.22. Datos recolectados para Gedit 2.18.1 75
7.23. 10 funciones más utilizadas en Gedit 2.18.1 75
7.24. Actividad en el año previo a la publicación de Gedit 2.18.1 76
7.25. Actividad hasta la publicación de Gedit 2.18.1 77
7.26. Datos recolectados para GNOME Terminal 2.14.2 78
7.27. 10 funciones más utilizadas para GNOME Terminal 2.14.2 79
7.29. Actividad hasta la publicación de GNOME Terminal 2.14.2 80
7.28. Actividad en el año previo a la publicación de GNOME Terminal 2.14.2 81
7.30. Datos recolectados para Rhythmbox 0.9.6 83
7.31. 10 funciones más utilizadas para Rhythmbox 0.9.6 83
7.32. Actividad en el año previo a la publicación de Rhythmbox 0.9.6 84
7.33. Datos recolectados para Rhythmbox 0.10.0 85
7.34. 10 funciones más utilizadas para Rhythmbox 0.10.0 85

7.35. Actividad en el año previo a la publicación de Rhythmbox 0.10.0 86
7.36. Actividad hasta la publicación de Rhythmbox 0.10.0 88
7.37. Estimación de error provocado sobre el estudio de la historia completa 89

D.1. Medidas calculadas sobre el número de llamadas a funciones del estudio 107

E.1. Medidas calculadas sobre el número de llamadas a funciones acumuladas por los ficheros del estudio 109
E.2. Medidas calculadas sobre el número de commits en el año previo a la publicación 109
E.3. Medidas calculadas sobre el número de commits 110

Capítulo 1

Resumen

Este proyecto fin de carrera aporta un nuevo método para el estudio del desarrollo de aplicaciones de software libre. El método hace uso de aplicaciones modificadas para la determinación de las partes más utilizadas del código fuente y su posterior relación con la historia del código. Con el objetivo de estudiar la viabilidad de aplicar este método a distribuciones completas de software, se aplicó el método a un conjunto de aplicaciones del escritorio GNOME.

La primera de las etapas del proyecto tuvo como objetivo la familiarización con las herramientas de análisis, concretamente con las ofrecidas por el proyecto *Cooperative Bug Isolation*, así como determinar si éstas reunían los requisitos adecuados para el resto del proyecto. Una vez elegida la aplicación de análisis del software, se procedió mediante su utilización a construir prototipos de aplicaciones instrumentadas que emitieran informes sobre su utilización al terminar. Tras adquirir el conocimiento necesario sobre la herramienta, se buscó un conjunto de aplicaciones que sirvieran de ejemplo para el objetivo propuesto y se dispuso a montar la infraestructura necesaria para el estudio. En cuanto a la infraestructura fue necesario tener un sitio donde publicitar las herramientas modificadas, un servidor que recogiera los informes de uso enviados por los usuarios finales y un repositorio que sirviera a los usuarios las herramientas para una fácil instalación. Dado que el grupo de usuarios que iban a participar en el estudio usaban distribuciones basadas en Debian (con entorno de escritorio GNOME), se procedió a empaquetar las herramientas modificadas y a colgarlas de un repositorio de aplicaciones Debian creado para la ocasión.

Tras unos meses en los que se continuó aumentando el número de aplicaciones instrumentadas, se llegaron a recolectar cerca de seis mil informes de uso de siete aplicaciones en dos distribuciones GNU/Linux distintas. Estos informes fueron analizados con unas herramientas desarrolladas para la ocasión, que además relacionaban las partes críticas de cada aplicación con la historia del código fuente de cada una de ellas. El resultado del estudio de los informes ofrece:

- las funciones más utilizadas de una aplicación
- los diez ficheros más utilizados según sus funciones y sus datos cuantitativos sobre commits y committers durante diferentes intervalos de tiempo
- medidas objetivas como media, mediana, desviación estándar y coeficiente de variación para todos los datos obtenidos

Capítulo 2

Introducción

2.1. Motivación

En un nicho del desarrollo de aplicaciones software en el que el acceso al código fuente es casi una tarjeta de presentación, cualquier persona con ánimo de aprender puede leer el código como si se tratara de una bonita historia. En el ámbito del software libre, esa persona no sólo puede conocer el estado actual del libro, sino que además mediante herramientas de minería de repositorios software, llegar a conocer la manera en la que un grupo de desarrolladores fueron hilando el código actual. Pese a que el estudio de la historia del software puede proporcionar una gran cantidad de datos de interés, carece de un punto de vista sustancialmente importante, el del usuario final. Hasta ahora no ha habido manera de saber qué parte del código es más importante para el usuario o qué parte está lejos de ser ampliamente utilizada. Es ese vacío el que se busca llenar utilizando información sobre el uso de las herramientas, obtenida gracias a usuarios finales que utilizan aplicaciones compiladas especialmente con ese fin.

Con el objetivo de conocer como se utiliza un grupo de aplicaciones del entorno del software libre, será necesario en primer lugar aplicar a éstas herramientas de análisis de funcionamiento. Dentro de la ingeniería del software, se entiende por análisis de funcionamiento a la investigación que tiene lugar alrededor del comportamiento de un programa y que se obtiene en tiempo de ejecución. Las herramientas claves en este análisis, son aquellas que ofrecen un perfil sobre el comportamiento de determinadas partes del programa; estos datos posibilitarán mejoras tales como optimizar la velocidad o el uso de la memoria.

A diferencia del estudio que se realiza del código antes de ser implementado, un analizador investiga el comportamiento de la aplicación usando la información que le proporciona el programa en tiempo de ejecución. Típicamente, los analizadores han sido utilizados para evaluar el comportamiento de programas en nuevas arquitecturas o bien para identificar qué partes del código ocupan durante más tiempo la CPU y qué funciones son llamadas en esa ejecución. Identificar las partes que consumen más CPU puede mostrar al desarrollador qué partes del programa son más lentas de lo esperado, lo que ofrece una interesante lista de porciones de código candidatas a ser mejoradas. También se utilizan frecuentemente para observar qué funciones son llamadas y si lo son más o menos veces de lo esperado, lo que ayuda a localizar errores de una manera mucho más sencilla.

La primera aplicación bajo Unix destinada a obtener el perfil de un programa en ejecución data del año 1979, cuando los sistemas Unix introdujeron una herramienta básica llamada *prof*

que mostraba las funciones del programa así como el tiempo de ejecución que usaban. Años después, en 1982, *gprof* extendió el concepto hacia el grafo de llamadas, en el que se representaban gráficamente las relaciones existentes entre llamadas a funciones dentro de un código.

En 1994, Amitabh Srivastava y Alan Eustace de la Digital Equipment Corporation publicaron un artículo en el que describían *ATOM(Analysis Tools with OM)* [SE94], un conjunto de herramientas que lograban que un programa se convierta en su propio analizador. Esto se conseguía en tiempo de compilación, ya que se insertaba código dentro del programa a ser analizado. Este código insertado daba como resultado datos del análisis del programa. Esta técnica mediante la cual un programa se analiza sí mismo y que será utilizada en el desarrollo de este proyecto, se conoce como *instrumentación*.

En el año 2004, el artículo sobre *ATOM* fue nombrado uno de los veinte más influyentes de la historia del PLDI o *Programming Language Design and Implementation*, que es la conferencia más importante de la SIGPLAN, sección enfocada en los lenguajes de programación de la ACM.

2.2. Realización del estudio

La naturaleza de este proyecto fin de carrera hace que abarque diferentes áreas de trabajo, todas ellas utilizando herramientas de software libre. La primera etapa se ocupará de aplicar una herramienta de análisis de funcionamiento a un grupo de aplicaciones; la herramienta además deberá encargarse de gestionar y entregar los informes sobre ese funcionamiento. Las aplicaciones generadas por la herramienta de análisis serán ofrecidas a un grupo de voluntarios utilizando un sistema de paquetes que automatice la instalación. Finalmente, el esfuerzo se centrará en estudiar los informes obtenidos y en compararlos con diferentes medidas para la actividad de un proyecto de software libre.

En lineas generales las tareas realizadas han sido:

- obtener el conocimiento sobre una herramienta compleja utilizada en el análisis de código fuente como *sampler-cc* (del proyecto CBI)
- instrumentar aplicaciones utilizando *sampler-cc*
- probar las aplicaciones instrumentadas para asegurar un rendimiento óptimo
- empaquetar las herramientas modificadas en formato Debian, para asegurar una fácil instalación de las aplicaciones
- crear repositorios con las aplicaciones modificadas para facilitar su instalación
- publicitar el estudio en un entorno real de usuarios
- instalar la infraestructura necesaria para la recolección de informes
- realizar scripts de análisis de los informes y de la historia del código fuente
- estudiar las partes del código más utilizado de cada aplicación
- estudiar la historia de cada aplicación
- calcular medidas estadísticas para los datos obtenidos

2.3. Resto del documento

El resto de la documentación ofrecida para el proyecto fin de carrera tratará primero de ofrecer una visión introductoria de la tecnología utilizada, así como de mostrar los objetivos. A continuación se detallará el ciclo de vida y se comentarán con medidas objetivas los informes analizados para las aplicaciones. Finalmente y basándose en el trabajo realizado y los informes obtenidos, se incluye la sección de conclusiones y trabajos futuros.

En la parte final de la memoria se adjunta documentación sobre secciones que pese a ser imprescindibles durante el proyecto, no encajaban en la documentación por semántica o estilo del mismo.

Capítulo 3

Entorno Tecnológico

A lo largo de este capítulo se pretende introducir con más detalle algunas de las tecnologías que a lo largo de la vida del proyecto se han decidido utilizar, no adentrándose demasiado en detalles técnicos y justificando su inclusión en el proyecto.

La primera sección enumerará las características básicas de los desarrollos efectuados en el software libre y los objetivos de su estudio. La sección 3.2 se dedicará a poner en contexto la creación de la herramienta de instrumentación utilizada y a resumir las ventajas que aporta al proyecto. A continuación, la sección 3.3 ofrecerá un vistazo general sobre las políticas de la distribución GNU/Linux que fue utilizada para la distribución de las aplicaciones instrumentadas. La sección 3.4 pondrá en contexto al lector sobre el proyecto GNOME, del que han sido elegidas las aplicaciones a instrumentar que servirán de ejemplo a trabajos futuros. Finalmente, la sección 3.5 resumirá las características que han hecho de Python el lenguaje utilizado para la creación de los scripts que realizan el estudio sobre los datos obtenidos.

3.1. Software libre

3.1.1. Principios básicos

Según la Free Software Foundation, software libre se refiere a la libertad de los usuarios para ejecutar, copiar, distribuir, estudiar, cambiar y mejorar el software. De modo más preciso, se refiere a cuatro libertades de los usuarios del software:

- **Libertad 0**: La libertad de usar el programa, para cualquier propósito.
- **Libertad 1**: La libertad de estudiar cómo funciona el programa, y adaptarlo a las necesidades de los usuarios.
- **Libertad 2**: La libertad de distribuir copias.
- **Libertad 3**: La libertad de mejorar el programa y hacer públicas las mejoras a los demás, de modo que toda la comunidad se beneficie.

Aunque no se menciona explícitamente, para que puedan cumplirse las libertades 1 y 3 es requisito previo el acceso al código fuente [Sta99].

Otra definición de software libre podría ser la siguiente:

> *Software libre es un término acuñado por Richard Stallman y la Free Software Fundation para referirse al software que puede ser usado, estudiado y modificado sin restricción, y que puede ser copiado y redistribuido ya sea modificado o no sin restricción, o con los requisitos necesarios para asegurar que los receptores tendrán las mismas libertades.* [wik01]

3.1.2. Estudio del desarrollo del software libre

Este estudio se basa en la enorme cantidad de información que ofrece el software libre. Se ha descubierto que en la mayoría de los desarrollos de software libre se emplean una serie de características que difieren del desarrollo *tradicional* y que hacen su estudio ciertamente interesante:

- Prototipado rápido, muchas y muy frecuentes publicaciones [Ray98].
- Evolución rápida en los desarrollos. Esto implica un alto crecimiento del tamaño del código fuente, llegando a ser superlineal en algunos casos [RAGBH05]. Desarrollos que antes sólo podían ser llevados a cabo por grupos de desarrolladores remunerados han sido producidos al margen del mundo empresarial.
- Factor humano/auto-selección: Desarrolladores motivados trabajando en tareas que les gustan. La integración de voluntarios en el equipo de desarrollo y la posibilidad de participación de los usuarios del software forman parte de esta característica.
- Comunicación fluida por varios canales, generalmente de manera abierta y a través de Internet. Listas de correo, clientes de canales de comunicación como IRC, sistemas de gestión de errores y sistemas de control de versiones.
- Jerarquía basada en méritos (meritocracia), en la que hay un pequeño núcleo de desarrolladores respetados y muchos pequeños participantes [CH05].
- Licencias que permiten el acceso y modificación de código.
- Conocimiento basado en el código fuente.
- Reutilización de código y componentes de otros proyectos de software libre.
- Rápida detección y solución de los defectos del programa.
- Resolución de problemas mediante la "inteligencia colectiva" (auto-organización) [RMGB05].
- Alta flexibilidad. Por ejemplo, eliminación de la burocracia propia de los proyectos de software tradicional [Sca04].

No todos los proyectos de software libre tienen todas estas características, si bien algunas de ellas pueden ser muy interesantes por la posible aplicación de estas a la industria del software tradicional.

En contraposición a las técnicas utilizadas por el software libre, es necesario también enumerar los problemas que existen en lo que se refiere a su estudio:

- Es algo relativamente novedoso, lo que hace difícil prever su evolución.
- Una vez se tienen los datos, es difícil sacar conclusiones.
- Imposibilidad de definir cuál es el método correcto para el desarrollo de un proyecto. Es decir, la franja que delimita un proyecto exitoso de uno que vaya a fracasar es cuanto menos difícil de trazar ya que no sigue un patrón fijo.

3.1.3. Herramientas de minería de software

Antes de comenzar a hablar de las herramientas de MSR [1]en el ámbito del software libre es necesaria una definición de repositorios de software. Los repositorios de software son aplicaciones informáticas que implementan un control de versiones, éste mantiene el registro de las modificaciones realizadas sobre los ficheros que componen un proyecto y permite que distintos desarrolladores colaboren.

En el software libre, la disposición de la información proporcionada por los repositorios hace que aumente la investigación mediante herramientas de *minería de repositorios software* o *MSR*. El objetivo de éstas es descubrir la forma en la que extraer información de esos repositorios puede ayudar a entender del desarrollo software, apoyar o rebatir predicciones sobre el crecimiento y a planear aspectos futuros de los proyectos.

Existen dos tendencias en el estudio de minería de repositorios software. La primera está orientada en ayudar a la productividad del desarrollador de software libre, mientras que la segunda está más orientada a aprovechar el conocimiento extraído para mejorar la gestión de los proyectos dentro de este ámbito.

3.1.4. CVSAnalY, análisis de repositorios software

CVSAnalY es una herramienta de MSR originalmente desarrollada por Gregorio Robles para el grupo de investigación GSyC/LibreSoft [GSy05]. Su inclusión en el proyecto, además de por su funcionalidad básica, está motivada por el alto grado de conocimiento que hay sobre su funcionamiento en el grupo.

La metodología de CVSAnalY está basada en el análisis del log[2] del sistema control de versiones (ya sea CVS o SVN). En CVSAnalY cualquier interacción[3] que efectúa un committer[4] con el servidor del sistema de control de versiones es guardado con los siguientes datos asociados: nombre del committer, fecha, fichero, número de revisión, líneas añadidas, líneas eliminadas y el comentario que el committer introdujo en el momento del commit.

Básicamente CVSAnalY consta de tres etapas: preprocesado, inserción en la base de datos y postprocesado.

La primera etapa analiza los logs que el sistema de control de versiones alberga para un determinado proyecto. Mientras se realiza el análisis y antes de realizar la inserción del código SQL generado, ocurren algunas cosas como que cada fichero es marcado según sea su tipo. Esto último nos permite diferenciar entre diferentes grupos de contribuidores, ya que pueden

[1] *MSR* es la abreviación de Mining Software Repositories

[2] En la informática es común referirse a los registros cronológicamente almacenados como *logs*

[3] Las interacciones con los repositorios de código se denominan *commits*

[4] Un *committer* es una persona que tiene acceso de escritura en un respositorio y hace un commit en un tiempo dado.

existir tipos de fichero como imágenes, traducciones, documentación, código fuente, interfaces de usuario, código fuente y ficheros de sonido.

Una vez que los logs han sido analizados y transformados en un formato estructurado, se procede a insertar esos datos previa optimización en una base de datos.

El última etapa, la del postprocesado se compone de varios scripts que interactúan con la base de datos, analizan estadísticamente su información y generan diferentes gráficas basándose en la evolución en el tiempo de parámetros interesantes como número de commits, committers y líneas de código.

3.2. Cooperative Bug Isolation project

Cooperative Bug Isolation project o el proyecto de aislamiento de errores cooperativo, surge de la necesidad de mejorar los sistemas de notificación de error automática para llegar a identificar errores automáticamente. Su autor es Ben Liblit.

3.2.1. Un poco de historia

Antes de la llegada del acceso masivo a internet, la mayoría de los errores que ocurrían en aplicaciones software morían en el olvido, ya que los desarrolladores carecían de retroalimentación por parte de los usuarios finales. Con la llegada del acceso masivo a la red de un alto porcentaje de los usuarios, se comenzó a utilizar software para facilitar y automatizar la notificación de una ejecución errónea. De esa manera, cuando una aplicación termina de manera incorrecta se recolecta información que puede ser de ayuda a los desarrolladores y se envía a éstos (frecuentemente a un servidor dedicado para éste fin). El grupo de desarrolladores puede tener acceso de esa manera al contenido de los registros, al stack trace[5] o incluso a una descripción por parte del usuario de como reproducir el error.

A medida que fueron aumentando las aplicaciones que incluían software para la notificación de errores automática, se fueron haciendo más populares los sistemas de gestión de errores o *Bug Tracking Systems* cuya misión es ayudar a los desarrolladores a clasificar y consultar los informes de error. Su funcionalidad se apoya en una base de datos que almacena los eventos que ocurren alrededor de un bug[6] o error. Estos eventos incluyen fechas, estados y comentarios. Por ejemplo, es posible saber cuando fue notificado, cuál es la importancia que le da el usuario, cuál es el comportamiento anómalo o cómo es posible reproducir el error. En entornos corporativos estos sistemas de gestión son frecuentemente utilizados para medir la productividad de sus programadores arreglando errores de código, aunque el número de bugs cerrados no es una media objetiva ya que pueden variar mucho en lo que se refiere a importancia y complejidad.

Para desgracia de los desarrolladores, todo este nuevo flujo de información creaba un nuevo problema, la tarea de filtrar y ordenar todas las notificaciones recaía sobre ellos. Dado que los gestores no pueden identificar notificaciones duplicadas de un mismo error, esta tarea recae sobre el grupo de desarrollo y aunque algo tan sencillo como marcar una notificación de fallo como duplicada parece no requerir demasiado tiempo, la realidad es bien distinta. A principios de Junio del 2007, el número de notificaciones de error abiertas para el proyecto Firefox de

[5]Se conoce por *stack trace* a un listado de los elementos de la pila en un instante dado. Aunque puede ser ejecutado en cualquier instante, se suele utilizar para mostrar donde se produce un error.

[6]Se utiliza *bug* para denominar a un error de software

Mozilla ascendía a cuatro mil y el número de notificaciones marcadas como duplicadas ascendía a diecinueve mil. Para la misma fecha en el total de los proyectos de Mozilla el número de notificaciones de error creadas era de cuarenta mil mientras que el de duplicadas de ciento diez mil.

La hipótesis en la que Liblit se basó es que las notificaciones convencionales de error van en el camino adecuado, pero que no profundizan en la solución. La idea de Liblit es aprovechar al máximo que los usuarios y los desarrolladores están conectados a la red. Uno de los errores que ve Liblit en las notificaciones convencionales es que no recogen información alguna de lo que pasó antes del error. De igual manera ve imprescindible recolectar información sobre las ejecuciones que acabaron correctamente, lo que le proporcionaría una herramienta para diferenciar entre los comportamientos anómalos (extraídos de las ejecuciones erróneas) de los comportamientos inocuos comunes a todas las ejecuciones.

La idea básica que sustenta el trabajo en el proyecto CBI es que puesto que los errores más importantes son aquellos que pasan con más frecuencia a la mayoría de los usuarios, no es necesario trazar el comportamiento completo de un programa. En lugar de eso Liblit utiliza una *instrumentación ligera* para muestrear una pequeña cantidad de información sobre cada ejecución y de esa manera acumular información para crear una perspectiva aproximada de cómo el software está actuando.

Para aplicar esta idea en el mundo real era necesario cubrir ciertos requisitos. Primero la aplicación instrumentada no debía diferir en rendimiento de la aplicación normal, al menos, que no fuera perceptible para el usuario. Segundo, las aplicaciones instrumentadas debían emitir un informe y este a su vez ser enviado a un servidor central en el que éstos se recopilarían.

Ahondar en las soluciones que utilizó Liblit se escapa de este capítulo (consultar apéndice A), si bien es interesante mencionar que su utilización para evaluar cuál es el código más utilizado de una aplicación no fue uno de sus objetivos. Actualmente el proyecto ofrece un conjunto de herramientas para el escritorio *GNOME* que el usuario final puede descargar y utilizar para colaborar con el aislamiento automático de errores.

3.2.2. Razones para su utilización

Dado el grupo de investigación en el que se ha desarrollado el proyecto, uno de los requisitos básicos que debía cumplir el analizador es que se pudiera aplicar a proyectos grandes en el mundo del software libre. Uno de los primeros objetivos era evaluar si sería posible, a través de informes de uso de las aplicaciones, conocer el comportamiento de distribuciones completas de GNU/Linux como Debian, por lo que el analizador debería al menos soportar el lenguaje C. De hecho si observamos una distribución como Debian GNU/Linux en la tabla 3.1 [Amo07], podremos observar que la mayoría del código está escrito en lenguaje C.

Existen una serie de aplicaciones libres[7] que permiten analizar código C, algunas de ellas son *gprof*, *Valgrind*, *OProfile*, *Sysprof* y *sampler-cc* (esta última del proyecto CBI).

La elección de *sampler-cc* para el proyecto se basa en dos ventajas:

- mínimo impacto en el rendimiento
- dependencias en las aplicaciones instrumentadas mínimas

[7] El término *libre* se refiere a software cuya licencia permita, una vez obtenido, ser usado, copiado, estudiado, modificado y redistribuido libremente

Lenguaje	SLOC	*%SLOC*
ansic	145372177	51.369
cpp	52992617	18.726
sh	29309783	10.357
java	8969572	3.17
perl	8074619	2.853
lisp	7659404	2.707
python	7219357	2.551
php	3269837	1.155
fortran	2678345	0.946
cs	2336406	0.826

Cuadro 3.1: Los diez lenguajes más utilizados en Debian 4.0

La primera ventaja tiene relación con cómo funciona el instrumentador *sampler-cc* y su capacidad de no muestrear datos todo el tiempo, lo que Liblit denomina *instrumentación ligera*[8]. La segunda ventaja tiene que ver con el objetivo de llegar a distribuir aplicaciones modificadas para el estudio a un "público", para lo cuál es vital que el software distribuido se asemeje en lo más posible al original. Es decir, que no haga falta ejecutar más procesos de los de la propia aplicación o que no hagan falta por ejemplo modificaciones en el kernel[9].

Por otro lado, el inconveniente que ofrece la herramienta del proyecto de Liblit es que el proceso por el que una aplicación se convierte en su propio analizador (proceso de instrumentación) es dependiente del lenguaje utilizado.

3.3. Debian

Debian GNU/Linux o el proyecto Debian, es una distribución conformada por desarrolladores y usuarios alrededor del mundo que pretenden mantener un sistema operativo basado en software libre precompilado y empaquetado en un formato sencillo para diversas arquitecturas y varios núcleos. El motivo de su utilización en el proyecto radica en que el grupo de usuarios del grupo GSyC/LibreSoft utilizan distribuciones basadas en Debian, además de que el conocimiento sobre esta distribución es mayor que sobre el resto de distribuciones GNU/Linux.

La historia de esta distribución nació comenzada la década de los noventa de la mano de Ian Murdock, que por entonces era un estudiante de la Purdue University, de Indiana (EEUU). Desde su creación el proyecto aspiró a ser gestionado de una manera abierta, al estilo de Linux y GNU.

Pese al lento crecimiento inicial, el proyecto cuenta actualmente con más de mil desarrolladores, cada uno de ellos con un rol específico ya sea dentro del área de mantenimiento, documentación, control de calidad u otra relacionada con la infraestructura del proyecto. Además es conocido no sólo por su apoyo a Unix y el software libre, sino también por ofrecer una gran variedad de software.

[8]Dado que explicar el funcionamiento del instrumentador escapa del objetivo del capítulo, los detalles pueden ser consultados en el capítulo A del apéndice

[9]Se denomina *kernel* a la parte de un sistema operativo responsable de gestionar los recursos de una computadora

La versión actual ofrece unos quince mil paquetes de software para once arquitecturas de computadores, abarcando desde la arquitectura ARM comúnmente utilizada en sistemas empotrados hasta la más común empleada en los últimos ordenadores personales. Debian GNU/Linux es la base para muchas otras distribuciones, algunas de actualidad como Knoppix, Linspire o Ubuntu y otras que quedaron por el camino como Libranet, Corel Linux o Stormix Linux 2000.

El sistema de paquetes de Debian, que se explica a continuación, es una de sus características principales, siendo también de importancia sus estrictas políticas en lo que se refiere a calidad y publicaciones.

3.3.1. El sistema de paquetes en Debian GNU/Linux

Cada uno de los paquetes software de la distribución Debian tiene un mantenedor, que es el encargado de actualizar las versiones a partir de la versión del autor principal del software. Este mantenedor debe asegurar que el paquete cumple la política Debian, es coherente con el resto de paquetes de la distribución (de la versión de la distribución) y cumple los estándares de calidad de Debian. El mantenedor debe estar atento a los posibles errores y parches que aparezcan en la herramienta e incluirlos si fuera necesario. La tarea de mantenimiento de un paquete se lleva a cabo de manera individual o en grupos reducidos.

Periódicamente, un mantenedor publica una nueva versión del paquete, para lo que la sube al apartado de *incoming* en el archivo de paquetes de Debian. Una vez allí se comprueba entre otras cosas la integridad del paquete y la identidad del mantenedor. Una vez que se tiene la certeza de que el paquete alojado en *incoming* es válido, se distribuye a cientos de servidores espejo a lo largo de la red. En un principio, los paquetes subidos son aceptados en la rama inestable de paquetes, que es la que contiene la versión más actualizada de cada aplicación.

Los paquetes que están en la rama inestable, necesitan pasar por la rama en pruebas (*testing*) antes de ser candidatos para la rama estable (*stable*) de la distribución Debian. A su vez, para que el paquete pase a *testing* desde la rama *unstable* es necesario que se cumplan los siguientes requisitos:

- tiempo en inestable: debe de haber estado un mínimo de tiempo en la rama inestable (la duración exacta depende de la urgencia declarada en la subida)
- bugs de publicación críticos: no debe tener un número de bugs críticos mayor que la versión de la aplicación que está en la rama *testing*
- disponibilidad: debe estar compilado para todas las arquitecturas para las que se lanzará la versión
- dependencia: no debe depender de paquetes que no cumplan los anteriores requisitos

De esta manera, un error crítico para la publicación en un paquete del que varios dependen, como una biblioteca compartida, detendrá el paso de esos paquetes a la rama *testing* mientras esa biblioteca sea considerada deficiente.

La publicación de una nueva versión de Debian, debe ser pactada entre el coordinador de la publicación y los desarrolladores. Este primero elabora una serie de líneas de trabajo para preparar el lanzamiento. Una vez que todo el software que se considera más importante está razonablemente actualizado en la versión candidata para todas las arquitecturas, se produce el lanzamiento de la nueva versión estable.

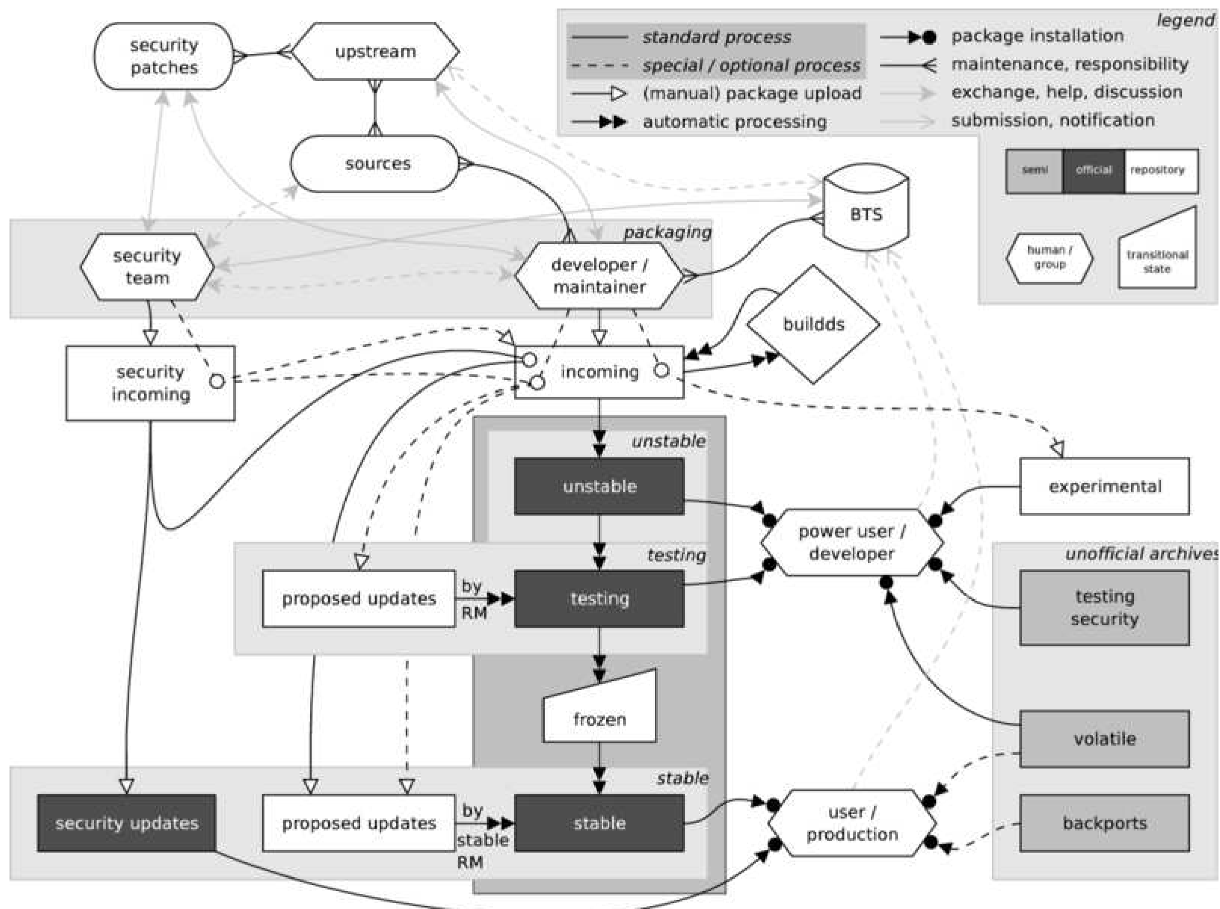

Figura 3.1: Ciclo de vida de los paquetes Debian.

3.3.2. Partes y creación de paquetes Debian

Los paquetes Debian son fichero *ar*[10] que incluyen dos ficheros comprimidos: uno que ofrece la meta-información sobre el paquete y otro que contiene los datos de la aplicación. Existen dos tipos de paquetes, los de binarios que ofrecen los ejecutables de las aplicaciones precompiladas y los paquetes que ofrecen el código fuente de la aplicación. Estos últimos son los que interesan para realizar la instrumentación sobre la aplicación, tras modificar el código original y realizar el proceso de empaquetamiento de ellos se obtendrá un paquete con binarios compilados especialmente para el estudio.

Dentro del fichero de datos de un paquete Debian de tipo fuente, se encuentra un directorio de nombre *debian* que contiene un grupo de ficheros que dirigen la compilación y construcción del paquete. Los más importantes son *control*, *changelog*, *copyright* y *rules*.

El fichero *control* contiene varios valores que herramientas de gestión de paquetes como *dpkg* utilizarán para gestionar la aplicación empaquetada. El fichero *copyright* como su nombre indica contiene información sobre la licencia y los derechos de copia. El fichero *changelog* es también utilizado por *dpkg* y otros programas para obtener el número de versión, revisión, distribución y urgencia del paquete. Por último y no menos importante, el fichero *rules* que es un Makefile[11],

[10] *ar* es una aplicación que agrupa varios ficheros en uno.

[11] Un *Makefile* es un fichero de texto que leído por el binario *make* describe la construcción de objetivos mediante reglas. Un *Makefile* contiene dependencias a nivel de código utilizadas en la compilación.

se compone de varias reglas que especifican cómo tratar las fuentes. Cada regla se compone de objetivos, ficheros o nombres de acciones que se deben llevar a cabo (por ejemplo, *build:* o *install:*). En un fichero *debian/rules* típico existen varias llamadas a commandos ofrecidos por *debhelper*, que son un conjunto de herramientas pequeñas, simples y fáciles de entender que son usadas para automatizar varios aspectos comunes a la hora de construir paquetes. Tienen funciones tan diversas como ejecutar strip[12] sobre ejecutables, bibliotecas compartidas y estáticas, calcular dependencias, comprobar directorios, facilitar la instalación de páginas de manual y limpiar los directorios de construcción de los paquetes.

Durante la ejecución del proyecto se ha utilizado la herramienta *debuild* para la creación de paquetes Debian. Esta llama a su vez a un comando de las herramientas ofrecidas por *dpkg* con el objetivo de comenzar la compilación y empaquetamiento; al finalizar ejecuta comprobaciones comunes a todos los paquetes Debian y firma el .deb obtenido con la clave pública del creador.

3.3.3. Ramas de desarrollos en Debian GNU/Linux

La última versión lanzada de Debian es llamada stable (estable). El último lanzamiento de la versión 4.0 (alias etch) como stable, ha propiciado que la versión que hasta ese momento era considerada como estable pase a ser oldstable (antigua estable).

Los alias de las versiones de Debian pertenecen a los nombres de los personajes del film *Toy Story*. El número de versiones publicadas asciende a nueve, siendo la primera del año 96:

- 4.0 etch, 8 de Abril del 2007
- 3.1 sarge, 6 de Junio del 2005
- 3.0 woody, 19 de Julio del 2002
- 2.2 potato, 15 de Agosto del 2000
- 2.1 slink, 9 de Marzo de 1999
- 2.0 ham, 24 de Julio de 1998
- 1.3 bo, 2 de Junio de 1997
- 1.2 rex, 12 de Diciembre de 1996
- 1.1 buzz, 17 de Junio de 1996

Como se ha mencionado en el apartado anterior, existen dos versiones más. La primera de ellas, la versión inestable (*unstable*) es en la que tiene lugar el desarrollo activo del proyecto, mientras que la segunda, la versión de pruebas (*testing*) será la siguiente a ser publicada como stable, ya que en ella se encuentran paquetes que han estado previamente en inestable y que contienen muchos menos fallos.

[12]En Unix strip es un programa que elimina todos los símbolos de depuración de errores de los binarios ejecutables y bibliotecas, ofreciendo un mejor rendimiento e incluso reduciendo el espacio ocupado en memoria secundaria

3.3.4. La aparición de Ubuntu

Es necesario mencionar la distribución quizá más popular de las que se han basado en Debian, ya que una parte del grupo de usuarios que van a utilizar las herramientas modificadas utilizan Ubuntu GNU/Linux.

La historia de Ubuntu comienza recientemente, cuando varios desarrolladores Debian y otra gente del entorno del software libre comenzaron a trabajar en esta nueva distribución, basada en los principios y paquetes de Debian pero patrocinada por una compañía, la empresa Canonical Ltd.

De acuerdo son sus fundadores, Debian era un proyecto demasiado burocrático donde cualquier propuesta interesante se ahogaba en un mar de discusiones. Tras varios meses de trabajo y un breve período de pruebas, la primera versión de Ubuntu (alias Warty Warthog) fue lanzada el 20 de octubre de 2004. Además de que el núcleo de los desarrolladores de Ubuntu son asalariados, hay más diferencias con Debian:

- Las versiones estables se liberan cada 6 meses y se mantienen actualizadas en materia de seguridad hasta 18 meses después de su lanzamiento.
- Dispone de 4 arquitecturas: Intel x86, AMD64, PowerPC, SPARC

El empeño por parte de la comunidad Ubuntu en ofrecer un software de escritorio más usable y más novedoso y la facilidad de instalación ha tenido éxito. Las estadísticas de Google Trends [Inc06] indican que desde finales del 2005 Google ha tenido más búsquedas sobre Ubuntu que sobre cualquier otra distribución de GNU/Linux. Un hecho que refleja su éxito es que el parlamento francés ha decidido migrar a Ubuntu [Inc07].

Pero no todo son críticas buenas en lo que se refiere a la corta vida de Ubuntu. La distribución y Canonical han recibido críticas por parte de la comunidad de software libre por la decisión de incluir un largo número de drivers propietarios[13] en la publicación de Ubuntu 7.04. También ha recibido críticas por parte de la comunidad Debian, que aludían a la excesiva bifurcación que han realizado sus desarrolladores del sistema operativo del que partieron.

3.4. GNOME

GNOME, junto a KDE el entorno de escritorio más utilizado en GNU/Linux, ofrece al estudio el conjunto de herramientas que se modificaron para su posterior distribución. Además de contar en el grupo de investigación con un alto conocimiento del proyecto GNOME, las aplicaciones de este proyecto en Debian ofrecen una forma unificada por la cual se obtiene a partir del software original el software empaquetado para esa distribución. Además de éstos motivos, la inclusión de aplicaciones GNOME en el estudio se vio motivada por su alta compatibilidad con la herramienta creada por Liblit, *sampler-cc*.

El proyecto GNOME (GNU Network Object Model Environment) surgió en agosto de 1997 de la mano de los mejicanos Miguel de Icaza y Federico Mena, con el objetivo de crear un entorno de escritorio para sistemas operativos de licencia libre, en especial para GNU/Linux. Muy poco antes de la creación de GNOME, Icaza que ya había trabajo en proyectos de software libre fue

[13] En el ámbito del software libre, se denomina *software propietario* a aquél que no tiene una licencia compatible con los principios del software libre.

invitado a una entrevista en Microsoft donde tuvo un acercamiento a las tecnologías ActiveX y COM, las que suscitaron en él gran interés por las arquitecturas de componentes.

En el momento de la creación del proyecto GNOME existía otro proyecto que pese a compartir objetivos, difería en medios. El nombre era KDE y pese a que mostraba resultados muy prometedores tenía un gran problema de cara a la comunidad de software libre. Este era que, a pesar de que sus programas eran libres, se escogió como toolkit[14] gráfico a la biblioteca *Qt*, propiedad de la empresa Trolltech y que tenía una licencia incompatible con la GPL[15].

GNOME necesitaba también de un toolkit gráfico, por lo que se planteó escribir un reemplazo libre para Qt. Sin embargo, esto se preveía muy costoso en tiempo y esfuerzo y el resultado no estaba asegurado. Finalmente se decidió utilizar GTK+, el toolkit gráfico que se utilizaba internamente en GIMP[16], programa de tratamientos de imágenes en el que Mena estaba trabajando. Con el tiempo GTK+ se separó de GIMP y se convirtió en el primer toolkit gráfico libre.

La primera versión que se distribuyó fue la GNOME 0.20 en junio de 1998 y en diciembre del mismo año se anunció la versión 0.99 que daría paso a la primera versión de GNOME publicada en Marzo de 1999.

Actualmente el proyecto goza de un amplio número de colaboradores, habiendo publicado en marzo del 2007 su versión 2.18 y estando programada para septiembre la versión 2.20.

3.5. Python

Python es un un lenguaje de programación de alto nivel creado por Guido van Rossum y publicado por primera vez en 1991. El principal objetivo que persigue este lenguaje es la facilidad, tanto de lectura, como de diseño. Se considera que su sintaxis es minimalista, mientras que su biblioteca estándar es enorme.

Python soporta más de un paradigma de programación y dispone de un recolector de basura[17]. Además es interpretado, lo que ahorra un tiempo considerable en el desarrollo del programa puesto que no es necesario compilar ni enlazar.

El nombre del lenguaje proviene de la afición de su creador original por los humoristas británicos Monty Python. Python posee una licencia de código abierto, denominada Python Software Foundation License, que es compatible con la licencia GPL.

[14]En este contexto se entiende por *toolkit gráfico* al conjunto de elementos que componen una interfaz gráfica de usuario que a menudo son implementados como una biblioteca.

[15]General Public License o licencia pública general

[16]*GIMP* fue el primer programa para usuarios finales de licencia completamente libre

[17]El *recolector de basura* es un mecanismo de gestión de memoria implementado en algunos lenguajes de programación

Capítulo 4

Objetivos

La meta que persigue este proyecto, es ofrecer un método por el que se pueda llegar a comprender el modo en el que se comporta una aplicación en tiempo de ejecución y relacionar esta información con la extraída de desarrollo del código fuente. Hasta ahora, el estudio del desarrollo del software libre ha estado centrado en el análisis de datos sobre el comportamiento de los desarrolladores alrededor del código, evaluando cómo se ha creado y cómo ha ido cambiando a lo largo del tiempo. Uniendo ambas bases de conocimiento, informes sobre el uso de las aplicaciones y sobre la manera en la que se han desarrollado, se dota al estudio del desarrollo del software libre de un parámetro del que antes carecía. Los siguientes apartados de este capítulo se han dedicado a desarrollar partiendo de esta meta cada uno de los objetivos perseguidos.

4.1. Comprender el comportamiento de un programa

El objetivo principal del proyecto es dar un primer paso en lo que se refiere a comprender el modo en el que funciona un programa. Este conocimiento se refiere a qué partes de la aplicación no son utilizadas, cuáles lo son y en qué medida. De cara al desarrollo de una aplicación, saber de antemano que por ejemplo varias funciones del código son las más ejecutadas, obliga al equipo de desarrollo a ser muy cuidadosos en ese código. Es más, el código que se utiliza más, debería ser el que más se cuida desde el punto de vista de rendimiento y posibles errores tanto de uso, como de seguridad si la aplicación lo requiere.

Hemos imaginado ya la utilidad que le puede dar a un desarrollador que ante la modificación de un código crítico, escribe un código lo más limpio posible; pero desde el punto de vista de gestión de un proyecto puede ser también muy interesante. Cualquier operación que necesite de un reparto de esfuerzos por parte del jefe del proyecto, debería ser contrastada con qué código es el más utilizado. Pongamos por caso la refactorización, que debería asignar más recursos a aquél código que es imprescindible para una ejecución *normal* de la aplicación.

Existen aún puntos más interesantes en lo que se refiere a gestión de recursos. Poniendo el ejemplo de cualquiera de las aplicaciones desarrolladas en el ámbito del software libre que utilizan herramientas para la gestión de errores[1] como *Bugzilla*, aparece otro caso en el que la gestión de recursos es muy importante. Este es, la resolución de los informes de error que un grupo de desarrolladores recibe, puesto que los errores más graves son aquellos que afectan a partes de código más utilizadas y que en consecuencia, ocurren más a menudo.

[1] *Bug Tracking Systems*

4.2. Realizar el estudio en un entorno real

Dado que el estudio necesita de los informes generados por las herramientas instrumentadas, uno de los objetivos del proyecto es realizar este en un ambiente real. Se entiende ambiente real por un grupo de usuarios que utilicen las herramientas instrumentadas para realizar acciones cotidianas.

La idea de este proyecto surgió del grupo de investigación de GSyC/LibreSoft, lo que facilitó que un número importante de integrantes de este se mostraran voluntarios a utilizar herramientas instrumentadas en su trabajo diario. Debido al ambiente dinámico del grupo, las herramientas más interesantes por su mayor frecuencia de utilización serán aquellas que se utilicen en la comunicación de grupos deslocalizados (como herramientas de correo, IRC, mensajería instantánea), además de otras de visualización de documentos y edición de texto.

Surge de este objetivo la necesidad de instrumentar aplicaciones de escritorio utilizadas por usuarios finales, hecho que sin duda alguna es sólo disponible en el ámbito del software libre, dado que se dispone del código original que deberá ser modificado para ser posteriormente ser redistribuido a los usuarios. Además se hace indispensable que las herramientas sean fáciles de instalar, para no ofrecer una barrera de entrada elevada a potenciales usuarios, al igual que el rendimiento ofrecido sea difícilmente distinguible del de la herramienta original.

4.2.1. Identificar partes críticas en aplicaciones GNOME

Dado que el grupo de usuarios del entorno real utilizan distribuciones basadas en Debian y utilizan el escritorio GNOME, es indispensable para el estudio instrumentar aplicaciones GNOME y ofrecerlas empaquetadas en Debian.

Como vimos antes, las aplicaciones prácticas de conocer cuales son las partes más utilizadas de una aplicación software son múltiples. La más evidente es destacar cuál es el código crítico de una aplicación, es decir, qué código debe ser modificado muy cuidadosamente no sólo en lo que se refiere a errores, sino también a rendimiento. Hasta ahora, los desarrolladores de aplicaciones GNOME no han tenido una manera de disponer de retroalimentación sobre las partes más utilizadas de las aplicaciones que ellos ofrecen al público. Conociendo las partes más utilizadas del código, pueden saber sobre que partes ser más cuidadosos y qué funcionalidades no llegan a ser tan utilizadas como se esperaba.

Para alcanzar este objetivo, será necesario servirse de las partes básicas del proyecto *Cooperative Bug Isolation* y adaptarlo a las necesidades del estudio. Finalmente, se ofrecerán un conjunto de aplicaciones compiladas especialmente con el objetivo de que, teniendo la misma funcionalidad que la original, sirvan para recolectar los datos necesarios por el estudio. Además, puesto que se pretende distribuir las aplicaciones a través de Debian, obteniendo una manera estándar de empaquetar aplicaciones GNOME en .deb se avanzará en el objetivo de facilitar la futura instrumentación automática del conjunto completo de aplicaciones del mencionado escritorio.

4.2.2. Relacionar partes críticas con historia del código fuente

Partiendo del conocimiento adquirido al identificar las partes críticas en aplicaciones GNOME y haciendo uso de la posibilidad del acceso a la historia del código que nos brindan los

repositorios de software del proyecto, será muy interesante observar la relación existente entre las partes de software que más se han modificado y las que más se están utilizando.

Para el análisis del repositorio software que contiene las herramientas de GNOME, se utilizará una herramienta de MSR conocida como CVSAnalY. Como se mencionó en el capítulo anterior, esta herramienta ha sido desarrollada por el grupo GSyC/LibreSoft, que es el mismo del que parte la idea del estudio, lo que facilitará su empleo.

Dado que la herramienta CVSAnalY observa y clasifica la actividad sobre los ficheros de un repositorio, será necesario que la información de las partes del código más utilizadas se agrupe por ficheros. Surgen a priori, una serie de preguntas sobre la posible relación entre los ficheros con código más utilizado y su número de modificaciones con respecto a la media. El objetivo de esa parte del estudio será comprobar precisamente interrogantes como si el código más utilizado es el que más se modifica o si el código que menos se utiliza el también el que menos se modifica.

Además de observar la relación entre modificaciones y uso en los ficheros, el estudio se centrará en identificar a los desarrolladores que han intervenido en las modificaciones de los ficheros más utilizados. Obteniendo una lista de los usuarios que modifican código más crítico, tendremos una idea aproximada de cuál es el grupo de desarrolladores que asume más responsabilidad en el proyecto.

4.2.3. Gráficas que relacionen informes con commits y medidas objetivas

Otro de los objetivos a realizar sobre los datos de uso y modificaciones obtenidos, es ofrecer gráficas que faciliten la visualización de la relación entre uso y modificaciones, así como el cálculo de medidas objetivas como la desviación estándar, mediana y coeficiente de variación.

La creación de las gráficas, si bien no ofrecerá nuevos datos, mostrará de una manera más evidente si la relación uso-modificaciones sigue patrones parecidos en las aplicaciones.

Por su parte, el cálculo de medidas como el *coeficiente de variación* permitirá que se pueda comparar la proporción existente entre la *media* y la *desviación típica*, lo que posibilitará una comparación objetiva entre el comportamiento de aplicaciones de diferente tamaño y actividad.

4.3. Aumentar el conocimiento sobre las herramientas de CBI

De manera tangencial al objetivo principal del proyecto, se persigue obtener un conocimiento mayor sobre las herramientas desarrolladas por Liblit para el proyecto CBI. Una de las dificultades a priori es la escasa documentación que ofrece el proyecto para efectuar estudios partiendo de las herramientas desarrolladas, por lo que se deberán efectuar pruebas hasta obtener el conocimiento necesario con el objetivo de llegar a instrumentar las aplicaciones deseadas.

4.4. Estudio de viabilidad sobre instrumentar distribuciones completas

El modo de comportarse de la aplicación con la que se ha escrito este documento es de por sí interesante, pero las posibilidades de acceso al código fuente que nos brinda el software libre hacen que sea un objetivo algo pequeño. Otro de los objetivos del proyecto, es evaluar la dificultad que conllevaría instrumentar todas las aplicaciones de una distribución. Para que esto

fuera posible el grado de automatización del proceso de instrumentación deberá ser muy alto, resultando difícil en cualquier otra situación.

Conseguir una distribución de la que las aplicaciones brindaran información sobre su uso, ofrecería una enorme cantidad de información que podría ser útil no sólo para evaluar el código de las aplicaciones de escritorio, sino también para observar el comportamiento de software de más bajo nivel como pudieran ser demonios[2] u otros binarios utilizados por el sistema.

[2]Un demonio es aquel programa que se ejecuta en segundo plano y no es controlado directamente por el usuario

Capítulo 5

Requisitos

5.1. Introducción

La descripción de los requisitos que a continuación se detallan se realizará intentando seguir la norma IEEE 830 o *IEEE Recommended Practice for Software Requirements Specifications* en todos los apartados en el que esto sea posible.

5.1.1. Propósito

El propósito de esta especificación de requisitos es ofrecer una descripción completa y global de la funcionalidad de operación que va a tener disponible el sistema.

5.1.2. Alcance

El resultado de la ejecución del proyecto se compone de un proceso en el cual se obtienen un conjunto de datos para su posterior análisis. El proceso de obtención de datos se realizará con el uso de paquetes Debian instrumentados, mientras que el software creado para el análisis tendrá las tareas de analizar los informes recibidos y relacionarlos con datos ofrecidos por la herramienta de MSR CVSAnalY.

5.1.3. Definiciones, acrónimos y abreviaturas

De ahora en adelante, usaremos las siguientes definiciones:

- Analizador: Elemento encargado de leer una entrada e interpretarla de acuerdo a ciertos criterios.
- Debian GNU/Linux: Sistema operativo GNU basado en software libre precompilado y empaquetado en un formato sencillo en múltiples arquitecturas y en varios núcleos.
- Commit: En el contexto de la Ciencia de la computación y la gestión de datos, commit(acción de cometer) se refiere a la idea de hacer que un conjunto de cambios "tentativos, o no permanentes"se conviertan en permanentes. Un uso popular es al final de una transacción de base de datos.
- CVS: Version Control System, sistema de gestión de versiones

- GPL: La GPL es una licencia que tiene como objetivo garantizar al usuario la libertad de compartir y cambiar software libre, es decir, asegurarse de que el software es libre para todos sus usuarios. Esta licencia Pública General es aplicable a la mayoría del software de la Free Software Foundation así como a cualquier otro programa cuyos autores se comprometan a usarlo.
- IA32: Arquitectura de microprocesadores de 32 bits de Intel.
- Instrumentador: Herramienta que, con el objetivo de que una aplicación software se convierta en su propio analizador, inserta código dentro del programa a ser analizado.
- MSR: Minería de Repositorios Software.
- Password: Palabra de paso.
- Repositorio: Máquina física, ya sea local o remota, que contiene el código fuente de un proyecto.
- Sites: Información que proporciona la herramienta de instrumentación en tiempo de compilación y que relaciona cada fichero y función con un identificador único.
- SVN: Subversion es un software de sistema de control de versiones diseñado específicamente para reemplazar al popular CVS.
- UNIX: UNIX es un sistema operativo de tiempo compartido (la computadora puede ser usada por varios usuarios al mismo tiempo).

5.1.4. Referencias

Estas son las referencias que han sido útiles para la realización de este capítulo:

- IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications [Std98]
- Sitio web de la herramienta CVSAnalY [GSy07]
- Sitio web del proyecto CBI [Lib06]

5.1.5. Resumen del resto del documento

El resto del documento describe de forma detallada los requisitos que se fueron construyendo en cada fase de la elaboración, así como su diseño e implementación . Todo esto evitando cualquier tipo de ambigüedad, así como buscando claridad y simplicidad.

5.2. Descripción general

5.2.1. Perspectiva del producto

El producto a desarrollar opera en dos etapas. La primera que se encarga de la obtención de los informes de uso de las aplicaciones y la segunda que se servirá de software creado para realizar un análisis de los datos obtenidos junto a los datos de la historia del código (que ofrece la herramienta CVSAnalY).

5.2.2. Interfaces del sistema

La interfaz del producto a desarrollar se englobará en el entorno de línea de comandos de un terminal UNIX.

5.2.3. Funciones del producto

El producto resultante debe ser capaz de:

- Ofrecer aplicaciones GNOME instrumentadas y empaquetadas según la política Debian para su instalación por parte de usuarios finales en Debian etch y Ubuntu feisty.
- Exportar los datos extraídos de los informes de uso a una base de datos.
- Relacionar los datos de uso albergados en la base de datos con la historia del código fuente de la aplicación que ofrece una herramienta de MSR.
- Generar listado con las funciones más utilizadas del código fuente de una aplicación.
- Generar listado con los ficheros que son más utilizados partiendo del listado de las funciones más utilizadas.
- Generar listado de desarrolladores de los ficheros más utilizados, incluyendo el número de commits realizados.
- Generar una gráfica que relacione para cada fichero de la aplicación el número de modificaciones contra el total de las llamadas a funciones acumuladas para ese fichero.

5.2.4. Características del usuario

Los usuarios que quieran utilizar o replicar el estudio deben tener conocimientos mínimos de SVN. Por el contrario si se desea ampliar el producto, se debe tener nociones avanzadas del sistema de paquetes Debian, del uso del instrumentador *sampler-cc*, de bases de datos, programación y finalmente de manejo y consulta de sistemas de control de versiones.

5.2.5. Restricciones

El producto resultante será dependiente de la plataforma Debian GNU/Linux, por lo que su uso se ve restringido a distribuciones basadas en Debian. Si se quiere emplear otra plataforma serán necesarias profundas modificaciones.

5.3. Requisitos específicos

5.3.1. Interfaz de usuario

No se realizará ninguna interfaz de usuario gráfica para analizar los informes, por lo que el software será utilizado en línea de comandos.

5.3.2. Interfaz Hardware

El producto se ha enfocado en software de escritorio utilizado sobre la arquitectura IA32.

5.3.3. Fuentes de entrada

Para la etapa de obtención de informes:

- Interfaz gráfica de usuario

Mientras que para la etapa de análisis de los informes:

- conjunto de parámetros pasados en línea de comandos

5.3.4. Destino de la salida

La salida del producto consistirá en un informe creado con sintaxis HTML.

5.3.5. Rangos válidos

Existen tres casos en los que puede existir un rango de datos inválido y es necesaria una comprobación. Estos son:

- aparición de informes de uso incompletos
- paso de parámetros erróneos a los scripts
- inexistencia de las bases de datos

5.3.6. Restricciones temporales

No existen restricciones de éste tipo referentes al análisis de la información.

5.3.7. Relaciones con otras entradas/salidas

El resultado del análisis de los informes de uso será relacionado con el resultado del estudio de la historia del código realizado por la herramienta de MSR CVSAnalY para cada aplicación estudiada.

5.4. Requisitos funcionales

5.4.1. Exportar los informes de uso a una base de datos

Validación de las entradas

La entrada a validar se leerá desde la línea de comandos y constará de dos cadenas de texto que se pasarán en forma de parámetros. Las cadenas contendrán:

- el nombre del directorio donde están guardados los informes
- el nombre del directorio donde se encuentra la información de los *sites*

Secuencia de las operaciones

- Llamar al ejecutable "report_parser.py" pasándole el nombre del directorio que contiene los informes y el nombre del fichero salida.

Mensajes de error y control de excepciones

- Si el directorio no existe, avisará del error por la salida estándar.
- Si durante el análisis encuentra informes inválidos, informará de ello en el log y continuará el análisis del resto de los informes.

Secuencia de la salida

La salida ofrecerá dos ficheros creados e información a través de la salida estándar.

El primero de los ficheros contendrá los informes listos para ser introducidos en la base de datos. El segundo será un fichero de log utilizado para corroborar que el análisis finalizó correctamente.

Finalmente la salida estándar mostrará para cada aplicación:

- nombre de la aplicación
- versión de la aplicación
- distribución y rama de desarrollo bajo la que se utiliza
- número de informes recolectados para la aplicación
- porcentaje medio de muestreo utilizado
- número total de llamadas a funciones recogido

5.4.2. Obtener el listado de funciones más utilizadas

Validación de las entradas

La entrada a validar se leerá desde la línea de comandos y constará de cinco cadenas de texto que se pasarán en forma de parámetros. Las cadenas contendrán:

- usuario de la base de datos
- password de la base de datos para el usuario antes incluido
- nombre de la base de datos que contiene los informes
- identificador de la aplicación a estudiar en la base de datos de los informes
- nombre de la base de datos que contiene la historia del código fuente de la aplicación
- fichero HTML de salida

Secuencia de las operaciones

Llamar al ejecutable "db_analizer.py" pasándole los parámetros previamente indicados.

Mensajes de error y control de excepciones

Informará por salida estándar del error en los siguientes casos:

- Si alguna de las base de datos no existen
- Si el usuario no tiene permisos de lectura sobre los datos
- Si el identificador de la aplicación no corresponde con ninguna tupla

Secuencia de la salida

La salida constará de un fichero HTML que contendrá una tabla con las cien funciones más utilizadas de la aplicación. La tabla constará de los campos:

- número total de llamadas
- nombre de la función
- fichero al que pertenece

5.4.3. Obtener el listado de los ficheros más utilizados y sus desarrolladores

Validación de las entradas

La entrada a validar se leerá desde la línea de comandos y constará de seis cadenas de texto que se pasarán en forma de parámetros. Las cadenas contendrán:

- usuario de la base de datos
- password de la base de datos para el usuario antes incluido
- nombre de la base de datos que contiene los informes
- identificador de la aplicación a estudiar en la base de datos de los informes
- nombre de la base de datos que contiene la historia del código fuente de la aplicación
- fichero HTML de salida

Secuencia de las operaciones

Llamar al ejecutable "db_analizer.py" pasándole los parámetros previamente indicados.

Mensajes de error y control de excepciones

Informará por salida estándar del error en los siguientes casos:

- Si alguna de las base de datos no existen
- Si el usuario no tiene permisos de lectura sobre los datos
- Si el id de la aplicación no corresponde con ninguna tupla

Secuencia de la salida

La salida constará de un fichero HTML que contendrá una tabla con los ficheros que más llamadas a funciones tienen acumulados y datos sobre la actividad de éstos desde el comienzo del proyecto a su publicación. La tabla constará de los campos:

- número total de llamadas de las funciones del fichero
- nombre del fichero
- número de commits sobre el fichero
- número de desarrolladores que realizaron los commits
- nombre de los desarrolladores que han realizado commits sobre el fichero así como el número de éstos

Además la salida ofrecerá una gráfica de puntos en la que cada fichero estará representado por uno. Esta tendrá dos ejes, el horizontal que ofrecerá información sobre el número de commits y el vertical sobre el total de llamadas.

5.4.4. Obtener el listado de los ficheros más utilizados y sus desarrolladores del último año

Validación de las entradas

La entrada a validar se leerá desde la línea de comandos y constará de seis cadenas de texto que se pasarán en forma de parámetros. Las cadenas contendrán:

- usuario de la base de datos
- password de la base de datos para el usuario antes incluido
- nombre de la base de datos que contiene los informes
- identificador de la aplicación a estudiar en la base de datos de los informes
- nombre de la base de datos que contiene la historia del código fuente de la aplicación
- fichero HTML de salida

Secuencia de las operaciones

Llamar al ejecutable "db_analizer.py" pasándole los parámetros previamente indicados.

Mensajes de error y control de excepciones

Informará por salida estándar del error en los siguientes casos:

- Si alguna de las base de datos no existen
- Si el usuario no tiene permisos de lectura sobre los datos
- Si el id de la aplicación no corresponde con ninguna tupla

Secuencia de la salida

La salida constará de un fichero HTML que contendrá una tabla con los ficheros que más llamadas a funciones tienen acumulados y datos sobre la actividad de éstos en el último año previo a su publicación. La tabla constará de los campos:

- número total de llamadas de las funciones del fichero
- nombre del fichero
- número de commits sobre el fichero en el año previo a la publicación
- número de desarrolladores que realizaron los commits en el año previo a la publicación
- nombre de los desarrolladores que han realizado commits sobre el fichero en el año previo a la publicación así como el número de éstos

Además la salida ofrecerá una gráfica de puntos en la que cada fichero estará representado por uno. Esta tendrá dos ejes, el horizontal que ofrecerá información sobre el número de commits del último año y el vertical sobre el total de llamadas.

5.5. Número máximo de usuarios

No existen restricciones de éste tipo referentes al análisis de la información.

5.6. Restricciones de diseño

El diseño del producto estará sujeto a las restricciones que ofrezca la herramienta de MSR CVSAnalY.

Capítulo 6

Diseño e Implementación

El desarrollo se ha llevado a cabo a través de un proceso iterativo en espiral, incrementando en cada iteración el desarrollo del mismo añadiendo nuevas funcionalidades a cada versión que se ha realizado.

Figura 6.1: Modelo de desarrollo en espiral

Las aplicaciones han sido instrumentadas para las distribuciones Debian etch y Ubuntu feisty, utilizando la herramienta de instrumentación *sampler-cc*. El lenguaje de programación que se ha usado para crear la scripts utilizados en el análisis ha sido Python. Se ha requerido el uso de las bibliotecas MySQLdb y DOM que han sido brevemente introducidas al final del capítulo.

El resto del capítulo está dedicado a explicar la funcionalidad obtenida en cada uno de los prototipos creados, el diseño de los scripts realizados para el análisis de los datos y a detallar cualquier decisión tomada a lo largo del proyecto.

6.1. Prototipos de aplicación instrumentada

Para aproximarse de una manera segura al objetivo de tener un conjunto de aplicaciones instrumentadas y empaquetadas en Debian, se trabajó en el desarrollo de prototipos en dos

iteraciones. La primera que buscó un primer contacto serio con la herramienta de análisis y la segunda que se centró en elaborar un proceso de instrumentación y empaquetamiento.

Una vez se hizo posible la instalación de la aplicación instrumentada como paquete Debian, se procedió en otra iteración a elaborar los requisitos necesarios para obtener un conjunto de scripts que realizaran un estudio de las relaciones existentes entre los informes de uso y la historia del código de la aplicación que nos ofrece la herramienta CVSAnalY.

6.1.1. Aplicación instrumentada

La primera etapa del proyecto se ha centrado en la obtención de una aplicación prototipo instrumentada con la herramienta de Liblit[1]. Las opciones eran o bien crear una aplicación escrita en lenguaje C con entorno gráfico GTK o bien utilizar una cuya complejidad sea baja.

Este apartado refleja el trabajo realizado desde su elección a su puesta en marcha.

Elección de la aplicación a instrumentar

Con el objetivo de poder utilizar una aplicación existente del entorno GNOME, que además de ahorrar tiempo ofrecerá una primera aproximación más real, se hizo uso de una herramienta de métricas de software, *SLOCCount* [2] de David A. Wheeler.

Una vez se obtuvo el código fuente de un conjunto de aplicaciones GNOME candidatas a ser instrumentadas y fueron analizadas con la herramienta de SLOCCount, se decidió utilizar *GNOME-terminal* o el terminal de GNOME, que siendo una aplicación de un número de líneas discreto ofrece la suficiente interacción con el usuario para probar a fondo el instrumentador.

La información que ofrece SLOCCount para *GNOME-terminal* es:

```
Totals grouped by language (dominant language first):
ansic:        19044 (99.78\%)
sh:              42 (0.22\%)

Total Physical Source Lines of Code (SLOC)                = 19,086
Development Effort Estimate, Person-Years (Person-Months) = 4.42 (53.08)
 (Basic COCOMO model, Person-Months = 2.4 * (KSLOC**1.05))
Schedule Estimate, Years (Months)                         = 0.94 (11.31)
 (Basic COCOMO model, Months = 2.5 * (person-months**0.38))
Estimated Average Number of Developers (Effort/Schedule)  = 4.69
Total Estimated Cost to Develop                           = \$ 597,575
 (average salary = \$56,286/year, overhead = 2.40).
SLOCCount, Copyright (C) 2001-2004 David A. Wheeler
```

Figura 6.2: SLOCCount aplicado a GNOME-terminal-2.14.2

De la información obtenida la parte que más interesa es que está escrita casi por completo en lenguaje C y que el número de líneas no pasa de veinte mil, lo que facilitará acudir al código

[1] En la sección del entorno tecnológico 3.2.2 se explica los motivos por los que se utilizó la herramienta creada por Liblit

[2] Esta cuenta el número de lineas de código fuente físicas de una aplicación, clasificándolas según lenguaje; además estima el esfuerzo que ha sido necesario utilizando el modelo COCOMO

fuente a comprobar el cometido de las funciones más utilizadas según el informe de uso.

Utilización de sampler-cc

Dado que sampler-cc actúa de capa o envoltorio alrededor de GCC, hay ciertos detalles que debemos tener en cuenta a la hora de la compilación. El primero es el tipo de muestreo que se va a utilizar para generar el ejecutable. A ésto Ben Liblit lo llama *instrumentation scheme* y puede ser una combinación de los siguientes: *bounds, branches, float-kinds, function-entries, g-object-unref, returns* y *scalar-pairs*. La combinación que elijamos repercutirá directamente en el ejecutable generado ya que cada una de ellas añadirá más código para el muestreo al fuente original. Dado que el objetivo de nuestro estudio sólo aspira a conocer el comportamiento de las llamadas a funciones deberemos elegir la instrumentación con el muestreo de *function-entries*.

El siguiente parámetro a tener en cuenta es el que se ocupa de generar la cuenta atrás, imprescindible para poder contar con la posibilidad de no realizar muestreos todo el tiempo. El parámetro que gestiona la cuenta atrás puede tener tres valores: *fixed, offline* y *online*. De los tres el único que no ofrece un muestreo aleatorio es el primero (*fixed*), mientras que la opción *offline* necesitará de una generación aleatoria previa a la ejecución de la aplicación. Para facilitar el muestreo utilizamos *online* que utilizará como semilla en tiempo de ejecución alguna fuente de entropía como podría ser /dev/urandom

El tercer y último parámetro necesario se refiere a la generación de código preparado para la ejecución multihilo. Por defecto la herramienta crea código que es monohilo, aunque pasará a multihilo si el parámetro *-pthread* de GCC es incluido en la compilación.

Una vez han sido elegidos los parámetros necesarios para la obtención del ejecutable instrumentado, se procede a compilar la herramienta substituyendo el compilador GCC por su adaptador *sampler-cc*. GNOME terminal como el resto de aplicaciones del escritorio GNOME, se distribuyen con una serie de scripts generados por Autoconf[3] que ayudan a personalizar y simplificar la configuración, compilación y posterior instalación. Para compilar la herramienta con un compilador personalizado, haremos uso de las variables de entorno, que son leídas por los scripts antes mencionados.

```
$ export CC="/usr/bin/sampler-cc"
$ export CFLAGS="--sampler-scheme=function-entries --sampler-random=online"
```

Una vez declaradas las variables de entorno podemos proceder a la compilación:

```
$ ./configure && make
```

A éstas alturas, la aplicación ya ha sido instrumentada y está lista para ser probada, si bien es cierto que el siguiente paso lógico es extraer la información asociada a cada función y fichero del código fuente.

Obtención de los puntos de instrumentación

Para la validación de un informe de uso generado por la aplicación es necesario contar antes con la información que se generó en la compilación. Dicho de otra manera, si en la compilación

[3] *Autoconf* es una herramienta para producir scripts de shell que de manera automática configuren paquetes de código fuente para adaptarlo a distintos tipos de sistemas de tipo UNIX

no se obtuvo correctamente ésta información, que Liblit llama *sites*, no será posible obtener información útil del informe. Los *sites* se encuentran empotrados dentro de cada fichero objeto, biblioteca compartida o ejecutable que fue instrumentado en secciones ELF[4] diseñadas a medida para albergar cada parte de esos datos.

```
luis@fraggle:~/Gterm$ readelf -S src/GNOME-terminal|grep site_info
  [34] .debug_site_info  PROGBITS        00000000 088c08 0153e2 00      0   0  1
```

Con el objetivo de extraer la información con los puntos de instrumentación para cada fichero, se utiliza un sencillo script creado por Liblit. Este script extrae la información encontrada en la sección ELF *debug_site_info* y la vuelca a un fichero. Para acelerar el proceso y dado que la información puede estar repartida en varios ficheros, se ha creó un script que por cada fichero con secciones ELF extrae la información y la vuelca a un fichero.

El resultado de la obtención de *sites* es un fichero que tiene para cada fichero instrumentado un identificador único y a continuación una línea por cada función con el nombre del fichero, el nombre de la función y el número de línea en el que se encuentra[5]:

```
<sites unit="f8e6649ebb331ea1eb4fec44a31eadb6" scheme="function-entries">
/home/luis/Gterm/src/skey/skeyutil.c       3       skey_sevenbit   0
/home/luis/Gterm/src/skey/skeyutil.c       11      skey_lowcase    0
</sites>
```

Revisión del informe generado

La ejecución de la aplicación instrumentada necesitará una ligera configuración si queremos obtener el informe sin preocuparnos aún de instalar los datos de configuración en la base de datos *Gconf*[6]. Necesitaremos la creación de dos variables de entorno, la primera para que el informe sea volcado ahí cuando la aplicación acabe y la segunda que se refiere a la frecuencia con la que se produce el muestreo. Para nuestro ejemplo pondremos el muestreo máximo, el valor 1, ya que 1/1 muestreos de producen (siendo el mínimo el valor 100 donde sólo se muestrea 1/100 veces).

```
$ export SAMPLER_FILE="/tmp/sampler_report"
$ export SAMPLER_SPARSITY=1
```

Una vez configuradas las nuevas variables de entorno ejecutaremos la aplicación y al terminar la ejecución de esta, el fichero declarado dentro de *SAMPLER_FILE* contendrá el informe de uso de la aplicación.

```
<samples unit="f8e6649ebb331ea1eb4fec44a31eadb6" scheme="function-entries">
0
0
</samples>
```

[4]Estas secciones pueden albergar diferente información como comentarios, código ejecutable, información para el enlazado dinámico, datos para búsqueda de errores, tablas de símbolos, tablas de cadenas de texto y notas.

[5]Existe además un cuarto campo que por nula utilidad para el objetivo del estudio no se nombra, su propósito es realizar diagramas de relación entre subrutinas

[6]Gconf es un sistema utilizado por el escritorio GNOME para albergar la configuración para el escritorio y las aplicaciones que se ejecutan en él.

Ahora es cuando el informe de uso[7] se puede relacionar la información que proporcionan los *sites* y así deducir que para nuestro ejemplo las funciones *skey_sevenbit* y *skey_lowcase* del fichero *src/skey/skeyutil.c* no fueron utilizadas.

6.1.2. Aplicación empaquetada en Debian

Antes de iniciar el proceso de obtener el .deb[8] de una aplicación instrumentada, se empaquetaron las herramientas del proyecto CBI, ya que una de ellas es imprescindible para el funcionamiento correcto del resto de aplicaciones instrumentadas y empaquetadas. Esta herramienta se incluye en el paquete *sampler* y contiene los scripts Python que manejarán el informe resultante de la aplicación y lo enviarán a un servidor.

Para crear el primer .deb con una aplicación instrumentada, se obtuvo el fuente de la aplicación en Debian así como sus ficheros de configuración originales y se aplicaron los cambios necesarios en la configuración del paquete para que tras la compilación se obtuviera la herramienta instrumentada.

Antes de comenzar a explicar las modificaciones necesarias en la aplicación, es obligatorio referirse a las modificaciones que se tuvieron que efectuar sobre diferentes ficheros ofrecidos por Debian para la creación de paquetes de aplicaciones GNOME. Estas modificaciones están disponibles en el apéndice, en la sección de *Modificación de Makefiles de Debian*B.

Modificación del paquete Debian

Los primeros cambios que se aplican conciernen a la información de las dependencias de la aplicación y a su información descriptiva. Posteriormente se debe modificar el fichero encargado de la configuración y compilación del código fuente a través de ficheros Makefile[9]. El listado de ficheros a modificar en el fuente del paquete Debian de la aplicación es el siguiente:

- debian/control
- debian/changelog
- debian/rules

El fichero *control* contiene la información más vital sobre el fuente del paquete y sobre los binarios que este crea. El primer párrafo del fichero de control contiene información sobre el paquete fuente en general, mientras que el siguiente se dedica a configurar cada paquete binario que se construye desde el fuente. En general la información más importante que se puede encontrar en este fichero es la relativa al mantenedor del paquete Debian, su sección dentro de la distribución, la descripción del binario generado, las dependencias de instalación y compilación, la arquitectura y por supuesto el nombre. Para el prototipo sobre GNOME-terminal (ver figura 6.3) se añadió al fichero control la dependencia que incluye el paquete generado con los scripts Python del proyecto CBI llamado *sampler* y un texto descriptivo sobre esta versión instrumentada.

[7] La sección C del apéndice ofrece un ejemplo más completo sobre un informe de uso

[8] *deb* es la extensión que se asocia a los ficheros que contienen un paquete Debian

[9] Este término, que se usará en el resto del capítulo, se refiere a un fichero que contiene instrucciones para la herramienta *make*, que es una utilidad para la compilación automática de aplicaciones grandes

```
Package: GNOME-terminal
Architecture: i386
Depends: ${shlibs:Depends}, scrollkeeper (>= 0.3.14-5),
 GNOME-control-center (>= 1:2.8.0), GNOME-terminal-data (= 2.14.2-1),
 sampler(>=1.4.6)
Conflicts: GNOME-terminal2
Replaces: GNOME-terminal2
Provides: x-terminal-emulator
Recommends: yelp
Description: The GNOME 2 terminal emulator application
 GNOME Terminal is a terminal emulation application that you can use to
 perform the following actions:
 .
 - Access a UNIX shell in the GNOME environment.
 - Run any application that is designed to run on VT102, VT220, and xterm
 terminals.
 .
 GNOME Terminal features the ability to use multiple terminals in a single
 window (tabs) and profiles support.
 .
 This *special package* was rebuild specially for the MUCode project
 which aims to analyse the behaviour of this application and its
 developers. Each time you use this application you can help us to improve
 this knowledge and make GNOME-terminal better for everyone.
```

Figura 6.3: Modificación aplicada al fichero control

```
GNOME-terminal (2.14.2-1.0mucode1) unstable; urgency=low

  * Non-maintainer upload.
  * Built with sampler-cc for the MUCode project

 -- Luis Cañas Díaz <lcanas@gsyc.es>  Tue, 17 Apr 2007 23:09:05 +0200
```

Figura 6.4: Modificación aplicada al fichero changelog

El segundo fichero a modificar, el fichero *changelog*, necesita una modificación también trivial. Este fichero se utiliza para registrar los cambios realizados sobre la aplicación o en este caso paquete Debian. Se introdujo una breve descripción sobre los cambios que se van a realizar a GNOME-terminal, como se puede ver en la figura 6.4.

Después de modificar la información que va a ofrecer el nuevo paquete Debian, es necesario modificar el fichero *rules*, que es el encargado de llamar a los ficheros de configuración y compilación del código fuente. A este como se puede ver en la figura 6.5 se le añade la regla de instalación por la cual se llama al script python *install-wrappers*, cuya misión es substituir al ejecutable instrumentado por un script que lo envuelve y recoge su salida para posteriormente enviarlo a través de la red.

Una vez configurado todo lo básico para empaquetar en Debian, la compilación es sencilla. Valiéndonos de la herramienta *debuild* y de las modificaciones hechas a la versión de la aplicación ofrecida por Debian, se compila a través de la herramienta *sampler-cc* con las opciones que conseguirán una instrumentación centrada en vigilar la actividad de las funciones.

```
$ export CC=/usr/bin/sampler-cc
$ export CFLAGS="--sampler-scheme=function-entries --sampler-random=online"
$ debuild -d --preserve-envvar=CFLAGS --preserve-envvar=CC > ../output.txt
```

El resultado es un paquete Debian listo para ser instalado y/o distribuido que contiene los binarios compilados con el analizador *sampler-cc* y un fichero que contiene los *sites*.

La aplicación instalada, envía al finalizar el informe a un servidor web mediante una petición POST[10].

6.2. Selección de aplicaciones objetivo para el estudio

Tras definir el trabajo a realizar con las aplicaciones instrumentadas y elaborar los prototipos, fue necesario elegir un conjunto de aplicaciones que tuviera un interés suficiente como para ser utilizado por un grupo de usuarios voluntarios.

Dado que el grupo de usuarios que eran objetivo básico se encontraban en el entorno de trabajo del grupo de investigación de GSyC/LibreSoft, se buscó instrumentar aplicaciones que utilizaran habitualmente. Estas son aplicaciones de gestión de la comunicación, lectores de documentos y herramientas típicas en la administración de sistemas. De igual manera, dado que los voluntarios utilizaban *Ubuntu feisty* o *Debian etch*, se crearon paquetes para ambas distribuciones (algunos de versiones distintas, una para cada distribución).

[10]POST es un método utilizado para realizar peticiones HTTP.

```
# -*- mode: makefile; coding: utf-8 -*-

include /usr/share/cdbs/1/rules/debhelper.mk
include /usr/share/cdbs/1/rules/utils.mk
include /usr/share/cdbs/1/rules/simple-patchsys.mk
include /usr/share/cdbs/1/class/GNOME.mk
include /usr/share/GNOME-pkg-tools/1/rules/uploaders.mk

DEB_CONFIGURE_SCRIPT_ENV += LDFLAGS="-Wl,-O1 -Wl,--as-needed"
DEB_CONFIGURE_EXTRA_FLAGS += --disable-scrollkeeper

build/GNOME-terminal::
/usr/bin/docbook-to-man debian/GNOME-terminal.sgml > debian/GNOME-terminal.1

binary-install/GNOME-terminal::
/usr/lib/sampler/tools/install-wrappers --name=GNOME-terminal --version=2.14.2
--release=2.14.2-1 --install=debian/GNOME-terminal
--template=/usr/lib/sampler/tools/wrapper.in /usr/bin/GNOME-terminal

clean::
rm -f debian/GNOME-terminal.1
```

Figura 6.5: Modificación aplicada al fichero rules

En el resto de la sección aparece un listado de las aplicaciones elegidas, incluyendo una breve descripción de su funcionamiento, una clasificación de su código y un comentario sobre detalles que puedan ser importantes a la hora de analizar los resultados.

6.2.1. EOG

La aplicación *Eye of GNOME* cuenta con dos versiones empaquetadas e instrumentadas. La primera de ellas, la versión *2.16.3*, se empaquetó para la versión estable de Debian. La segunda, versión *2.18.1* se empaquetó para su uso en Ubuntu Feisty.

Descripción

Eye of GNOME o eog es un visor de gráficos para el escritorio GNOME que usa la biblioteca gdk-pixbuf. Puede tratar con imágenes grandes y hacer zoom y navegar por la imagen con uso constante de memoria. El objetivo es un visor estándar de gráficos para futuras publicaciones de GNOME.

Datos sobre el código fuente

A continuación se ofrecen algunos de los datos que SLOCCount da para el código fuente de la aplicación en sus dos versiones.

eog-2.16.3

```
Totals grouped by language (dominant language first):
ansic:        20498 (99.82%)
sh:              36 (0.18%)

Total Physical Source Lines of Code (SLOC)                = 20,534
SLOCCount, Copyright (C) 2001-2004 David A. Wheeler
```

eog-2.18.1

```
Totals grouped by language (dominant language first):
ansic:        22211 (99.83%)
sh:              38 (0.17%)

Total Physical Source Lines of Code (SLOC)                = 22,249
SLOCCount, Copyright (C) 2001-2004 David A. Wheeler
```

Detalles

La aplicación EOG ofrecerá informes de uso de versiones muy cercanas en el tiempo, dos meses y diez días. Debido a la cercanía en el tiempo de éstas, la comparación con la historia del código será prácticamente igual para ambas, por lo que se optará en principio por la versión que más informes de uso aporte para el estudio.

6.2.2. Evince

La aplicación *Evince* cuenta con dos versiones empaquetadas e instrumentadas. La primera de ellas, la versión *0.4.0*, se empaquetó para la versión estable de Debian. La segunda, versión *0.8.1* se empaquetó para su uso en Ubuntu Feisty.

Descripción

Evince es un visor multi-página de documentos. Puede mostrar e imprimir PDF, PostScript (PS), PostScript encapsulado (EPS), DJVU y DVI. Si lo admite el documento, permite búsqueda de texto, copiar texto al portapapeles, navegación por el texto y entre la tabla de contenidos.

Datos sobre el código fuente

A continuación se ofrecen algunos de los datos que SLOCCount da para el código fuente de la aplicación en sus dos versiones.

0.4.0

```
Totals grouped by language (dominant language first):
ansic:        36777 (93.24%)
cpp:           2620 (6.64%)
sh:              48 (0.12%)

Total Physical Source Lines of Code (SLOC)                = 39,445
SLOCCount, Copyright (C) 2001-2004 David A. Wheeler
```

0.8.1

```
Totals grouped by language (dominant language first):
ansic:        50926 (97.14%)
cpp:           1398 (2.67%)
python:          53 (0.10%)
sh:              48 (0.09%)

Total Physical Source Lines of Code (SLOC)                = 52,425
SLOCCount, Copyright (C) 2001-2004 David A. Wheeler
```

Detalles

Evince comenzó como una refactorización del código de *GPdf*, pero en un corto período de tiempo alcanzó más funcionalidad que este. La primera versión estudiada, la *0.4.0*, fue lanzada a los 8 meses y la *0.8.1* 20 meses después. En ese tiempo y según mostró SLOCCount en la sección anterior, el desarrollo aumentó el código de la aplicación en un 37 % por lo que será interesante ver si las funciones más utilizadas son parecidas en ambas y si siguen los mismos desarrolladores alrededor de ellas.

6.2.3. Evolution

La aplicación *Eye of GNOME* cuenta con dos versiones empaquetadas e instrumentadas. La primera de ellas, la versión *2.16.3*, se empaquetó para la versión estable de Debian. La segunda, versión *2.18.1* se empaquetó para su uso en Ubuntu Feisty.

Descripción

Evolution es un gestor de correo que integra además, calendario, agenda de contactos, listas to-do y herramientas de recordatorio. La funcionalidad extra incluye integración con servidores Exchange y Groupwise, integración con LDAP, calendarios web y sincronización con dispositivos Palm.

Datos sobre el código fuente

A continuación se ofrecen algunos de los datos que SLOCCount da para el código fuente de la aplicación en sus dos versiones.

evolution-2.6.3

```
Totals grouped by language (dominant language first):
ansic:       295277 (98.74%)
perl:          2093 (0.70%)
cs:            1071 (0.36%)
sh:             592 (0.20%)

Total Physical Source Lines of Code (SLOC)                = 299,033
SLOCCount, Copyright (C) 2001-2004 David A. Wheeler
```

evolution-2.10.1

```
Totals grouped by language (dominant language first):
ansic:       306623 (98.90%)
perl:          1760 (0.57%)
cs:            1071 (0.35%)
sh:             592 (0.19%)

Total Physical Source Lines of Code (SLOC)                = 310,046
SLOCCount, Copyright (C) 2001-2004 David A. Wheeler
```

Detalles

Evolution, una de las aplicaciones más conocidas del entorno GNOME tiene algunos detalles a los que se debe prestar atención en su estudio. El primero es que su código tiene un tamaño considerable y que este creció un 3 % en los nueve meses que separan a las dos versiones. El segundo es que de las aplicaciones estudiadas es la que más ficheros fuente tiene, sobrepasando los seiscientos.

Problemas durante la instrumentación

Al igual que otras herramientas, Evolution tiene inconsistencias en los parámetros referentes a la compilación monohilo y multihilo. Al intentar compilar código thread-safe[11] y non-thread-safe[12] se obtienen errores en tiempo de enlazado.

Por defecto, la herramienta hecha por Liblit, compila código *non-thread-safe.* Si en algún momento aparece código compilado de manera segura (con pthread") se producirá un error. La solución es cambiar la invocación al instrumentador para que compile y enlace código adecuado para ejecución multihilo. La variable de entorno *CC* debería ser cambiada así antes de la compilación:

```
CC="/usr/local/bin/sampler-cc --threads"
```

6.2.4. Gaim

La aplicación *Gaim* cuenta con dos versiones empaquetadas e instrumentadas. La primera de ellas, la versión *2.0.0+beta5*, se empaquetó para la versión estable de Debian. La segunda, versión *2.0.0+beta6* se empaquetó para su uso en Ubuntu Feisty.

Descripción

Gaim es un cliente gráfico de mensajería instantánea multi-protocolo. Es capaz de usar AIM/ICQ, Yahoo!, MSN, IRC, Jabber, Napster, Zephyr, Gadu-Gadu, Bonjour, Groupwise, Sametime, y SIMPLE a la vez.

Datos sobre el código fuente

A continuación se ofrecen algunos de los datos que SLOCCount da para el código fuente de la aplicación en sus dos versiones.

gaim-2.0.0+beta5

```
Totals grouped by language (dominant language first):
ansic:       223422 (98.98%)
perl:           997 (0.44%)
python:         857 (0.38%)
cs:             176 (0.08%)
sh:             126 (0.06%)
tcl:             86 (0.04%)
pascal:          66 (0.03%)

Total Physical Source Lines of Code (SLOC)                = 225,730
SLOCCount, Copyright (C) 2001-2004 David A. Wheeler
```

[11] Código *thread-safe* es aquél que es apto para la ejecución multihilo

[12] Código *non-thread-safe* es aquél que no es apto para la ejecución multihilo

gaim-2.0.0+beta6

```
Totals grouped by language (dominant language first):
ansic:       229428 (98.99%)
perl:           989 (0.43%)
python:         865 (0.37%)
cs:             201 (0.09%)
sh:             133 (0.06%)
tcl:             86 (0.04%)
pascal:          66 (0.03%)

Total Physical Source Lines of Code (SLOC)               = 231,768
SLOCCount, Copyright (C) 2001-2004 David A. Wheeler
```

Detalles

Al igual que la aplicación *EOG*, ésta ofrecerá informes de uso de versiones muy cercanas en el tiempo, dos meses y diez días. Dado que la comparación con la historia del código será prácticamente igual para ambas, se optará por la versión de la que más informes de uso se hayan recolectado.

Problemas durante la instrumentación

Durante la compilación de Gaim apareció un problema en la compilación del código fuente. Uno de los binarios, en concreto el llamado *gaim-text*, no llegaba a compilarse cuando se utilizaba la herramienta *sampler-cc* en lugar del compilador por defecto.

El fichero encargado de realizar la configuración previa a la compilación emitía un mensaje de error diciendo que no se cumplían los requisitos por la carencia de una biblioteca que estaba ya correctamente instalada.

```
configure: WARNING:

/usr/include/ncursesw/ncurses.h: present but cannot be compiled
/usr/include/ncursesw/ncurses.h: check for missing prerequisite headers?
/usr/include/ncursesw/ncurses.h: see the Autoconf documentation
/usr/include/ncursesw/ncurses.h: section "Present But Cannot Be Compiled"
/usr/include/ncursesw/ncurses.h: proceeding with the preprocessor's result
/usr/include/ncursesw/ncurses.h: in the future, the compiler will take precedence
```

El problema estaba relacionado con la herramienta de Liblit ya que compilando con *GCC*[13] no aparecía el problema. La solución, que aportó Liblit semanas más tarde, tenía que ver con el no siempre correcto uso de los booleanos como tipo primitivo. Los parámetros que se debieron añadir a los parámetros de CFLAGS fueron lo siguientes:

```
-DNCURSES_ENABLE_STDBOOL_H=0 -Dfalse=FALSE
```

[13]GCC es el compilador de lenguaje C por defecto en GNU/Linux

Además, una de las funciones de la aplicación impedía que la compilación terminara correctamente, para lo que se prefirió evitarla añadiendo el parámetro *–exclude-function=gaim_markup_html_to_xhtml* a la variable de entorno.

Finalmente y tras observar algunas inconsistencias en el código en lo que se refiere a la compilación con monohilo o multihilo, se utilizó el parámetro *–threads* para añadirlo a la variable de entorno *CC*.

Las variables de entorno pasaron a tener éstos valores, que serían luego utilizados por la compilación mediante *debuild* que fue explicada en secciones anteriores.

```
CFLAGS="--sampler-scheme=branches --sampler-scheme=g-object-unref
--sampler-scheme=scalar-pairs --sampler-scheme=returns --sampler-
random=online -DNCURSES_ENABLE_STDBOOL_H=0 -Dfalse=FALSE
--exclude-function=gaim_markup_html_to_xhtml"
CC="/usr/local/bin/sampler-cc --threads"
```

6.2.5. GNOME-terminal

La aplicación *GNOME-terminal* cuenta con una versión empaquetada e instrumentada para la versión estable de Debian.

Descripción

GNOME Terminal es una aplicación que emula un terminal que se puede usar para las siguientes acciones:

- Acceder a una shell de UNIX en el entorno GNOME.
- Ejecutar cualquier aplicación que está diseñada para VT102, VT220 y los terminales xterm.

Datos sobre el código fuente

A continuación se ofrecen algunos de los datos que SLOCCount da para el código fuente de la aplicación en sus dos versiones.

gnome-terminal-2.14.2

```
Totals grouped by language (dominant language first):
ansic:        19044 (99.78%)
sh:              42 (0.22%)

Total Physical Source Lines of Code (SLOC)                = 19,086
SLOCCount, Copyright (C) 2001-2004 David A. Wheeler
```

Detalles

Esta aplicación se instrumentó como resultado del primer prototipo de aplicación instrumentada y empaquetada en Debian. El número de ficheros no supera los veinte, por lo que el estudio por parte de la aplicación de MSR no conseguirá demasiados detalles.

6.2.6. Rhythmbox

La aplicación *Rhythmbox* cuenta con dos versiones empaquetadas e instrumentadas. La primera de ellas, la versión *0.9.6*, se empaquetó para la versión estable de Debian. La segunda, versión *0.10.0* se empaquetó para su uso en Ubuntu Feisty.

Descripción

Rhythmbox es un sencillo reproductor de música que soporta una amplia gama de formatos de audio (incluyendo mp3 y ogg). Originalmente inspirado en el iTunes de Apple, la versión actual soporta además: radio a través de internet, integración con iPod y demás reproductores genéricos portátiles, creación de audio CD, audio CD playback y podcasts.

Datos sobre el código fuente

A continuación se ofrecen algunos de los datos que SLOCCount da para el código fuente de la aplicación en sus dos versiones.

rhythmbox-0.9.6

```
Totals grouped by language (dominant language first):
ansic:        75976 (98.57%)
python:        1060 (1.38%)
sh:              45 (0.06%)

Total Physical Source Lines of Code (SLOC)                = 77,081
SLOCCount, Copyright (C) 2001-2004 David A. Wheeler
```

rhythmbox-0.10.0

```
Totals grouped by language (dominant language first):
ansic:        84094 (96.97%)
python:        2583 (2.98%)
sh:              45 (0.05%)

Total Physical Source Lines of Code (SLOC)                = 86,722
SLOCCount, Copyright (C) 2001-2004 David A. Wheeler
```

Detalles

Las dos versiones de *Rhythmbox* fueron lanzadas con medio año de diferencia. En ese tiempo el código creció en un 12 %.

6.3. Clases utilizadas por los scripts de análisis

En esta sección se detallan cuales son las clases que se realizaron en el diseño de los scripts, cuya misión es interpretar los datos de los informes de uso y relacionarlos con los ofrecidos por la herramienta de MSR CVSAnaly.

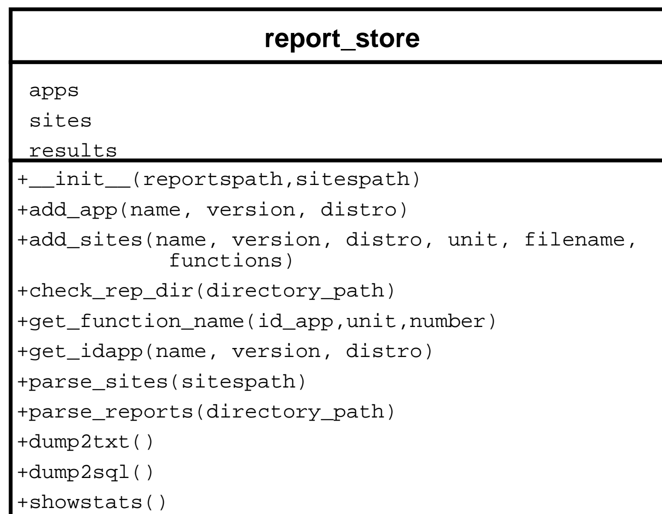

Figura 6.6: Clase report_store

El desarrollo realizado en esta etapa comenzó después de tener la primera aplicación instrumentada y empaquetada en Debian. Los requisitos capturados se encuentran detallados en el capítulo anterior. La herramienta CVSAnalY crea como resultado de su ejecución una base de datos de sintaxis compatible con MySQL, por lo que en el diseño de los scripts se se buscará la compatibilidad con esa base de datos.

6.3.1. Clase report_store

La clase *report_store* que se puede consultar en la figura 6.6 será la encargada de analizar los informes de uso cuya estructura fue explicada en el apartado anterior. Lo primero que realiza la clase es un análisis de los directorios de los informes, chequeando cuales tienen la información adecuada. Después deberá preocuparse de extraer la información del fichero *environment* y de analizar el fichero XML con el informe de uso.

Antes de comenzar el análisis de los informes, la clase carga en memoria los identificadores únicos de ficheros con los que podrá relacionar la información de los informes con los nombres de ficheros y funciones del código fuente.

6.3.2. Clase Analyzer

La clase *analyzer* que se puede consultar en la figura 6.7 se encargará de relacionar la base de datos que contiene las funciones más utilizadas de una aplicación y la base de datos que genero CVSAnalY con la historia del código fuente.

La mayoría de los métodos de la clase están dedicados a realizar consultas a la base de datos de la historia del código, con el objetivo de obtener el número de commits que tuvo cierto fichero, el número de desarrolladores que lo modificaron o el nombre de éstos.

```
analyzer
-----------------------------------------------
top_functions
top_files
cbi_cursor
cvsanaly_cursor
-----------------------------------------------
+__init__(dbuser,dbpass,cbidb,cbiapp,ofile,
          cvsdb="")
+get_lastdate()
+get_files_ids()
+get_commiter(commiter_id)
+get_commits_commiter(commiter_id, file_id,
                       releasedate="",daysbefore=0)
+get_commiters_on_file(file_id,releasedate="",
                       daysbefore=0)
+get_num_commits(file_id,releasedate="",
                 daysbefore=0,)
+get_ranking_of_file_id(file_id)
+top_functions(id)
+top_files(id)
+exists(list, unit)
+sort(files)
+dump_total_functions()
+dump_commiters_file_last_year()
+dump_commiters_file(daysbefore=0)
```

Figura 6.7: Clase analyzer

6.4. Funciones de los scripts de análisis

6.4.1. Exportar los informes de uso a una base de datos

El caso de uso comienza cuando el usuario efectúa la llamada al analizador de informes y le pasa en línea de comandos el directorio que contiene los informes y el directorio que contiene los sites. El objeto *report_store* se encargará primero de analizar todos los informes y una vez hecho ésto, de relacionar éstos con la información que aportan los *sites* sobre los nombres de fichero y función. Una vez se obtiene la información estructurada en memoria, se procede a exportar esta información a un fichero para su posterior carga en una base de datos MySQL.

El diseño de la base de datos que contendrá los informes de uso es el reflejado en la figura 6.8.

6.4.2. Obtener el listado de funciones más utilizadas

El caso de uso comienza cuando el usuario llama al analizador de informes y le pasa los siguientes parámetros:

- usuario de la base de datos
- password de la base de datos para el usuario antes incluido
- nombre de la base de datos que contiene los informes
- identificador de la aplicación a estudiar en la base de datos de los informes
- nombre de la base de datos que contiene la historia del código fuente de la aplicación

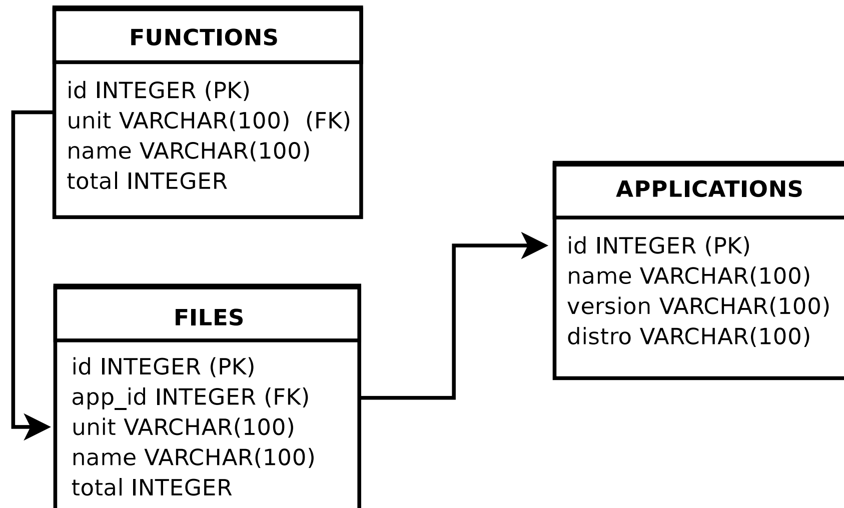

Figura 6.8: Base de datos de los informes de uso.

- fichero HTML de salida

El objeto *analyzer* obtiene la información de la base de datos de los informes de uso y tras ordenar las funciones por total de llamadas, vuelca el listado de las cien más utilizadas a un fichero HTML.

6.4.3. Obtener el listado de los ficheros más utilizados y sus desarrolladores

El caso de uso comienza cuando el usuario llama al analizador de actividad y uso y le pasa los siguientes parámetros:

- usuario de la base de datos
- password de la base de datos para el usuario antes incluido
- nombre de la base de datos que contiene los informes
- identificador de la aplicación a estudiar en la base de datos de los informes
- nombre de la base de datos que contiene la historia del código fuente de la aplicación
- fichero HTML de salida

El objeto *analyzer* consulta a la base de datos de los informes de uso para obtener el listado total de ficheros así como el total de llamadas que acumulan sus funciones. A continuación abre una nueva conexión a la base de datos de la historia del código fuente y para cada fichero consulta el número de commits, el número de committers y el nombre de éstos.

Tras obtener toda la información deseada de los ficheros, vuelca el listado a un fichero HTML detallando:

- número total de llamadas de las funciones del fichero
- nombre del fichero

- número de commits sobre el fichero
- número de desarrolladores que realizaron los commits
- nombre de los desarrolladores que han realizado commits sobre el fichero así como el número de éstos

6.4.4. Obtener el listado de los ficheros más utilizados y sus desarrolladores del último año

El caso de uso comienza cuando el usuario llama al analizador de actividad y uso y le pasa los siguientes parámetros:

- usuario de la base de datos
- password de la base de datos para el usuario antes incluido
- nombre de la base de datos que contiene los informes
- identificador de la aplicación a estudiar en la base de datos de los informes
- nombre de la base de datos que contiene la historia del código fuente de la aplicación
- fichero HTML de salida

El objeto *analyzer* consulta a la base de datos de los informes de uso para obtener el listado total de ficheros así como el total de llamadas que acumulan sus funciones. A continuación abre una nueva conexión a la base de datos de la historia del código fuente y para cada fichero y dentro del último año de su historia consulta el número de commits, el número de committers y el nombre de éstos.

Tras obtener toda la información deseada de los ficheros, vuelca el listado a un fichero HTML detallando:

- número total de llamadas de las funciones del fichero
- nombre del fichero
- número de commits sobre el fichero durante el año previo a su publicación
- número de desarrolladores que realizaron los commits durante el año previo a su publicación
- nombre de los desarrolladores que han realizado commits sobre el fichero así como el número de éstos, también durante el año previo a su publicación

6.5. Bibliotecas utilizadas por los scripts de análisis

6.5.1. MySQLdb

MySQLdb es una interfaz multihilo de la conocida base de datos MySQL. A continuación se explica brevemente su uso por parte de los scripts.

El paquete utilizado se invoca de la siguiente manera:

```
import MySQLdb
```

Una vez importado, conectarse a la base de datos se realiza llamando a la función *connect*:

```
connection=MySQLdb.connect(host="localhost",
                           user=dbuser,
                           passwd=dbpass,
                           db=cbidb)
```

Para finalizar, la manera de realizar una consulta a la base de datos se realiza mediante la función *execute* del objeto *cursor*.

```
self.cbi_cursor = connection.cursor()
self.cbi_cursor.execute("SELECT * FROM Functions")
rows = self.cbi_cursor.fetchall()
```

6.5.2. Aplicaciones de la biblioteca DOM

La biblioteca DOM ha sido utilizada para el análisis de los ficheros de sintaxis XML. El paquete utilizado se invoca de la la siguiente manera:

```
from xml.dom import minidom
```

Una vez importado el módulo, cargar y validar un documento XML en memoria se hace en un sólo paso:

```
xmldoc = minidom.parse('fichero.xml')
```

Ahora, para obtener el nodo marcado por la etiqueta deseada realizamos lo siguiente:

```
reflist= xmldoc.getElementsByTagName('etiqueta')
```

Obtenemos de esta manera una lista de objetos correspondiente a los elementos XML que tienen esa etiqueta. Para verla en formato cadena de texto bastaría con aplicarle la función *toxml()* a un elemento de la lista.

Capítulo 7

Estudio de los informes

Este capítulo muestra algunos resultados obtenidos en el estudio de los informes, además de información a tener en cuenta antes de su lectura. Esta información se compone de una estimación objetiva del error que los informes contienen debido a un error en la herramienta MSR y de las correcciones que se tuvieron que efectuar en los resultados por la dependencia de una librería de GNOME.

Para las aplicaciones de las que se cuenta con más de una versión y éstas están separadas en al menos un año, se estudiará por separado el año previo a la publicación de cada una y la historia completa, todo ello relacionado con los informes de uso obtenidos. Para el caso de varias versiones, como se mencionó en el capítulo anterior, se mostrará el estudio de la que más informes de uso haya producido.

7.1. Entorno real del estudio

Una de las dificultades del estudio ha sido la necesidad de establecer una colaboración por parte de usuarios reales. Esta colaboración ha sido ofrecida por un grupo de unos diez usuarios, la gran mayoría del entorno del grupo de investigación GSyC/LibreSoft, que han usado las aplicaciones instrumentadas para la distribución Ubuntu feisty y Debian etch.

Con el objetivo de facilitar que se unieran nuevos voluntarios al estudio mediante el uso de herramientas modificadas, se publicitó el proyecto a través de una web[pro07], en la que se ofrecieron instrucciones de cómo colaborar utilizando las herramientas y cuáles habían sido los resultados obtenidos hasta la fecha. Además, en ella se publicitaba la dirección del repositorio Debian/Ubuntu que se creó con el objetivo de automatizar la instalación por parte de los usuarios finales.

Además de la necesidad de publicitar el proyecto, fue necesario la instalación de un servidor web en una de las máquinas del grupo de investigación para recolectar los informes de uso. El número de informes recolectados en cuatro meses ascendió a más de cinco mil setecientos, de los cuales un 97 % fueron válidos.

La figura 7.1 viene a resumir los componentes y entorno del estudio, que se realizó en su gran mayoría durante el año 2007.

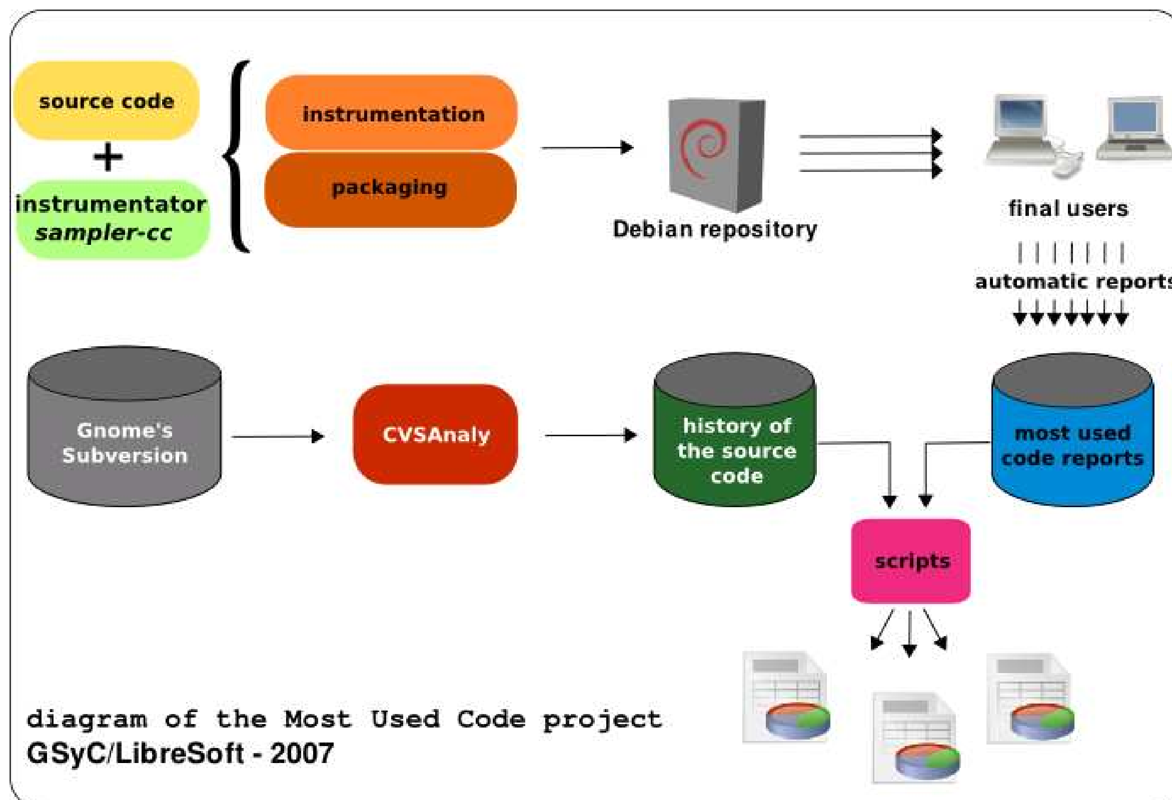

Figura 7.1: Componentes del proyecto MUCode

7.2. Tipo de estudio realizado

Como resultado de la colaboración de los usuarios en el entorno que se mostró en el apartado anterior, se obtuvieron más de cinco mil informes de uso que debían ser procesados y estudiados para poder ofrecer datos de interés. El estudio se compone de diversas etapas. La primera etapa es la encargada de obtener los informes de uso y relacionarlos con los sites, cuyo resultado será una base de datos con las funciones de las aplicaciones de las que se recibió informes y el número de llamadas que acumularon. Una vez se supo qué funciones eran las más utilizadas de cada aplicación, se pasó a identificar la funcionalidad de las que parecían más relevantes.

Tras haber estudiado las funciones de las aplicaciones, se pasó a estudiar la actividad en torno a los ficheros y el número de llamadas que acumulaban (calculándolo con las funciones de cada fichero). Una vez se obtuvieron los datos sobre el número de llamadas para cada fichero, se estudiaron los diez ficheros que más llamadas acumulaban y su número de commits y committers en relación a medidas halladas como media, mediana y coeficiente de variación. Para finalizar y partiendo del listado completo de ficheros con sus llamadas acumuladas y commits, se representó la relación entre informes y commits mediante una serie de gráficas.

7.3. Resumen de los datos ofrecidos por el estudio

Los siguientes apartados de este capítulo se centrarán en ofrecer los datos que han sido obtenidos tras estudiar los informes de uso obtenidos para cada aplicación y la historia de su código mediante la herramienta CVSAnalY. Con el objetivo de mostrar más claridad se

explicarán brevemente el tipo de datos que se van a ofrecer.

Para cada aplicación se mostrarán:

- un resumen esquemático de los informes obtenidos, detallando el número de éstos y la media de muestreo.
- las diez funciones más utilizadas en las ejecuciones recogidas por el estudio, comentando aquellas que por número de llamadas son más relevantes
- los diez ficheros más utilizados según sus funciones y sus datos cuantitativos sobre commits y committers
- medidas objetivas como media, mediana, desviación estándar y coeficiente de variación para todos los datos obtenidos, que se pueden consultar en los anexos E y D

7.4. Resultados sobre Eye of Gnome 2.18.1

7.4.1. Datos obtenidos

Aplicación	Informes	Media de muestreo	Total llamadas
Gedit 2.18.1	154	100 %	230316764

Cuadro 7.1: Datos recolectados para EOG 2.18.1

7.4.2. Funciones más utilizadas

La función más utilizada en EOG 2.18.1 según los informes es *eog_collection_item_get_type*, que devuelve el identificador del tipo del objeto registrado en el sistema de tipos de GLib. La segunda, *eog_collection_item_get_size* devuelve datos sobre el tamaño de la imagen, mientras que la tercera de nombre *copy_tile* interviene en el dibujado de la imagen. Según la media recogida en el capítulo E, todas las funciones que aparecen en el listado están por encima de la media de llamadas, que es de 11283,34. La función más utilizada lo es más de cien veces más que la media y el valor de la mediana es de 1.

Total llamadas	Función	Fichero
1215777	eog_collection_item_get_type	libeog/eog-collection-item.c
916772	eog_collection_item_get_size	libeog/eog-collection-item.c
722137	copy_tile	libeog/uta.c
458386	set_item_position	libeog/eog-wrap-list.c
378292	paint_background	libeog/eog-scroll-view.c
348346	eog_image_get_type	libeog/eog-image.c
281670	is_unity_zoom	libeog/eog-scroll-view.c
235557	eog_scroll_view_get_type	libeog/eog-scroll-view.c
213228	eog_job_get_type	libeog/eog-job.c
187453	compute_scaled_size	libeog/eog-scroll-view.c

Cuadro 7.2: 10 funciones más utilizadas en EOG 2.18.1

7.4.3. Actividad alrededor de los ficheros más utilizados en año previo a la publicación

Llamadas	Fichero	commits	committers	id committers(commits)
2469696	libeog/eog-collection-item.c	5	3	friemann(2), csaavedr(2), lucasr(1),
1674144	libeog/eog-scroll-view.c	4	3	csaavedr(1), friemann(1), lucasr(2),
1085389	libeog/uta.c	0	0	
679488	libeog/eog-wrap-list.c	6	3	lucasr(2), friemann(3), csaavedr(1),
601374	libeog/eog-image.c	16	4	csaavedra(1), lucasr(5), friemann(7), csaavedr(3),
423992	libeog/eog-job.c	2	1	csaavedr(2),
351413	libeog/eog-info-view-exif.c	1	1	friemann(1),
317361	libeog/eog-image-list.c	12	4	csaavedra(1), lucasr(3), csaavedr(5), friemann(3),
109711	libeog/eog-canvas-pixbuf.c	2	2	csaavedr(1), lucasr(1),
99852	shell/eog-window.c	42	7	csaavedra(1), lucasr(12), csaavedr(14), friemann(12), hendrikr(1), lferrett(1), serrador(1),

Cuadro 7.3: Actividad en el año previo a la publicación de EOG 2.18.1

En el año previo a la publicación, los diez ficheros que más llamadas a funciones acumulan en su versión *2.18.1* suman un total de 86 commits frente a una media de 4,11. Mientras que los ficheros más utilizados superan el millón de llamadas a funciones, el fichero *shell/eog-window.c*, que es claramente el más modificado de todos, se acerca a las cien mil (la media es de 221268).

Los desarrolladores más destacados de esos ficheros son *csaavedr* y *friemann* seguidos por *lucasr*.

Figura 7.2: Actividad en el año previo a la publicación de EOG 2.18.1

Como se puede ver en la figura 7.2, destacan dos ficheros sobre el resto. Primero, el fichero que más llamadas acumula; segundo, el fichero que más modificaciones ha sufrido en el año previo a la publicación. Mientras que el fichero con más llamadas a funciones se encuentra a menos de un commits de diferencia de la media, el fichero con más modificaciones se encuentra por debajo de la media de llamadas a funciones. Es decir, el fichero que más se ha utilizado es modifica poco, mientras que el fichero que más se ha modificado tiene la mitad de llamadas a funciones que la media. Además, los cuatro ficheros más utilizados se encuentran bajo el umbral de 8 commits.

7.4.4. Actividad alrededor de los ficheros más utilizados

A lo largo de la historia, como muestra la tabla 7.4, los diez ficheros más utilizados suman 447 commits, habiéndose ejecutado la mitad de éstos en el fichero *shell/eog-window.c*. En el análisis de la historia de éstos ficheros, aparece un desarrollador que no lo hace en el año previo, su identificador es *jens* y acumuló 170 commits; lo sigue *lucasr* con 104, que también es uno de los más activos en el último año antes de la publicación.

El resultado ofrecido por la figura 7.3 se asemeja al obtenido en el estudio del año previo a la publicación de la aplicación. Se observa que siete de los ocho ficheros que más se han utilizado no superan el umbral de 50 commits, estando la media en 20; mientras que unos de los ficheros que menos se ha utilizado supera ampliamente y en solitario los 200 commits.

Llamadas	Fichero	commits	committers	id committers(commits)
2469696	libeog/eog-collection-item.c	15	6	friemann(2), csaavedr(2), lucasr(3), timg(2), jens(5), jamesh(1),
1674144	libeog/eog-scroll-view.c	34	6	csaavedr(1), friemann(1), lucasr(11), timg(4), jens(16), federico(1),
1085389	libeog/uta.c	8	4	lucasr(1), jens(1), federico(5), darin(1),
679488	libeog/eog-wrap-list.c	19	4	lucasr(8), friemann(3), csaavedr(1), jens(7),
601374	libeog/eog-image.c	83	11	csaavedra(1), lucasr(14), friemann(7), csaavedr(3), pborelli(1), timg(4), pvanhoof(3), rburton(3), jens(45), vnoel(1), federico(1),
423992	libeog/eog-job.c	7	4	csaavedr(2), lucasr(3), timg(1), jens(1),
351413	libeog/eog-info-view-exif.c	11	5	friemann(1), lucasr(2), timg(2), jens(5), vnoel(1),
317361	libeog/eog-image-list.c	33	8	csaavedra(1), lucasr(11), csaavedr(5), friemann(3), ryanl(1), timg(2), jens(9), vnoel(1),
109711	libeog/eog-canvas-pixbuf.c	6	4	csaavedr(1), lucasr(3), timg(1), jens(1),
99852	shell/eog-window.c	231	23	csaavedra(1), lucasr(48), csaavedr(15), friemann(12), hendrikr(1), lferrett(1), serrador(1), timg(13), pvanhoof(2), pborelli(1), rburton(2), tml(2), jens(96), mariano(1), kmaraas(1), federico(10), murrayc(1), michael(2), seth(1), lutz(6), jberkman(10), mmeeks(1), martin(3),

Cuadro 7.4: Actividad hasta la publicación de EOG 2.18.1

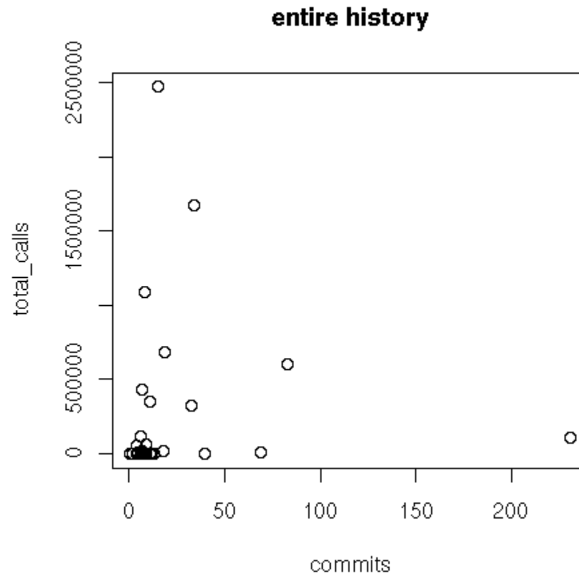

Figura 7.3: Actividad hasta la publicación de EOG 2.18.1

7.5. Resultados sobre Evince 0.4.0

7.5.1. Datos obtenidos

Aplicación	Informes	Media de muestreo	Total llamadas
Evince 0.4.0	149	85,2	10738578

Cuadro 7.5: Datos recolectados para Evince 0.4.0

7.5.2. Funciones más utilizadas

La media de llamadas a función es de 1618, mientras que la mediana es 6.

De las primeras veinte funciones en cuanto a llamadas acumuladas, quince formaban parte de la librería *libegg* que, como se comentó el principio de este capítulo, no será tenida en cuenta en el estudio.

Así pues, contando con los datos corregidos, se puede observar que de las tres primeras, *compute_border* se encarga de calcular el tamaño de el borde de la interfaz gráfica. La segunda, *save_values* se encarga de llamar a la función *xmlSetProp*, que forma parte de la biblioteca utilizada para el manejo de XML en GNOME. Finalmente la más utilizada, *ev_page_cache_get_type*, se utiliza de registrar los tipos en el sistema GLib y si el tipo está registrado devuelve el identificador del tipo. Esta función ha sido llamada unas unas 170 veces más que la media.

7.5.3. Actividad alrededor de los ficheros más utilizados en el último año

De igual manera que para las funciones más utilizadas, ha sido necesario eliminar del listado los dos primeros ficheros que más llamadas acumulaban, *cut-n-paste/recent-files/egg-recent-model.c* y *cut-n-paste/recent-files/egg-recent-item.c*.

Total llamadas	Función	Fichero
270886	ev_page_cache_get_type	shell/ev-page-cache.c
175046	save_values	shell/ev-metadata-manager.c
82972	compute_border	shell/ev-view.c
82586	ev_document_misc_get_page_border_size	backend/ev-document-misc.c
75985	ev_page_cache_get_size	shell/ev-page-cache.c
64905	ev_page_cache_get_max_width	shell/ev-page-cache.c
57586	ev_page_cache_get_n_pages	shell/ev-page-cache.c
52513	ev_link_get_type	backend/ev-link.c
43074	ev_job_get_type	shell/ev-jobs.c
41928	get_page_extents	shell/ev-view.c

Cuadro 7.6: 10 funciones más utilizadas en Evince 0.4.0

La historia de los ocho primeros meses de proyecto relacionada con los informes de uso deja un total de 423 commits para los 10 ficheros más utilizados, mientras que la media de commits es de 11,82. Destacan por su número de commits los desarrolladores identificados por *marco* y *jrb* con 194 y 92 commits respectivamente de los 423 que acumularon los ficheros mostrados en la tabla.

Es sensible la diferencia de commits entre los ficheros *shell/ev-window.c* y *shell/ev-view.c* y el resto de los más utilizados, ya que fueron los que más actividad tuvieron durante los primeros 8 meses de proyecto con 211 y 150 commits respectivamente.

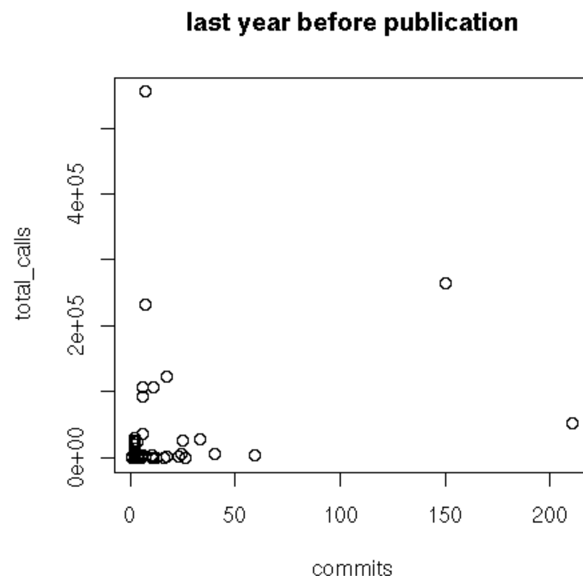

Figura 7.4: Actividad en el año previo a la publicación de Evince 0.4.0

En la figura 7.4 se aprecia que hay nueve ficheros que destacan del resto, estando tres de ellos totalmente separados. El fichero que más modificaciones acumula es más usado que la media, mientras que el fichero que más llamadas acumula tiene menos modificaciones que la media. De los siete ficheros que más llamadas acumulan, seis se encuentran bajo el umbral de los 50

Total Llamadas	Fichero	commits	committers	id committers(commits)
555415	shell/ev-page-cache.c	7	3	jrb(1), marco(5), nshmyrev(1),
265035	shell/ev-view.c	150	10	jrb(33), marco(74), nshmyrev(20), krh(6), jbowes(1), martink(5), dave_malcolm(1), hp(4), markmc(2), owen(4),
231558	shell/ev-metadata-manager.c	7	4	jrb(1), marco(3), chpe(2), martink(1),
123583	shell/ev-pixbuf-cache.c	17	5	krh(1), jrb(11), marco(3), msw(1), nshmyrev(1),
106820	shell/ev-jobs.c	11	4	marco(5), jrb(4), krh(1), nshmyrev(1),
105497	backend/ev-link.c	6	3	marco(3), jrb(2), martink(1),
92517	backend/ev-document-misc.c	6	2	jrb(5), marco(1),
51286	shell/ev-window.c	211	19	ryanl(1), jrb(34), nshmyrev(34), chpe(4), martink(10), marco(96), msw(1), aldug(1), carlosgc(2), jbowes(2), dave_malcolm(3), pborelli(1), krh(4), clarkbw(2), vnoel(1), hp(5), andersca(3), markmc(3), owen(4),
36342	backend/ev-document-find.c	6	3	krh(1), marco(4), hp(1),
30238	dvi/mdvi-lib/font.c	2	1	nshmyrev(2),

Cuadro 7.7: Actividad en el año previo a la publicación de Evince 0.4.0

commits; si bien es verdad que, a diferencia de lo que ocurre en otros estudios, el octavo más utilizado es el fichero más modificado con 211 commits.

7.6. Resultados sobre Evince 0.8.1

7.6.1. Datos obtenidos

Aplicación	Informes	Media de muestreo	Total llamadas
Evince 0.8.1	3611	97,9	380835188

Cuadro 7.8: Datos recolectados para Evince 0.8.1

7.6.2. Funciones más utilizadas

A diferencia de lo que pasaba en la versión *0.4.0*, las funciones de la biblioteca *libegg* no aparecen en las primeras posiciones del listado de funciones más utilizadas. De hecho hay que irse hasta la posición sesenta y cinco para ver la primera, pero que en esta caso es *egg_editable_toolbar_get_type*.

De las tres primeras, sólo cambia con respecto a la versión anterior *compare_recent_items*, que interviene en la selección que se realizada entre los últimos ficheros abiertos por cualquier aplicación GNOME para elegir los que serán manejados por Evince. Destacado papel tienen *ev_page_cache_get_type* y *compute_border* que en el código de casi dos años después sigo siendo la primera y tercera más utilizada.

Como dato curioso, en la posición novena podemos ver a la función *get_page_y_offset*, que se utiliza para la funcionalidad de leer un documento de manera continua sin saltos entre página y página. Esto significa que la gran mayoría de usuarios han preferido este modo al que enseña las páginas de una en una. Además, la función más utilizada pasa ahora a ejecutarse 275 veces más que la media.

Total llamadas	Función	Fichero
69834137	ev_page_cache_get_type	shell/ev-page-cache.c
60553497	compare_recent_items	shell/ev-window.c
25678716	compute_border	shell/ev-view.c
25660651	ev_document_misc_get_page_border_size	libdocument/ev-document-misc.c
25568268	ev_page_cache_get_max_width	shell/ev-page-cache.c
16695438	ev_page_cache_get_size	shell/ev-page-cache.c
15619307	ev_document_find_get_type	libdocument/ev-document-find.c
12857535	get_page_extents	shell/ev-view.c
12767569	ev_page_cache_get_height_to_page	shell/ev-page-cache.c
12767569	get_page_y_offset	shell/ev-view.c
12475067	ev_page_cache_get_n_pages	shell/ev-page-cache.c

Cuadro 7.9: 10 funciones más utilizadas para Evince 0.8.1

Llamadas	Fichero	commits	committers	id committers(commits)
141611228	shell/ev-page-cache.c	7	2	carlosgc(3), nshmyrev(4),
72249434	shell/ev-view.c	51	4	carlosgc(30), nshmyrev(14), wbolster(6), serrador(1),
64251387	shell/ev-window.c	100	6	nshmyrev(39), matthiasc(2), carlosgc(48), wbolster(8), julienr(2), csaavedr(1),
26257714	libdocument/ev-document-misc.c	1	1	carlosgc(1),
15918803	shell/ev-metadata-manager.c	2	1	nshmyrev(2),
15773498	shell/ev-pixbuf-cache.c	4	2	carlosgc(3), nshmyrev(1),
3338802	backend/ps/ps.c	2	1	carlosgc(2),
1326859	shell/ev-jobs.c	19	2	carlosgc(14), nshmyrev(5),
539789	libdocument/ev-link.c	1	1	nshmyrev(1),
486972	shell/ev-job-queue.c	2	1	carlosgc(2),

Cuadro 7.10: Actividad en el año previo a la publicación de Evince 0.8.1

Figura 7.5: Actividad en el año previo a la publicación de Evince 0.8.1

7.6.3. Actividad alrededor de los ficheros más utilizados en el último año

A diferencia de los datos obtenidos para su versión *0.4.0*, los que eran desarrolladores más activos de los ficheros más utilizados, *marco* y *jrb*, no vuelven a aparecer en actividad alguna relacionada con el código fuente del proyecto. Su testigo en este caso ha sido cogido por *carlosgc* y *nshmyrev* con 103 y 66 commits respectivamente. Entre ambos suman casi el total de modificaciones sobre esos diez ficheros, que asciende a 189 commits y que deja la media de modificaciones en este periodo en 6,34.

Al igual que en la versión *0.4.0*, los dos ficheros que más se han modificado han sido *shell/ev-window.c* y *shell/ev-view.c*, con 100 y 51 commits sobre los 190 de los ficheros de la tabla. En esta versión además se encuentran entre los tres que más se han utilizado.

Es reseñable que el fichero que más se ha utilizado y que se encarga de la gestión de la caché de documentos sólo ha sido modificado en el último año en siete ocasiones y que sólo lo han modificado los dos desarrolladores que más peso tienen en los ficheros más utilizados.

Si observamos la figura 7.5 podemos ver que los ficheros que en la versión *0.4.0* eran los más modificados, siguen siéndolo y además se han utilizado más. De igual manera podemos ver que el fichero que más se utiliza sigue estando en el grupo de los que menos se modifican estando menos de un commit por encima de la media.

7.6.4. Actividad alrededor de los ficheros más utilizados

Los datos referentes a la actividad de desarrollo sobre el código y a los informes de uso de la versión *0.8.1* destacan a dos grupos de desarrolladores que se identificaron tanto en el año previo a la publicación de la versión *0.4.0* como de la *0.8.1*.

El cálculo, esta vez realizado sobre los veintiocho meses de la vida del código fuente, da un total de 713 commits sobre los diez ficheros que más llamadas a funciones acumulan para la versión *0.8.1*. Los desarrolladores más activos sobre éstos ficheros (*carlosgc*,*nshmyrev*, *marco* y

Llamadas	Fichero	commits	committers	id committers(commits)
141611228	shell/ev-page-cache.c	19	4	carlosgc(3), nshmyrev(7), marco(8), jrb(1),
72249434	shell/ev-view.c	247	14	carlosgc(32), nshmyrev(59), wbolster(6), serrador(1), marco(91), chpe(1), jrb(34), krh(6), jbowes(1), martink(5), dave_malcolm(1), hp(4), markmc(2), owen(4),
64251387	shell/ev-window.c	368	24	nshmyrev(107), matthiasc(2), carlosgc(51), wbolster(8), julienr(2), csaavedr(1), chpe(6), marco(115), caillon(1), ryanl(1), jrb(34), martink(10), msw(1), aldug(1), jbowes(2), dave_malcolm(3), pborelli(1), krh(4), clarkbw(2), vnoel(1), hp(5), andersca(3), markmc(3), owen(4),
26257714	libdocument/ev-document-misc.c	1	1	carlosgc(1),
15918803	shell/ev-metadata-manager.c	14	6	nshmyrev(6), carlosgc(1), jrb(1), marco(3), chpe(2), martink(1),
15773498	shell/ev-pixbuf-cache.c	25	6	carlosgc(3), nshmyrev(5), marco(4), krh(1), jrb(11), msw(1),
3338802	backend/ps/ps.c	2	1	carlosgc(2),
1326859	shell/ev-jobs.c	31	5	carlosgc(14), nshmyrev(7), marco(5), jrb(4), krh(1),
539789	libdocument/ev-link.c	1	1	nshmyrev(1),
486972	shell/ev-job-queue.c	5	3	carlosgc(2), nshmyrev(2), marco(1),

Cuadro 7.11: Actividad hasta la publicación de Evince 0.8.1

Figura 7.6: Actividad hasta la publicación de Evince 0.8.1

jrb) acumulan un total de 615 commits, o lo que es lo mismo un 86,2 % de los commits.

La media de modificaciones para la vida del proyecto es de 21,82 commits.

Al igual que en el estudio del año previo a la publicación de las versiones *0.4.0* y *0.8.1*, la figura 7.6 muestra tres grupos según la clasificación obtenida por modificación y utilización del código. Los grupos se puede identificar por:

- la mayoría de los ficheros, que reciben modificaciones esporádicas
- un grupo muy pequeño de ficheros, dos en este caso, que son con diferencia los que más se modifican y que se utilizan más que los del primer grupo
- el fichero *shell/ev-page-cache.c* que es el que más intensamente se utiliza y que se modifica con baja frecuencia igual que los del primer grupo.

7.7. Resultados sobre Evolution 2.6.3

7.7.1. Datos obtenidos

Aplicación	Informes	Media de muestreo	Total llamadas
Evolution 2.6.3	99	83,1 %	128689375

Cuadro 7.12: Datos recolectados para Evolution 2.6.3

7.7.2. Funciones más utilizadas

Se observa que existe un grupo de cuatro funciones que se diferencia ligeramente del resto en cuando a número de llamadas acumuladas. De ellas la décima es 238 mayor que la media, mientras que la primera lo es 658 veces. Esto, unido a una mediana de valor 0, constata que hay una enorme cantidad de código que no se está utilizando, debido claro al enorme tamaño de la aplicación Evolution.

Las dos primeras funciones de la lista, *e_sort_callback* y *qsort_callback* contienen código que realiza operaciones muy básicas que son utilizadas por otras funciones para realizar búsquedas y ordenaciones más complejas. La tercera se utiliza para comparación de un tipo utilizado por evolution y la cuarta, *e_tree_model_get_type* devuelve el identificador del tipo de objeto registrado en el sistema de tipos de GLib, lo que se utiliza para no registrar más de una vez un mismo objeto. El alto número de llamadas de esta última se debe a que se utiliza en cada conversión explícita de tipos.

7.7.3. Actividad alrededor de los ficheros más utilizados en el último año

Es muy llamativo que, a diferencia de lo que pasa con otras aplicaciones, la actividad sobre los diez ficheros más utilizados para *Evolution 2.6.3* es prácticamente nula, acumulando tan sólo 12 commits repartidos en tres ficheros, mientras que la media es de 2,66 modificaciones.

Hay que remontarse casi al puesto cien de esta lista para encontrar al fichero que más modificaciones tiene, que es *calendar/gui/e-calendar-view.c* con 29 commits. Para encontrar un fichero que supere el umbral de los diez commits hay que irse al puesto trigésimo quinto.

Total llamadas	Función	Fichero
6948054	e_sort_callback	widgets/table/e-table-sorting-utils.c
6947867	qsort_callback	e-util/e-util.c
6947276	e_int_compare	e-util/e-util.c
6590600	e_tree_model_get_type	widgets/table/e-tree-model.c
5911654	e_tree_memory_get_type	widgets/table/e-tree-memory.c
4534263	e_cell_text_get_type	widgets/table/e-cell-text.c
3452392	e_table_model_get_type	widgets/table/e-table-model.c
2873300	e_tree_memory_node_get_data	widgets/table/e-tree-memory.c
2517349	e_table_model_value_at	widgets/table/e-table-model.c
2517173	etta_value_at	widgets/table/e-tree-table-adapter.c

Cuadro 7.13: 10 funciones más utilizadas para Evolution 2.6.3

Llamadas	Fichero	commits	committers	id committers(commits)
24028725	widgets/table/e-tree-memory.c	0	0	
16344690	widgets/table/e-tree-model.c	0	0	
14261690	e-util/e-util.c	6	5	haip(1), tml(2), kharish(1), rsushma(1), zucchi(1),
11796595	widgets/table/e-tree-table-adapter.c	3	3	kmaraas(1), sragavan(1), tml(1),
9538346	widgets/table/e-cell-text.c	3	3	simonz(1), kmaraas(1), pchen(1),
6961579	widgets/table/e-table-sorting-utils.c	0	0	
6740270	widgets/table/e-table-model.c	0	0	
4766905	widgets/table/e-table-header.c	0	0	
4299916	widgets/table/e-tree-memory-callbacks.c	0	0	
3514575	mail/e-searching-tokenizer.c	0	0	

Cuadro 7.14: Actividad en el año previo a la publicación de Evolution 2.6.3

Figura 7.7: Actividad en el año previo a la publicación de Evolution 2.6.3

Pese a lo extraño del número de commits en el año previo a la publicación, la figura 7.7 sí refleja que los ficheros que más se modifican son los que menos llamadas a ficheros acumulan. Por el contrario, de los diez que más llamadas acumulan, sólo uno supera el umbral de los 5 commits.

7.8. Resultados sobre Evolution 2.10.1

7.8.1. Datos obtenidos

Aplicación	Informes	Media de muestreo	Total llamadas
Evolution 2.10.1	417	100 %	5351244961

Cuadro 7.15: Datos recolectados para Evolution 2.10.1

7.8.2. Funciones más utilizadas

Es significativo que la lista de las diez funciones más utilizadas en *Evolution 2.6.3* y *2.10.1* sea tan parecida. De hecho las cuatro funciones más utilizadas son las mismas, aunque en orden distinto. Es más, las cuatro funciones se alejan ligeramente de la misma función que en la versión anterior, de la función que ocupa el quinto lugar cuyo nombre es *e_tree_memory_get_type*. Pero aún hay más, en la versión *2.6.3* la primera función era llamada 658 veces más que la media; pues bien, ahora lo es 860 veces más que la media y la mediana sigue siendo de 0.

7.8.3. Actividad alrededor de los ficheros más utilizados en el último año

Durante el año previo a la publicación, la media de commits es de 2,25, cifra aún menor que la de la versión pasada.

Al igual que en los resultados ofrecidos por los informes de uso, la actividad alrededor de los ficheros es parecida. A simple vista se observa que los diez ficheros más utilizados coinciden en su mayoría con los de la versión previa, pero no sólo eso. Los dos ficheros más utilizados, *widgets/table/e-tree-memory.c* y *widgets/table/e-tree-model.c* que en la versión anterior acumularon 0 commits durante el año previo a la publicación, acumulan sólo 1 commit para esta. De

Total llamadas	Función	Fichero
370118708	e_tree_model_get_type	widgets/table/e-tree-model.c
345627999	qsort_callback	e-util/e-util.c
345165767	e_sort_callback	widgets/table/e-table-sorting-utils.c
344231406	e_int_compare	e-util/e-util.c
292757027	e_tree_memory_get_type	widgets/table/e-tree-memory.c
145202707	etmm_get_next	widgets/table/e-tree-memory.c
145202707	e_tree_model_node_get_next	widgets/table/e-tree-model.c
142012433	check_children	widgets/table/e-tree-memory.c
137890479	e_cell_text_get_type	widgets/table/e-cell-text.c

Cuadro 7.16: 10 funciones más utilizadas para Evolution 2.10.1

Llamadas	Fichero	commits	committers	id committers(commits)
1183743609	widgets/table/e-tree-memory.c	1	1	kharish(1),
870832311	widgets/table/e-tree-model.c	0	0	
710142784	e-util/e-util.c	9	5	mbarnes(1), sragavan(4), kmaraas(1), kharish(1), tml(2),
493804580	widgets/table/e-tree-table-adapter.c	3	3	mbarnes(1), sragavan(1), kharish(1),
346680509	widgets/table/e-table-sorting-utils.c	0	0	
289028663	widgets/table/e-cell-text.c	15	6	mbarnes(3), kmaraas(1), sragavan(6), kharish(3), hiikezoe(1), simonz(1),
199053714	widgets/table/e-table-model.c	0	0	
179010766	widgets/table/e-tree-memory-callbacks.c	0	0	
129162415	mail/message-list.c	19	6	mbarnes(5), sragavan(9), kmaraas(1), kharish(2), aklapper(1), liyuan(1),
107915204	filter/filter-option.c	1	1	mbarnes(1),

Cuadro 7.17: Actividad en el año previo a la publicación de Evolution 2.10.1

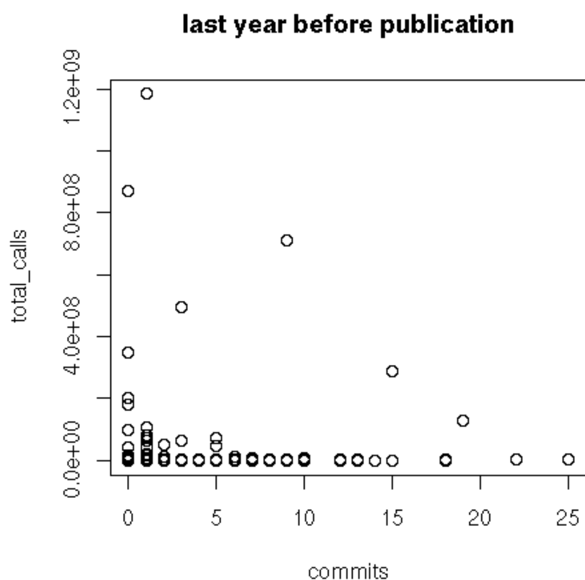

Figura 7.8: Actividad en el año previo a la publicación de Evolution 2.10.1

igual manera, hay otro conjunto de ficheros que estando en la lista de los más utilizados tienen un total de 0 commits durante el año previo a ambas publicaciones.

Existe una diferencia apreciable con los datos de la versión *2.6.3* en lo ficheros *widgets/table/e-cell-text.c* y *mail/message-list.c* que ahora acumulan más commits de diferencia con respecto al resto.

Estos ficheros acumulan en el período de un año 48 commits, de lo que se obtiene un papel destacado de los desarrolladores *sragavan* y *mbarnes* con 20 y 11 commits respectivamente. El primero de ellos ya se encontraba entre los committers de los ficheros más utilizados de la versión *2.6.3*.

Uno de los ficheros hace que la figura 7.8 se aleje de lo habitual, se trata del fichero *e-util/e-util.c* que casi alcanza el umbral de los 10 commits siendo el tercero más utilizado. De los ocho ficheros que acumulan más llamadas a funciones, seis están por debajo de 5 commits, uno se aproxima a 10 y otro supera los 15. Si apartamos esos tres ficheros, el resto de los ficheros con más modificaciones acumulan el menor número de llamadas a funciones, de igual manera que los más utilizados (salvo los tres antes comentados) rondan el umbral de 5 commits.

7.8.4. Actividad alrededor de los ficheros más utilizados

Llamadas	Fichero	commits	committers	id committers(commits)
1183743609	widgets/table/e-tree-memory.c	19	7	kharish(1), kaushal(1), tml(1), zucchi(1), mkestner(2), clahey(12), kmaraas(1),

870832311	widgets/table/e-tree-model.c	60	14	kaushal(1), tml(1), mkestner(4), clahey(22), mmeeks(1), kmaraas(1), toshok(17), miguel(3), jpr(1), zucchi(3), arios(1), federico(1), ettore(1), danw(3),
710142784	e-util/e-util.c	79	21	mbarnes(1), sragavan(4), kmaraas(1), kharish(2), tml(5), haip(1), rsushma(1), zucchi(1), kaushal(1), kcsuresh(1), mkestner(5), rodo(1), toshok(3), fejj(3), clahey(33), ettore(6), rhult(1), hallski(1), rconover(2), trow(1), danw(5),
493804580	widgets/table/e-tree-table-adapter.c	64	16	mbarnes(1), sragavan(2), kharish(1), kmaraas(2), tml(2), kaushal(1), haip(1), mkestner(19), hansp(1), toshok(1), zucchi(1), ettore(1), fejj(3), clahey(26), danw(1), mmeeks(1),
346680509	widgets/table/e-table-sorting-utils.c	8	4	kaushal(1), tml(1), clahey(5), jirka(1),
289028663	widgets/table/e-cell-text.c	160	29	mbarnes(3), kmaraas(3), sragavan(6), kharish(3), hiikezoe(1), simonz(2), pchen(1), kaushal(2), tml(1), zucchi(1), haip(1), rodo(2), fejj(1), gilbertfang(2), kcsuresh(2), hansp(1), mkestner(11), toshok(9), ettore(1), clahey(62), damon(3), trow(2), danw(1), federico(2), iain(7), miguel(10), peterw(1), lauris(3), unammx(16),

199053714	widgets/table/e-table-model.c	42	10	kaushal(1), tml(1), dobey(1), mkestner(2), clahey(17), mmeeks(1), miguel(5), zucchi(1), federico(1), unammx(12),
179010766	widgets/table/e-tree-memory-callbacks.c	8	4	kaushal(1), tml(1), mkestner(2), clahey(4),
129162415	mail/message-list.c	444	35	mbarnes(5), sragavan(11), kmaraas(4), kharish(2), aklapper(1), liyuan(1), simonz(1), kbrae(1), tml(2), kaushal(2), zucchi(103), fejj(99), haip(1), jpr(2), dobey(1), rodo(4), danw(62), katzj(1), ettore(10), peterw(14), clahey(70), trow(4), chyla(1), jleach(7), sammy(1), iain(3), miguel(5), mmeeks(2), toshok(5), mloper(3), jwise(1), federico(1), lewing(1), bertrand(8), unammx(5),
107915204	filter/filter-option.c	31	9	mbarnes(1), zucchi(10), fejj(8), ettore(3), rodo(1), danw(3), kmaraas(2), lauris(2), clahey(1),
95478618	widgets/table/e-table-header.c	54	11	kaushal(1), tml(1), toshok(1), mkestner(3), clahey(27), mmeeks(1), federico(1), danw(2), miguel(6), iain(1), unammx(10),

Cuadro 7.18: Actividad hasta la publicación de Evolution 2.10.1

El número de commits sobre los diez ficheros más utilizados es de 915 commits. El fichero que más commits acumula se encuentra en el puesto 51 de la lista de 521 ficheros y su nombre es *composer/e-msg-composer.c*. La media de commits para los ficheros es de 36,82.

Hay un usuario que destaca claramente sobre los diez ficheros que acumularon más llamadas, su identificado es *clahey* y ha realizado 279 de los 951 commits. Tras él un grupo de tres desarrolladores: *zucchi*, *fejj* y *danw* con 121, 114 y 77 commits respectivamente. Cabe destacar que ninguno de ellos ha aparecido entre los committers de los ficheros más utilizados de las dos

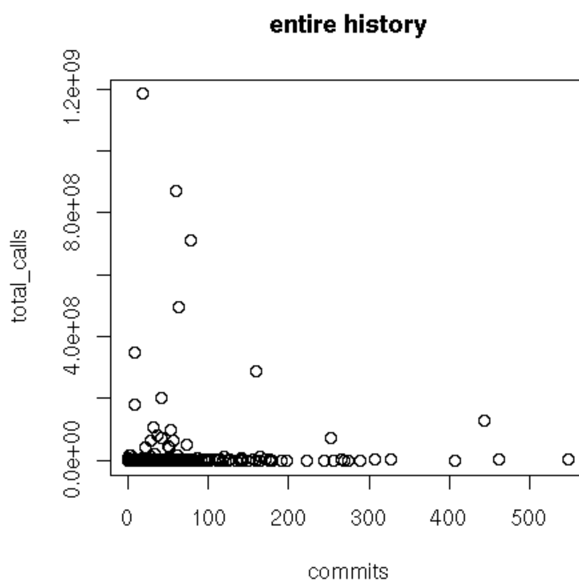

Figura 7.9: Actividad hasta la publicación de Evolution 2.10.1

versiones estudiadas de evolution. Además *zucchi* y *fejj* aparecen en el desarrollo previo a la publicación del *2.6.3*, pero sólo *fejj* continuó en la última versión.

Según la figura 7.9 que relaciona la historia completa de *Evolution* y los informes de uso de su última versión, siete de los ocho ficheros que más llamadas acumularon se encuentran bajo el umbral de los 100 commits. De igual manera se observa que los ficheros que más modificaciones tienen son por regla general los que se utilizan menos.

7.9. Resultados sobre Gedit 2.14.4

7.9.1. Datos obtenidos

Aplicación	Informes	Media de muestreo	Total llamadas
Gedit 2.14.4	83	96,7 %	68776487

Cuadro 7.19: Datos recolectados para Gedit 2.14.4

7.9.2. Funciones más utilizadas

La función *gedit_document_get_type* interviene en la identificación de tipos que realiza GLib en tiempo de ejecución, es con diferencia la que más llamadas acumula, repitiendo el comportamiento que se repite en la implementación de *EOG 2.18.1* con la función *eog_collection_item_get_type*.

La función más utilizada, lo es 330 veces más que la media de llamadas, que es 7113,12.

Como dato curioso, aparece la función *gedit_debug* que no cumple función alguna a no ser que estén activadas las opciones de DEBUG en la compilación, lo que ofrece una función que no ejecuta código alguno en la ejecución normal.

Total llamadas	Función	Fichero
2360860	gedit_document_get_type	gedit/gedit-document.c
1064885	gedit_document_mark_set	gedit/gedit-document.c
838264	gedit_debug	gedit/gedit-debug.c
834638	find_nearest_subregion	gedit/gedittextregion.c
564611	gedit_window_get_type	gedit/gedit-window.c
408641	gedit_tab_get_type	gedit/gedit-tab.c
355257	gedit_view_get_type	gedit/gedit-view.c
352182	gedit_tab_get_view	gedit/gedit-tab.c
350040	gedit_window_get_active_view	gedit/gedit-window.c
331867	gedit_statusbar_get_type	gedit/gedit-statusbar.c

Cuadro 7.20: 10 funciones más utilizadas en Gedit 2.14.4

7.9.3. Actividad alrededor de los ficheros más utilizados en el último año

Respecto a la actividad alrededor de los ficheros más utilizados llama poderosamente la atención que en este caso, dos de los tres ficheros más utilizados son también los más modificados. Se trata de los ficheros *gedit/gedit-document.c* y *gedit/gedit-window.c* y suman casi la mitad de los commits realizados sobre los diez ficheros más utilizados, que hacen un total de 63 commits.

Seis de los diez ficheros más utilizados están por debajo de la media de modificaciones que es de 3,17 commits.

Destaca el desarrollador *pborelli* al haber realizado el año previo a la publicación un 70 % de los commits sobre los diez ficheros más utilizados.

De la figura 7.10 se extrae la conclusión de que esta aplicación no va a seguir el patrón ni de Evince, EOG o Evolution. El fichero más modificado es con diferencia el más utilizado, además el segundo más modificado es el tercero más utilizado.

7.10. Resultados sobre Gedit 2.18.1

7.10.1. Datos obtenidos

7.10.2. Funciones más utilizadas

Las diez funciones más utilizadas de la versión *2.18.1* de Gedit coinciden casi por completo con las de la versión *2.14.4*. Este comportamiento ha sido observado igualmente en Evolution. El único cambio que aparece es que la función *gedit_debug* antes tercera pasa a ser segunda en detrimento de *gedit_document_mark_set*.

La proporción entre la función más utilizada y la media es muy parecida a la de la versión anterior estando esta en 307 veces superior a la media.

7.10.3. Actividad alrededor de los ficheros más utilizados en el último año

Al igual que en la versión anterior, el desarrollador más destacado de la aplicación en este último intervalo sigue siendo *pborelli*, que suma 33 de 50 commits sobre los diez ficheros más usados.

Llamadas	Fichero	commits	committers	id committers(commits)
4613026	gedit/gedit-document.c	17	3	pborelli(11), jrl(1), paolo(5),
1772423	gedit/gedittextregion.c	1	1	pborelli(1),
1475405	gedit/gedit-window.c	16	2	paolo(3), pborelli(13),
886469	gedit/gedit-debug.c	2	2	paolo(1), pborelli(1),
852741	gedit/gedit-tab.c	9	3	pborelli(6), paolo(2), farnold(1),
758027	gedit/gedit-view.c	10	2	pborelli(6), paolo(4),
499164	gedit/gedit-statusbar.c	1	1	pborelli(1),
146242	gedit/gedit-tooltips.c	2	2	pborelli(1), paolo(1),
102234	gedit/gedit-metadata-manager.c	3	2	pborelli(2), nshmyrev(1),
45261	gedit/gedit-plugin.c	2	1	pborelli(2),

Cuadro 7.21: Actividad en el año previo a la publicación de Gedit 2.14.4

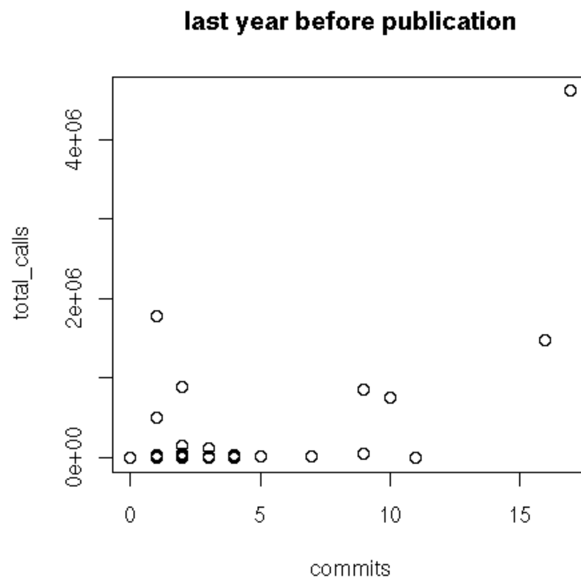

Figura 7.10: Actividad en el año previo a la publicación de Gedit 2.14.4

Aplicación	Informes	Media de muestreo	Total llamadas
Gedit 2.18.1	84	100 %	2926567

Cuadro 7.22: Datos recolectados para Gedit 2.18.1

Total llamadas	Función	Fichero
527375	gedit_document_get_type	gedit/gedit-document.c
235377	gedit_debug	gedit/gedit-debug.c
219372	gedit_document_mark_set	gedit/gedit-document.c
188822	find_nearest_subregion	gedit/gedittextregion.c
153006	gedit_window_get_type	gedit/gedit-window.c
105351	gedit_tab_get_type	gedit/gedit-tab.c
92921	gedit_view_get_type	gedit/gedit-view.c
91105	gedit_tab_get_view	gedit/gedit-tab.c
90994	gedit_window_get_active_view	gedit/gedit-window.c
84780	gedit_statusbar_get_type	gedit/gedit-statusbar.c

Cuadro 7.23: 10 funciones más utilizadas en Gedit 2.18.1

En este intervalo de tiempo la media de commits ha sido de 3,59 commits, medio commit más que para el año previo a la versión *2.14.4*.

A diferencia de lo que ocurría en la versión *2.14.4*, el comportamiento de los committers sobre el fichero más utilizado *gedit/gedit-document.c* se asemeja más en este caso a lo observador con otras aplicaciones. Como se puede observar en la figura 7.11, de los siete ficheros más utilizados cinco se mantienen bajo el umbral de 10 commits y es uno el que destacadamente se muestra como el más modificado.

7.10.4. Actividad alrededor de los ficheros más utilizados

Sobre los diez ficheros más utilizados se han producido un total de 346 commits, de los que el committer más activo en el año previo a las dos publicaciones, *pborelli*, suma 109. Sin embargo el desarrollador más destacado en la historia de éstos ficheros es el identificado por *paolo* con 191 commits.

La media de modificaciones acumulada para los ficheros de Gedit es de 17,42.

Pese a la tendencia de la figura 7.11 el fichero más utilizado sigue siendo claramente el más modificado, aunque como se ve en el año previo a la publicación de *Gedit 2.18.1*, parece que esa tendencia de modificaciones está estancándose ya que el fichero pasó de 17 commits a 7 de la última versión, mientras que la media rondó en ambos intervalos el 3,5.

Llamadas	Fichero	commits	committers	id committers(commits)
1017503	gedit/gedit-document.c	7	3	pborelli(3), sfre(1), paolo(3),
434519	gedit/gedit-window.c	28	4	pborelli(16), sfre(4), jessevdk(2), paolo(6),
388969	gedit/gedittextregion.c	0	0	
269125	gedit/gedit-debug.c	0	0	
224911	gedit/gedit-tab.c	11	3	pborelli(7), paolo(2), sfre(2),
185602	gedit/gedit-view.c	6	3	pborelli(4), sfre(1), paolo(1),
128116	gedit/gedit-statusbar.c	1	1	pborelli(1),
48346	gedit/gedit-metadata-manager.c	0	0	
35175	gedit/gedit-tooltips.c	0	0	
21269	gedit/gedit-prefs-manager.c	6	3	pborelli(2), sfre(3), paolo(1),

Cuadro 7.24: Actividad en el año previo a la publicación de Gedit 2.18.1

Figura 7.11: Actividad en el año previo a la publicación de Gedit 2.18.1

Llamadas	Fichero	commits	committers	id committers(commits)
1017503	gedit/gedit-document.c	137	10	pborelli(34), sfre(1), paolo(86), jrl(1), bdejean(1), jwillcox(5), federico(5), kmaraas(1), carlos(1), jleach(2),
434519	gedit/gedit-window.c	48	6	pborelli(29), sfre(4), jessevdk(2), paolo(9), jleach(3), chema(1),
388969	gedit/gedittextregion.c	1	1	pborelli(1),
269125	gedit/gedit-debug.c	12	3	pborelli(4), paolo(7), carlos(1),
224911	gedit/gedit-tab.c	20	4	pborelli(13), paolo(4), sfre(2), farnold(1),
185602	gedit/gedit-view.c	76	8	pborelli(14), sfre(1), paolo(56), asobala(1), kmaraas(1), jwillcox(1), chema(1), carlos(1),
128116	gedit/gedit-statusbar.c	2	1	pborelli(2),
48346	gedit/gedit-metadata-manager.c	8	4	pborelli(5), nshmyrev(1), marco(1), paolo(1),
35175	gedit/gedit-tooltips.c	5	3	pborelli(1), paolo(3), andersca(1),
21269	gedit/gedit-prefs-manager.c	37	5	pborelli(6), sfre(3), paolo(26), fcrozat(1), jwillcox(1),

Cuadro 7.25: Actividad hasta la publicación de Gedit 2.18.1

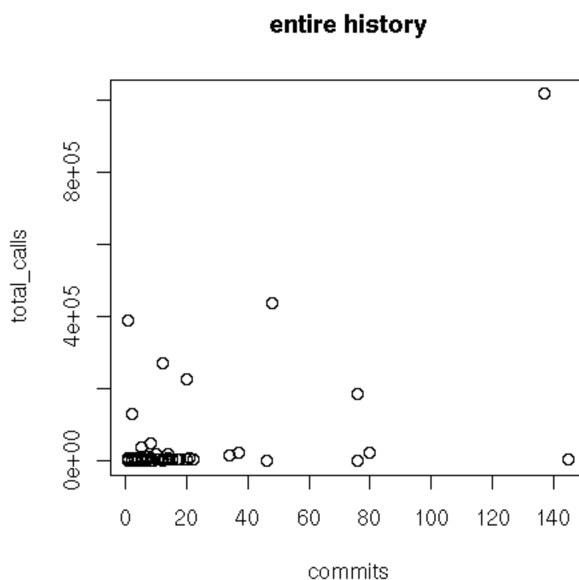

Figura 7.12: Gráfica de uso contra actividad sobre Gedit 2.18.1

7.11. Resultados sobre Gnome Terminal 2.14.2

Aplicación	Informes	Media de muestreo	Total llamadas
GNOME Terminal 2.14.2	420	82,27 %	256958

Cuadro 7.26: Datos recolectados para GNOME Terminal 2.14.2

7.11.1. Funciones más utilizadas

Al tratarse GNOME Terminal de una aplicación de sólo diecinueve ficheros, no es extraño que las diez primeras funciones se repartan en sólo tres ficheros. Es más, las tres funciones que más llamadas acumulan se encuentran en el mismo fichero.

De las diez funciones de la tabla, podemos destacar por funcionalidad dos grupos, el grupo de funciones que se encargan de gestionar los eventos recibidos por el teclado y el grupo de funciones que se encargan de gestionar la actividad del terminal.

Las dos funciones del primer grupo, *key_press_callback* y *accel_event_key_match*, se encargan respectivamente de realizar comprobaciones para cada evento generado por una tecla y de comparar la entrada recibida con las combinaciones de teclas previamente definidas. La primera función llama en ocasiones a la segunda dependiendo del número de pestañas que tenga el terminal.

El segundo grupo, que contiene el resto de funciones del listado, se encarga entre otras cosas de gestionar las copias de texto entre el terminal y otras aplicaciones del escritorio, para lo cual debe comprobar que la aplicación tenga el foco, es decir que esté siendo usada en ese momento por el usuario y no en segundo plano bajo otra ventana.

Total llamadas	Función	Fichero
57080	key_press_callback	src/terminal-window.c
18365	accel_event_key_match	src/terminal-window.c
9926	update_copy_sensitivity	src/terminal-window.c
9868	terminal_widget_get_has_selection	src/terminal-widget-vte.c
9842	terminal_screen_get_text_selected	src/terminal-screen.c
9830	terminal_screen_get_type	src/terminal-screen.c
9503	terminal_screen_widget_selection_changed	src/terminal-screen.c
9456	selection_changed_callback	src/terminal-window.c
6804	terminal_profile_get_type	src/terminal-profile.c
5829	window_state_event_callback	src/terminal-window.c

Cuadro 7.27: 10 funciones más utilizadas para GNOME Terminal 2.14.2

7.11.2. Actividad alrededor de los ficheros más utilizados en el último año

La media de llamadas acumuladas por los ficheros es de 13524, mientras que la media de commits a fichero durante el último año es de 5,21.

La actividad alrededor de los ficheros con más llamadas a funciones revela que los nueve últimos ficheros no acumulan una sola llamada. Es decir, los informes revelan que el uso de las funciones se ha ceñido a las que se encuentran en los diez ficheros mostrados en la lista, dejando sin uso nueve.

El número de commits del último año sobre los diez ficheros que con más llamadas ha funciones ha sido de 96, entre los que destaca la actividad del usuario *gpastore* con 50 commits, seguido de lejos por *kmaraas*.

La gráfica 7.13 que representa los informes de uso y los commits del último año, ofrece un resultado que no se asemeja al encontrado en la herramienta *Gedit*. Los dos ficheros que más se utilizan son también los que más se modifican.

7.11.3. Actividad alrededor de los ficheros más utilizados

Llamadas	Fichero	commits	committers	id committers(commits)
124271	src/terminal-window.c	124	19	gpastore(9), ovitters(1), kmaraas(7), markmc(1), hadess(1), newren(1), mariano(32), aflinta(1), hp(57), nalin(2), jrb(2), shivram_u(1), cneumair(1), deepa(1), michael(1), andersca(1), carbamide(3), bratislav(1), laca(1),

67080	src/terminal-screen.c	122	14	gpastore(14), ovitters(1), kmaraas(8), dcransto(1), mariano(24), hp(59), laszlo(3), nalin(3), satyak(1), jrb(2), tajima(3), michael(1), carbamide(1), owen(1),
28543	src/terminal-widget-vte.c	36	6	gpastore(2), mariano(3), nalin(22), jrb(1), hp(6), andersca(2),
19053	src/terminal-profile.c	37	7	gpastore(4), kmaraas(2), mariano(6), hp(22), jrb(1), michael(1), menesis(1),
6564	src/eggaccelerators.c	5	2	gpastore(2), hp(3),
4811	src/terminal-notebook.c	8	2	gpastore(7), kmaraas(1),
2647	src/terminal.c	95	13	gpastore(6), ovitters(2), kmaraas(5), dcransto(1), newren(1), mariano(23), hp(49), daniel(1), shivram_u(2), kristian(1), michael(2), jrb(1), andersca(1),
2552	src/encoding.c	18	7	kmaraas(1), pablo(2), mariano(4), roozbeh(1), cfergeau(1), nalin(2), hp(7),
1366	src/terminal-accels.c	33	8	gpastore(3), kmaraas(1), mariano(4), hp(19), jrb(2), gman(1), carbamide(2), bratislav(1),
71	src/profile-editor.c	57	9	gpastore(3), kmaraas(4), dcransto(1), mariano(9), hp(35), hegde(1), msp(2), notting(1), jrb(1),

Cuadro 7.29: Actividad hasta la publicación de GNOME Terminal 2.14.2

Llamadas	Fichero	commits	committers	id committers(commits)
124271	src/terminal-window.c	18	5	gpastore(9), ovitters(1), kmaraas(6), markmc(1), hadess(1),
67080	src/terminal-screen.c	23	4	gpastore(14), ovitters(1), kmaraas(7), dcransto(1),
28543	src/terminal-widget-vte.c	2	1	gpastore(2),
19053	src/terminal-profile.c	6	2	gpastore(4), kmaraas(2),
6564	src/eggaccelerators.c	2	1	gpastore(2),
4811	src/terminal-notebook.c	8	2	gpastore(7), kmaraas(1),
2647	src/terminal.c	14	4	gpastore(6), ovitters(2), kmaraas(5), dcransto(1),
2552	src/encoding.c	1	1	kmaraas(1),
1366	src/terminal-accels.c	4	2	gpastore(3), kmaraas(1),
71	src/profile-editor.c	8	3	gpastore(3), kmaraas(4), dcransto(1),

Cuadro 7.28: Actividad en el año previo a la publicación de GNOME Terminal 2.14.2

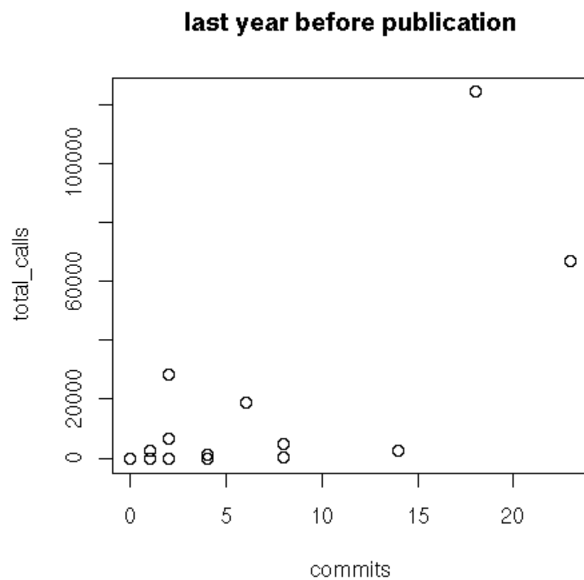

Figura 7.13: Actividad en el año previo a la publicación de GNOME Terminal 2.14.2

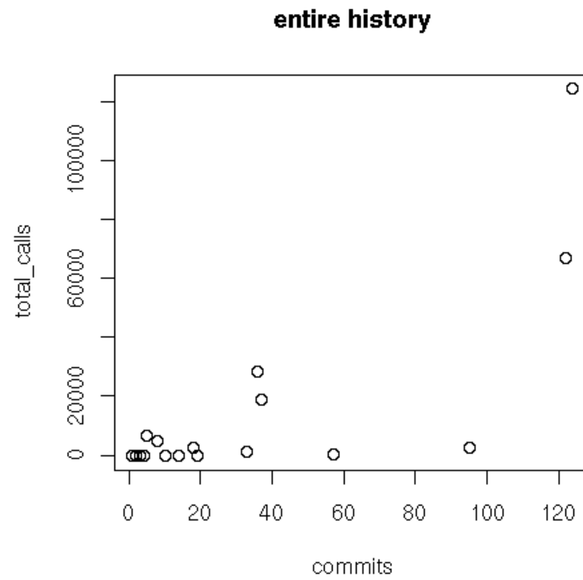

Figura 7.14: Gráfica de uso contra actividad sobre GNOME Terminal 2.14.2

La actividad total alrededor de los diez ficheros más utilizados nos da un total de 535 commits, pero a diferencia del estudio del último año antes de la publicación, para la historia completa el desarrollador que más commits acumula es *hp* con 208, seguido del desarrollador identificado como *mariano* con 105, casi el doble que el desarrollador más activo en el último año, *gpastore*. La media de commits para la historia de la aplicación es de 31,10.

Al igual que en la gráfica de los informes de uso y la actividad del año previo a la publicación, la figura 7.14 refleja un resultado que sólo se repite en la herramienta *Gedit*, destacando los dos ficheros que más se han modificado como los que más llamadas a funciones acumulan.

7.12. Resultados sobre Rhythmbox 0.9.6

7.12.1. Datos obtenidos

Aplicación	Informes	Media de muestreo	Total llamadas
Rhythmbox 0.9.6	43	95,02 %	125317052

Cuadro 7.30: Datos recolectados para Rhythmbox 0.9.6

7.12.2. Funciones más utilizadas

La función *g_sequence_node_update_fields* es la más utilizada con una amplia ventaja sobre el resto. Mientras que la media de llamadas a función es de 39757, la función más utilizada la supera 380 veces. Su utilidad es la de actualizar el número de nodos que se encuentra a ambos lados del nodo a actualizar, modificando una estructura de nodos enlazados que maneja la aplicación Rhythmbox. La función que más llamadas acumula es llamada por funciones como la segunda, *g_sequence_node_rotate*, que también se encarga de realizar modificaciones sobre esa estructura de nodos. La tercera función, de nombre *rb_player_get_type*, se utiliza en cada conversión explícita de tipos, ya que es la encargada de registrar los objetos en GLib.

7.12.3. Actividad alrededor de los ficheros más utilizados en el último año

Destaca sin duda el fichero *rhythmdb/rhythmdb.c*, que siendo el segundo más utilizado es el que más modificaciones acumula de los diez ficheros más utilizados y el segundo más modificado a sólo 2 commits de este.

El total de commits para los ficheros es de 413, entre los que destacan los committers *jrl* y *jmatthew* 249 y 104 commits respectivamente.

La figura 7.15 es peculiar ya que muestra como el fichero más utilizado recibe muy pocas modificaciones, pero el segundo más modificado es también el segundo más utilizado. La mediana de 10 commits, refuerza la idea de que para esta aplicación una gran parte del código recibe muy pocas modificaciones.

Total llamadas	Función	Fichero
15178709	g_sequence_node_update_fields	rhythmdb/gsequence.c
7494980	g_sequence_node_rotate	rhythmdb/gsequence.c
7071196	rb_player_get_type	backends/rb-player.c
4790232	rhythmdb_entry_podcast_post_get_type	rhythmdb/rhythmdb.c
4760981	rhythmdb_entry_podcast_feed_get_type	rhythmdb/rhythmdb.c
4554789	splay	rhythmdb/gsequence.c
4154095	rhythmdb_entry_unref	rhythmdb/rhythmdb.c
3382184	rhythmdb_entry_ref	rhythmdb/rhythmdb.c
3164768	g_sequence_ptr_is_end	rhythmdb/gsequence.c
2909734	ptr_compare	shell/rb-play-order-shuffle.c

Cuadro 7.31: 10 funciones más utilizadas para Rhythmbox 0.9.6

Llamadas	Fichero	commits	committers	id committers(commits)
39832337	rhythmdb/gsequence.c	4	3	mccann(1), rps(1), jrl(2),
33703754	rhythmdb/rhythmdb.c	133	5	jmatthew(30), jrl(91), rps(4), mccann(6), hadess(2),
10698792	rhythmdb/rhythmdb-query-model.c	67	4	jrl(38), jmatthew(21), mccann(6), rps(2),
10609105	backends/rb-player.c	4	2	jrl(3), mccann(1),
5400435	rhythmdb/rhythmdb-tree.c	48	4	jmatthew(16), jrl(27), mccann(3), rps(2),
3879978	backends/gstreamer/rb-player-gst.c	23	4	jmatthew(6), mc-cann(1), jrl(15), rps(1),
3393815	shell/rb-shell-player.c	86	4	jmatthew(28), jrl(48), mccann(9), rps(1),
2948916	shell/rb-play-order-shuffle.c	9	4	jrl(4), mccann(3), rps(1), jmatthew(1),
2940018	rhythmdb/rb-refstring.c	13	3	jrl(9), mccann(2), rps(2),
1765181	shell/rb-shell-clipboard.c	26	5	jmatthew(3), jrl(14), mccann(7), rps(1), hadess(1),

Cuadro 7.32: Actividad en el año previo a la publicación de Rhythmbox 0.9.6

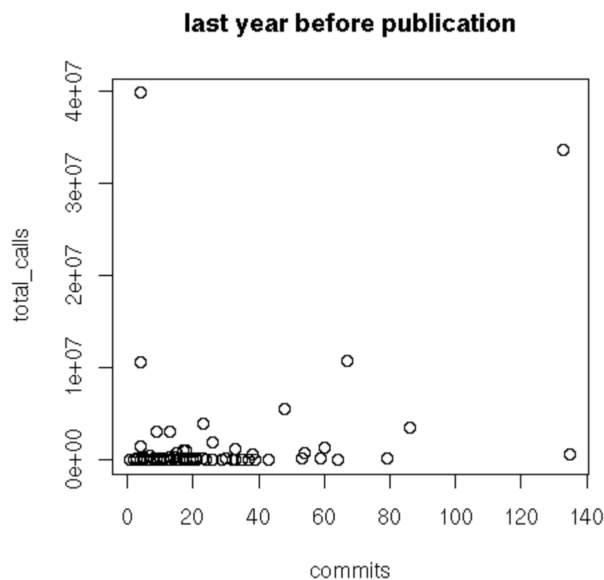

Figura 7.15: Actividad en el año previo a la publicación de Rhythmbox 0.9.6

7.13. Resultados sobre Rhythmbox 0.10.0

7.13.1. Datos obtenidos

Aplicación	Informes	Media de muestreo	Total llamadas
Rhythmbox 0.10.0	40	100 %	250896921

Cuadro 7.33: Datos recolectados para Rhythmbox 0.10.0

7.13.2. Funciones más utilizadas

A diferencia de su predecesor, el código de la versión *0.10.0* incluye la librería libegg, por lo que ha sido necesario corregir los resultados. Cinco de las diez funciones más utilizadas eran parte de ella, ocupando los tres primeros puestos.

De las tres primeras funciones de la tabla, las dos primeras se utilizan para el manejo de una sección crítica, mientras que la tercera *rb_player_get_type* se sigue utilizando de igual manera que para la versión *0.9.6*. El resto de las funciones de la lista se ocupan de consultar e insertar datos en la base de datos que Rhythmbox utiliza para los ficheros multimedia.

La proporción entre el número de llamadas de la función más utilizada y la media baja ahora hasta las 280.

7.13.3. Actividad alrededor de los ficheros más utilizados en el último año

El número total de commits para los diez ficheros más utilizados es de 263, destacando los desarrolladores *jrl* y *jmatthew* con 120 y 96 commits respectivamente. La media de commits en este periodo es de 9,43 y ocho de los diez ficheros más utilizados la sobrepasan.

La figura 7.16 muestra ahora un patrón mucho más parecido al de la figura 7.10 en el que el fichero más utilizado es el segundo más modificado. Si bien es verdad que el fichero que era más modificado en la versión anterior, ha pasado ahora a formar parte de la biblioteca libegg, por lo que se ha sido eliminado de el estudio.

El desarrollo de esta aplicación coloca también al tercer fichero más modificado en una posición destacada en cuanto a commits, estando sobre el umbral de 40 modificaciones (10 por

Total llamadas	Función	Fichero
11468611	_threads_enter	lib/rb-util.c
11468610	_threads_leave	lib/rb-util.c
6957027	rb_player_get_type	backends/rb-player.c
6396116	rhythmdb_entry_ref	rhythmdb/rhythmdb.c
6314580	rhythmdb_entry_unref	rhythmdb/rhythmdb.c
5342778	rhythmdb_entry_podcast_post_get_type	rhythmdb/rhythmdb.c
5313093	rhythmdb_entry_podcast_feed_get_type	rhythmdb/rhythmdb.c
4768126	rhythmdb_query_model_get_type	rhythmdb/rhythmdb-query-model.c
3083831	rhythmdb_entry_get_ulong	rhythmdb/rhythmdb.c
2931844	rhythmdb_entry_get_type	rhythmdb/rhythmdb.c

Cuadro 7.34: 10 funciones más utilizadas para Rhythmbox 0.10.0

Llamadas	Fichero	commits	committers	id committers(commits)
38061349	rhythmdb/rhythmdb.c	71	4	jrl(36), jmatthew(28), mccann(5), rps(2),
23781467	lib/rb-util.c	15	5	mccann(1), jrl(10), jmatthew(1), rps(1), teuf(2),
17793359	rhythmdb/rhythmdb-query-model.c	39	4	jrl(15), jmatthew(17), mccann(5), rps(2),
10438040	backends/rb-player.c	5	2	jrl(4), mccann(1),
5437976	rhythmdb/rb-refstring.c	8	3	jrl(5), mccann(2), rps(1),
3899102	backends/gstreamer/rb-player-gst.c	18	4	jmatthew(10), jrl(6), mccann(1), rps(1),
3340708	shell/rb-shell-player.c	41	4	jrl(11), jmatthew(25), mccann(4), rps(1),
3217914	rhythmdb/rhythmdb-property-model.c	20	4	jmatthew(5), jrl(10), mccann(4), rps(1),
2099856	widgets/rb-entry-view.c	29	5	jrl(15), mccann(6), jmatthew(6), otte(1), rps(1),
1656460	shell/rb-shell-clipboard.c	17	4	jrl(8), jmatthew(4), mccann(4), rps(1),

Cuadro 7.35: Actividad en el año previo a la publicación de Rhythmbox 0.10.0

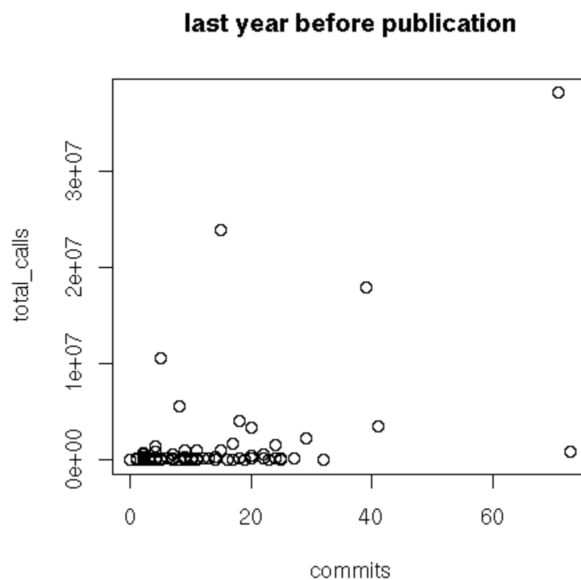

Figura 7.16: Actividad en el año previo a la publicación de Rhythmbox 0.10.0

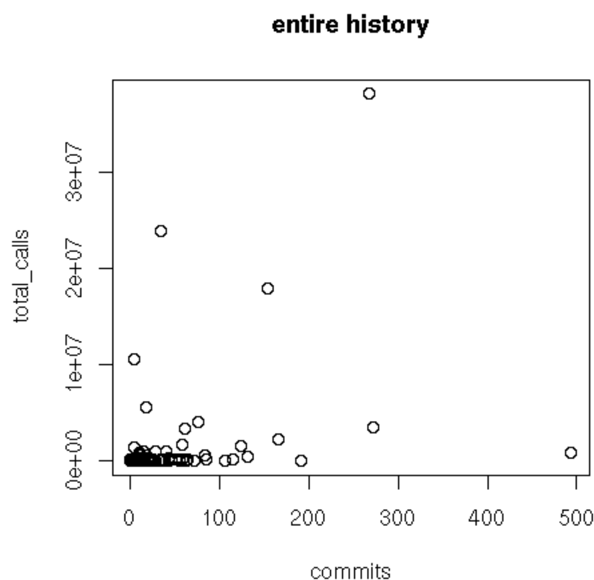

Figura 7.17: Actividad hasta la publicación de Rhythmbox 0.10.0

encima de la media).

7.13.4. Actividad alrededor de los ficheros más utilizados

El estudio de los informes de uso de *Rhythmbox 0.10.0* sobre su historia completa, nos ofrece un listado de los diez ficheros más utilizados en los que destaca de nuevo el fichero *rhythmdb/rhythmdb.c* y el fichero *shell/rb-shell-player.c*. Ambos son dos de los tres ficheros con más modificaciones.

Para estos diez primeros ficheros, a lo largo de su historia se acumula un total de 1112 commits, mientras que la media es de menos 39,43. Los committers más destacados de esos ficheros son *jrl* y *walters* con 371 y 318 commits respectivamente. Este último no aparece en actividad alguna relacionada con el año previo a las publicaciones de las dos versiones estudiadas de *Rhythmbox*.

La figura 7.17 destaca al fichero que más llamadas a funciones utiliza como el segundo más modificado. El resto de la gráfica refleja lo que se puede apreciar en los otros estudios, en los que hay un conjunto mayoritario de ficheros que se modifican con poca frecuencia y acumulan pocas llamadas a funciones y un fichero que acumula muchos commits y se utiliza poco.

7.14. Estimación de Error

Efectuando una verificación de los resultados obtenidos, se encontró que la herramienta MSR tenía un error de planteamiento que incidió directamente sobre los resultados obtenidos. Este error provocaba que la información asociada a los ficheros cuyo path ha sido cambiado a lo largo de su historia no fuera completa.

Lo primero que se hizo fue estimar en qué medida esa carencia de funcionalidad afectó a los datos obtenidos. Puesto que el error afecta a la historia de los ficheros, se generó una serie

Llamadas	Fichero	commits	committers	id committers(commits)
38061349	rhythmdb/rhythmdb.c	267	8	jrl(112), jmatthew(37), mccann(7), rps(4), hadess(12), walters(71), teuf(3), cwalters(21),
23781467	lib/rb-util.c	35	8	mccann(1), jrl(15), jmatthew(1), rps(1), teuf(3), walters(7), hadess(1), cwalters(6),
17793359	rhythmdb/rhythmdb-query-model.c	154	7	jrl(50), jmatthew(27), mccann(6), rps(2), hadess(2), walters(53), cwalters(14),
10438040	backends/rb-player.c	5	2	jrl(4), mccann(1),
5437976	rhythmdb/rb-refstring.c	18	5	jrl(11), mccann(2), rps(2), hadess(1), walters(2),
3899102	backends/gstreamer/rb-player-gst.c	76	7	jmatthew(11), jrl(21), mccann(1), rps(1), cschmidt(4), walters(36), teuf(2),
3340708	shell/rb-shell-player.c	272	17	jrl(58), jmatthew(39), mccann(9), rps(1), walters(67), rslinckx(1), rbultje(1), hadess(5), teuf(2), cwalters(57), cfergeau(1), jschurger(1), company(1), jbaayen(26), omartin(1), jwillcox(1), kenneth(1),
3217914	rhythmdb/rhythmdb-property-model.c	61	6	jmatthew(10), jrl(26), mccann(5), rps(1), walters(13), cwalters(6),
2099856	widgets/rb-entry-view.c	166	9	jrl(57), mccann(6), jmatthew(13), otte(1), rps(1), walters(63), hadess(2), teuf(2), cwalters(21),
1656460	shell/rb-shell-clipboard.c	58	9	jrl(17), jmatthew(6), mccann(7), rps(1), hadess(1), walters(6), cwalters(10), jbaayen(9), mpeseng(1),

Cuadro 7.36: Actividad hasta la publicación de Rhythmbox 0.10.0

de scripts que dado el fuente ofrecido por un cliente de SVN, comparaba el path actual de los ficheros fuente y el path original. El resultado total se puede observar en la figura 7.37.

Aplicacion	Version	$\%Ficheros$
eog	2.16.3	0
eog	2.18.1	0
evince	0.4.0	1,08
evince	0.8.1	15
evolution	2.6.3	1,4
evolution	2.10.1	0,14
gaim	2.0.0+beta5	35
gaim	2.0.0+beta6	38
gedit	2.14.4	1,2
gedit	2.18.1	1,2
gnome terminal	2.14.2	0
nautilus	2.14.3	0,06
rhythmbox	0.9.6	0,7
rhythmbox	0.10.0	0,6

Cuadro 7.37: Estimación de error provocado sobre el estudio de la historia completa

Para tres de las catorce aplicaciones estudiadas el error estimado es realmente alto. Dado que el estudio con la historia completa del código se ve afectado, se evaluó si el estudio sobre la actividad del último año previo a la publicación podría contener un error menor. El resultado para ambas versiones de la aplicación *Gaim* no fue en absoluto bueno. Ambas versiones comparten en el año previo a su publicación una reestructuración del código que movió unos ciento cuarenta ficheros, lo que deja el estudio sobre esta herramienta incompleto.

La otra aplicación que tenía un elevador porcentaje de error es *evince* en su versión *0.8.1*. La diferencia entre las dos versiones de la herramienta indica que en el año previo a la publicación de la versión *0.8.1* hubo una reestructuración de los ficheros en el repositorio. Al no ser el porcentaje tan alto como en el caso anterior el estudio se sigue dando por válido.

7.15. Aparición de la biblioteca libegg

El uso de la biblioteca *libegg* por parte de las aplicaciones afectó a una primera versión de los resultados que tuvo que ser corregida. La biblioteca forma parte del repositorio GNOME y contiene código que aún no ha sido migrado a su destino final o bien porque añade una funcionalidad demasiado nueva para ser incluida en gtk+, por problemas de incompatibilidad con la API o por alguna otra razón. Este código invalidaba el estudio de las partes más utilizadas de la aplicación, ya que aunque no lo sea oficialmente, es en la práctica considerada una biblioteca común a las aplicaciones del escritorio GNOME.

Los resultados más afectados se encontraron en las aplicaciones *EOG*, *Rhythmbox*[1] y *Evince*[2], en los que la gran mayoría de las funciones más utilizadas se hallaban en esa biblioteca. El

[1] En su versión 0.10.0
[2] En su versión 0.4.0

impacto en el resto de los resultados no se reflejaba en las funciones ni ficheros más utilizados, por lo que no fue necesario su corrección.

7.16. Conclusiones extraídas del estudio de informes

El estudio de los informes arroja comportamientos parecidos en muchas de las aplicaciones estudiadas. Comparando las gráficas obtenidas para el estudio de los proyectos desde su creación, podemos distinguir tres grupos bien diferenciados dentro de los que se encuentran la mayoría de los ficheros.

El grupo más numeroso de ficheros es aquél que obteniendo un número de llamadas acumuladas a funciones mínimo, se ha modificado por debajo de la media. Este grupo se ve claramente en las figuras 7.9, 7.12 y 7.17.

El segundo grupo o elemento a destacar, lo compone el fichero que con diferencia más llamadas a funciones acumula. Este, dependiendo de la aplicación puede estar en la cola de lo que se refiere a commits o en la cabeza, por lo que se puede inferir que los proyectos aquí estudiados se pueden dividir en aquellos cuya funcionalidad crítica raras veces se modifica y aquellos cuya funcionalidad más importante está en continua mejora. Los proyectos que rara vez modifican su código más crítico pueden ser consultados en las figuras 7.3, 7.6 y 7.9. Mientras que los proyectos cuya funcionalidad crítica recibe un número muy alto de moficaciones respecto al resto se pueden observar en las figuras 7.12, 7.14 y 7.17.

El tercer elemento a destacar del análisis de uso y modificación sobre el código, es el que ocupa el fichero que teniendo el mayor número de commits, tiene un número de llamadas acumuladas tan pequeño como los ficheros del primer grupo. Este fichero obtiene una elevada notoriedad en las figuras de 7.3, 7.9, 7.12 y 7.17.

Observando la actividad de los dos ficheros en cabeza de las llamadas a funciones y de commits, aparecen dos casos curiosos para el estudio de las herramientas *Gedit* y *Rhythmbox*. En ambas aplicaciones, como puede consultarse en las figuras 7.12 y 7.17, el fichero que más llamadas acumula está en segundo puesto en la clasificación del número de commits, pero la diferencia de uso entre el primero en la lista de commits y el segundo es muy grande. De hecho, el primer fichero en commits, acumula un número de llamadas a funciones despreciable. Es decir, el grupo de desarrolladores están realizando un número de commits parecidos en ambos ficheros; mientras que uno de ellos es el que tiene el código crítico, el otro contiene código que no ha sido prácticamente utilizado.

Después del vistazo general a la relación hallada entre el uso y las modificaciones de cada fichero, es conveniente observar las diferencias de esta relación entre versiones distintas de la misma aplicación. Por ejemplo, para la aplicación *Evince* se observa que durante el año previo a la publicación de su versión *0.4.0* y *0.8.1* se pueden diferenciar tres ficheros, que destacan sobre un grupo de ficheros que acumula pocas llamadas a funciones y commits (ver figuras 7.4 y 7.5). En el tiempo que separa ambas versiones, el fichero más modificado ha obtenido una mayor relevancia, lo que supone que ese esfuerzo en la inclusión de funcionalidad que antes no se usaba tanto, ha obtenido sus frutos en versiones posteriores.

Además del caso de *Evince*, existe otro caso que podría ser de interés, el de la aplicación *Gedit*. En esta, el fichero más utilizado tiene un comportamiento muy distinto en el año previo a las dos publicaciones estudiadas (ver figuras 7.10 y 7.11). Se puede observar que para la versión *2.14.4* el fichero más utilizado es también aquel sobre el que más commits se han realizado, lo

que implica un esfuerzo en la mejora de la parte más crítica de la aplicación. Como se puede ver en la segunda figura, para la versión *2.18.1* el fichero que contiene el código más utilizado ha dejado de ser modificado con tanta frecuencia. Este comportamiento hace suponer que la tendencia de ese fichero seguirá siendo la misma en versiones posteriores, si es que el grupo de desarrolladores no ve una carencia en la parte básica de la aplicación. En el resto de los casos en los que se observa el comportamiento de versiones distintas de una misma aplicación, no hay diferencias notables salvo alguna excepción como *Rhythmbox*, en la que para su segunda versión estudiada, el fichero que más se utilizaba pasó a formar parte de la biblioteca libegg y por consiguiente a dejar de formar parte del estudio.

Capítulo 8

Conclusiones y trabajos futuros

Durante la realización del proyecto fin de carrera se han realizado las siguientes tareas:

- obtención de un conocimiento amplio en una herramienta compleja como *sampler-cc*
- instrumentación de un grupo de aplicaciones GNOME
- realización de pruebas sobre el rendimiento de las aplicaciones instrumentadas
- las herramientas instrumentadas han sido empaquetadas en Debian, para asegurar una fácil instalación de las aplicaciones
- creación de un repositorio con las aplicaciones modificadas para facilitar su instalación
- el estudio se publicitó para fomentar los voluntarios dentro de un entorno real de usuarios
- instalación de la infraestructura necesaria para la recolección de informes
- realización de scripts de análisis de los informes y de la historia del código fuente
- estudio de las partes del código más utilizado de cada aplicación
- estudio de la historia de cada aplicación
- cálculo de medidas estadísticas para los datos obtenidos

Una de las consecuencias del trabajo realizado, es la obtención de un primer paso en lo que se refiere a comprender el modo en el que herramientas del ámbito del software libre son utilizadas por usuarios finales. El estudio de los casi seis mil informes de uso, ha hecho posible clasificar el código de las aplicaciones del estudio según su utilización por los usuarios, lo que plantea algunas vías de futuro en colaboración con los desarrolladores de las herramientas. Por ejemplo, para un desarrollador sería importante saber que el código que está modificando es crítico o todo lo contrario, ya que dependiendo de la importancia del mismo, se debería tener diferentes precauciones. Otra posible vía de trabajo surge del lado de la gestión de proyecto, que o bien para dedicar esfuerzos a nuevas implementaciones o bien para dedicar esfuerzos a eliminar bugs, podría tener en cuenta la importancia que el código tiene según el punto de vista de el cliente o usuario final.

Los datos sobre las partes más utilizadas de una aplicación no son algo nuevo, pero al unirlo al estudio que hacen las herramientas MSR de la historia del código obtenemos un nuevo

parámetro con el que ofrecer un estudio más profundo sobre el desarrollo del software libre. Este estudio se puede profundizar de dos maneras completamente compatibles, la primera es aplicando el instrumentador a un número elevado de herramientas, la segunda es obtener un número elevado de usuarios de los que obtener informes de uso. Sin duda alguna esto último, la participación de los usuarios, ha sido una de las claves del proyecto, que gracias a la participación de unos diez usuarios que utilizaron sus herramientas para realizar tareas cotidianas, pudo evaluar positivamente el rendimiento de las aplicaciones instrumentadas.

La selección de las herramientas del proyecto *Cooperative Bug Isolation* fue un acierto ya que consiguieron, no sin antes ser probadas y estudiadas, construir aplicaciones instrumentadas cuyo rendimiento era parejo al de la aplicación original. Por otro lado, la manera más o menos uniforme de empaquetar las aplicaciones GNOME en Debian, ha ofrecido un método sencillo de ofrecer las aplicaciones instrumentadas facilitando su instalación. Dicho esto, el objetivo de reunir el conocimiento suficiente de las herramientas de CBI se da también por cumplido. El cumplimiento de este objetivo era la raíz para poder efectuar (entre otros) el estudio de viabilidad sobre la instrumentación de distribuciones de software completas, que a su vez también depende del grado de automatización que se haya logrado con la instrumentación de las herramientas. Si se pretendiera instrumentar aplicaciones GNOME de manera automática, la carga de trabajo a efectuar después del conocimiento aglutinado en el proyecto no sería alta, ya que (al menos en Debian) los mantenedores de los paquetes GNOME usan una manera estándar de empaquetar las aplicaciones para la gran mayoría de estas. Comparando las facilidades que ofrecen las herramientas de CBI para trabajar con GNOME y las que brinda Debian para manejar paquetes de aplicaciones de este escritorio, la instrumentación de cualquier otro grupo de aplicaciones sí supondría una complicación. Debemos tener en cuenta que para acometer la instrumentación de las aplicaciones de toda una distribución, dependemos de que los mantenedores hayan usado Makefiles parecidos para que podamos automatizar al máximo el proceso de instrumentación.

Como resumen de las conclusiones y sin ánimo de repetir los resultados del estudio de los informes[1] , es necesario mencionar que tras el proyecto se conoce algo más del comportamiento de los ficheros y sus modificaciones para las aplicaciones estudiadas y que se cuenta con un método escalable con el que se puede ampliar el estudio a más aplicaciones y usuarios.

Para acabar, se muestra un listado de mejoras a tener en cuenta para posibles trabajos futuros como el de instrumentar una distribución de paquetes compleja (como GNOME o por qué no, Debian):

- la herramienta CVSAnalY, que imposibilitó la aparición de más herramientas en el estudio realizado para el proyecto[2] debe ser mejorada
- la herramienta MSR (ya sea CVSAnalY u otra) aportaría mucha más riqueza al estudio si ofreciera la historia no de cada fichero, sino de cada función del código
- no sólo el número de los usuarios debe aumentar, la variedad de los perfiles de los usuarios debe ser heterogénea para una evaluación más fiel de las partes críticas del código
- ofrecer versiones de las aplicaciones más recientes atraerá más al grupo de desarrolladores que verán como algo positivo el estudio, lo que podría ofrecer una buena publicidad de cara a tener un número mayor de usuarios

[1]El estudio de los informes está disponible en el capítulo 7.16

[2]El error que apareció durante el estudio con la herramienta CVSAnalY fue comentado en la sección 7.14

Bibliografía

[Amo07] Juan José Amor. Sloccount web for debian etch. Website, 2007. http://libresoft.dat.escet.urjc.es/debian-counting/etch/ (accedido el día 17 de Julio).

[CH05] Kevin Crowston and James Howison. The social structure of free and open source software development. *First Monday*, 10(2), February 2005. http://www.firstmonday.dk/issues/issue10_2/crowston/.

[GSy05] GSyC/LibreSoft. Libresoft research group. Website, 2005. https://www.libresoft.es (accedido el día 15 de Junio).

[GSy07] GSyC/LibreSoft. Libresoft tools web site. Website, 2007. https://forge.morfeo-project.org/projects/libresoft-tools/ (accedido el día 25 de Junio).

[Inc06] Google Inc. Google trends, 2006. http://www.google.com/trends?q=ubuntu\%2C+suse\%2C+debian\%2C+fedora\%2C+redhat\&ctab=0\&geo=all\&date=all (accedido el día 15 de Junio).

[Inc07] Softpedia Inc. French parliament will switch from microsoft to ubuntu, 2007. http://news.softpedia.com/news/French-Parliament-Will-Switch-from-Microsoft-to-Ubuntu-49186.shtml (accedido el día 15 de Junio).

[Lib06] Ben Liblit. Cbi project web site. Website, 2006. http://www.cs.wisc.edu/cbi/ (accedido el día 15 de Marzo).

[pro07] MUCode project. Libresoft research group - mucode project. Website, 2007. http://tools.libresoft.es/mucode (accedido el día 15 de Junio).

[RAGBH05] Gregorio Robles, Juan José Amor, Jesús M. González-Barahona, and Israel Herraiz. Evolution and growth in large libre software projects. In *Proceedings of the International Workshop on Principles in Software Evolution*, pages 165–174, Lisbon, Portugal, September 2005.

[Ray98] Eric S. Raymond. The cathedral and the bazar. *First Monday*, 3(3), March 1998. http://www.firstmonday.dk/issues/issue3_3/raymond/.

[RMGB05] Gregorio Robles, Juan Julián Merelo, and Jesús M. González-Barahona. Self-organized development in libre software: a model based on the stigmergy concept. In *Proceedings of the 6th International Workshop on Software Process Simulation and Modeling (ProSim 2005)*, St.Louis, Missouri, USA, May 2005.

[Sca04] Walt Scacchi. Free and Open Source development practices in the game community. *IEEE Software*, 21(1):59–66, 2004.

[SE94] Amitabh Srivastava and Alan Eustace. Atom: a system for building customized program analysis tools. In *PLDI '94: Proceedings of the ACM SIGPLAN 1994 conference on Programming language design and implementation*, pages 196–205, New York, NY, USA, 1994. ACM Press.

[Sta99] Richard Stallman. The GNU operating system and the free software movement. In Chris DiBona, Sam Ockman, and Mark Stone, editors, *Open Sources: Voices from the Open Source Revolution*. O'Reilly and Associates, Cambridge, Massachusetts, 1999.

[Std98] Ieee std 830-1998 ieee recommended practice for software requirements specifications. Website, 1998. `http://standards.ieee.org/reading/ieee/std/se/830-1998.pdf` (accedido el día 23 de Mayo).

[wik01] Free software, 2001. `http://en.wikipedia.org/wiki/Free_software` (accedido el día 15 de Agosto del 2007).

Apéndice A

Proceso de instrumentación con sampler-cc

Dado que una parte importante del esfuerzo ha sido enfocada en la comprensión de la herramienta de Liblit, vamos a dedicar éste punto a explicar cuál es el proceso por el cual una aplicación pasa a convertirse en una aplicación instrumentada. Este proceso es llevado a cabo por el instrumentador, una herramienta cuyo comportamiento externo imita al compilador nativo, pero que internamente inyecta código en el fuente original para obtener la instrumentación. El instrumentador, *sampler-cc*, está implementado como una transformación de código fuente a código fuente para el lenguaje C utilizando CIL (*C Intermediate Language*).

Para describir de manera amena y sencilla cómo funciona el instrumentador, se incluyen a continuación diferentes aproximaciones de cómo "vigilarüna parte del código. Se concluye con la estrategia que actualmente utiliza sampler-cc.

En el ejemplo a continuación se instrumenta el código con el objetivo de vigilar dos sentencias. En la primera se observará si el puntero apunta en algún momento a NULL, mientras que en la segunda se observará cual es el máximo valor elegido.

El fragmento de código a instrumentar será éste:

```
{
  p = p -> next;
  total += sizes[i];
}
```

La primera aproximación será inyectar código para que ejecute el análisis de las dos sentencias.

```
{
  check(p != NULL);
  p = p -> next;

  check(i < max);
  total += sizes[i];
}
```

Con la aproximación que se ve en el ejemplo de arriba, el rendimiento de la aplicación se vería afectado, ya que no habría manera de no efectuar el chequeo el 100 % de las veces. Con el objetivo de evitar ésto, otra aproximación algo simple sería la siguiente:

```
{
  if (rnd(100) == 0) check(p != NULL);
  p = p -> next;

  if (rnd(100) == 0) check(i < max);
  total += sizes[i]
}
```

Esta estrategia tiene algunos problemas prácticos. El principal es que la generación de números aleatorios no es gratuita en tiempo, de hecho puede ser incluso más lenta que ejecutar el *check* incondicionalmente.

Si fijamos la tasa de muestreo en 1 de cada 100 coincidencias, cada una de las sentencias tendrá un 99 % de posibilidades de no ser chequeada, con lo que el bloque tendrá un $(\frac{99}{100})^2$ de posibilidades de que ambas sentencias no sean chequeadas. La estrategia consiste en poder determinar antes del comienzo del bloque si alguna de las sentencias va a ser chequeada, para poder en caso contrario ejecutar el código sin comprobación alguna. Además será necesario poder determinar cuantas iteraciones faltan para que se produzca el siguiente muestreo. Para llevar a cabo ésto el analizador deberá generar dos versiones del código, una sin alterar y otra en la que se comprueba y llama a la función *check*.

La solución ofrecida por Ben Liblit se basa en simular una cuenta atrás mediante una distribución geométrica. Se busca mediante ésta solución poder determinar antes del comienzo del bloque si alguna de las sentencias va a ser chequeada, para poder en caso contrario ejecutar el código sin comprobación alguna. Además será necesario poder determinar cuantas iteraciones faltan para que se produzca el siguiente muestreo. Para llevar a cabo ésto, el analizador deberá generar dos versiones del código, una sin alterar y otra en la que se comprueba si toca muestrear.

```
{
  if (countdown > 2) {
    /* este es el atajo!! no se recogen datos*/
    countdown -=2;
    p = p->next;
    total += sizes[i];

  } else {

    /* el muestreo es inminente en alguno de los siguientes checks*/

    if (--countdown == 0) {   /* muestreo activado?*/
       check(p != NULL);
       countdown = getNextCountdown();  /* siguiente muestreo será ...*/
    }
```

```
    p = p-> next;

    if (--countdown ==0) {     /* muestreo activado?*/
      check(i<max);
      countdown = getNextCountdown();  /* siguiente muestreo será ...*/
    }
    total += sizes[i];
  }
}
```

La ventaja obtenida con el atajo o *free fast path* es clara, si fijamos la tasa de muestreo en 1 de cada 100 coincidencias, cada una de las sentencias tendrá un 99 % de posibilidades de no ser muestreada, con lo que el bloque tendrá un $(\frac{99}{100})^2$ de posibilidades de que ambas sentencias no sean muestreadas. (FIXME no me gusta ésto aquí al final, quizá antes tendría más sentido)

Apéndice B

Modificación de Makefiles de Debian

El cauce que sigue la creación de un paquete Debian usa una serie de scripts llamados Makefile, que automatizan la compilación y configuración de la herramienta, en nuestro caso del escritorio GNOME. Para realizar la compilación con las herramientas del proyecto *CBI* es necesario antes introducir una serie de pequeños cambios. Las modificaciones afectan a los siguientes ficheros:

- /usr/share/cdbs/1/rules/debhelper.mk
- /usr/share/cdbs/1/class/GNOME.mk
- /usr/share/cdbs/1/class/langcore.mk

La primera modificación surge de la necesidad de extraer la información que asocia los informes de uso emitidos por la aplicación con el código fuente. Esta información es alojada en las secciones ELF[1] y es eliminada después de la compilación, justo antes de construir el paquete. El responsable de eliminar esas secciones es el fichero *debhelper.mk* [2] y en él se incluye un script para extraer la información antes de la eliminación. La parte modificada del fichero queda así:

```
$(patsubst %,binary-strip-IMPL/%,$(DEB_ALL_PACKAGES)) :: binary-strip-IMPL/%:

    /home/luis/mucode/extract-sections $(DEB_BUILDDIR)/debian/$(cdbs_curpkg)
../$(cdbs_curpkg).sites

    $(if $(is_debug_package),,dh_strip -p$(cdbs_curpkg) $(call cdbs_add_dashx,
$(DEB_STRIP_EXCLUDE)) $(DEB_dh_STRIP_ARGS))
```

Figura B.1: Modificación aplicada al fichero debhelper.mk

La segunda modificación sobre los scripts originales, tiene lugar en el fichero *GNOME.mk* y tiene por objetivo guardar la configuración básica de la aplicación GNOME necesaria para el correcto envío y muestreo de los datos. Así pues es necesario modificar reglas del Makefile

[1] Hay más información sobre las secciones ELF en el apartado 6.1.1

[2] En el fichero debhelper.mk se encuentra el código responsable de eliminar los símbolos de los ejecutables, bibliotecas compartidas y estáticas que no son usadas para depuración

```
ifdef _cdbs_rules_debhelper
$(patsubst %,binary-install/%,$(DEB_PACKAGES)) :: binary-install/%:
/usr/lib/sampler/tools/install-gconf --name=GNOME-terminal --sparsity=1
--reporting-url=http://cbi4debian.libresoft.es/reports/sampler-upload.pl --install=
debian/GNOME-terminal/ --template=/usr/lib/sampler/tools/application.schemas.in
dh_scrollkeeper -p$(cdbs_curpkg) $(DEB_dh_SCROLLKEEPER_ARGS)
$(if $(wildcard /usr/bin/dh_gconf),dh_gconf -p$(cdbs_curpkg) $(DEB_dh_GCONF_ARGS))
$(if $(wildcard /usr/bin/dh_desktop),dh_desktop -p$(cdbs_curpkg) $(DEB_dh_DESKTOP_ARGS))
endif
```

Figura B.2: Modificación aplicada al fichero GNOME.mk

encargado de gestionar la instalación del binario, para que antes de acabar se añadan a la base de datos de Gconf[3] los datos que la aplicación instrumentada utilizará en su muestreo. Los datos a ser introducidos en GConf son la tasa de muestro (*sparsity*) y la dirección del servidor donde será enviado el informe de uso. La modificación realizada sobre el fichero añade la llamada al script *install-gconf*.

La tercera y última modificación se realizó tras encontrar un bug[4] en uno de los Makefiles que se utilizan en todos los paquetes de GNOME. El fallo en cuestión hacía que en el momento de compilación las opciones que se le pasaban la compilador C elegido (en nuestro caso, el hecho por Liblit) no fueran tomados en cuenta. Un error tan nimio en uno de los Makefile de las distribuciones Debian, que omitía el anterior valor de la variable de entorno CFLAGS retrasó el lanzamiento del primer prototipo.

La simple modificación de *CFLAGS = -g -Wall* a *CFLAGS += -g -Wall* solucionó el problema.

[3]Gconf es un sistema utilizado por el escritorio GNOME para albergar la configuración para el escritorio y las aplicaciones que se ejecutan en él.

[4]En informática es común referirse a un error software como un *bug*

```
_cdbs_scripts_path ?= /usr/lib/cdbs
_cdbs_rules_path ?= /usr/share/cdbs/1/rules
_cdbs_class_path ?= /usr/share/cdbs/1/class

ifndef _cdbs_class_langcore
_cdbs_class_langcore = 1

CFLAGS = -g -Wall
CXXFLAGS = -g -Wall
ifneq (,\$(findstring noopt,\$(DEB_BUILD_OPTIONS)))
        DEB_OPT_FLAG = -O0
else
        DEB_OPT_FLAG = -O2
endif
CFLAGS += \$(DEB_OPT_FLAG)
CXXFLAGS += \$(DEB_OPT_FLAG)

endif
```

Figura B.3: Error en la variable de entorno CFLAGS

Apéndice C

Informes de uso

Esta sección del apéndice está dedicada a explicar con algún breve ejemplo en qué consiste un informe de uso.

Los informes de uso, enviados por aplicaciones instrumentadas, son recolectados en un servidor para su posterior análisis. Una vez recibidos en el servidor se almacena cada uno de ellos en un directorio cuyo contenido son dos ficheros. El primero de ellos que está dedicado a ofrecer información sobre el entorno en el que fue obtenido el informe, mientras que el segundo contiene los datos sobre el uso de las funciones.

El fichero *environment* contiene datos sobre la aplicación instrumentada y su entorno:

```
HTTP_SAMPLER_APPLICATION_RELEASE        2.10.1-0ubuntu1
HTTP_SAMPLER_EXIT_SIGNAL        0
HTTP_SAMPLER_EXECUTABLE_PATH    /usr/lib/sampler/wrapped/usr/bin/evolution-2.10
HTTP_SAMPLER_APPLICATION_NAME   evolution
HTTP_SAMPLER_EXIT_STATUS        1
HTTP_SAMPLER_VERSION    1.4.8
HTTP_SAMPLER_APPLICATION_VERSION        2.10.1
HTTP_SAMPLER_INSTRUMENTOR_VERSION       1.4.8
HTTP_SAMPLER_BUILD_DISTRIBUTION ubuntu-feisty-i386
HTTP_SAMPLER_SPARSITY   1
DATE    2007-07-24 14:42:31
```

Figura C.1: El fichero environment

El segundo fichero cuyo nombre es *samples.gz* contiene, en formato comprimido, un fichero de sintaxis XML con información asociada a los ficheros de la aplicación y sus funciones. Esta información asociada es en este caso el número de llamadas a cada función que observó la aplicación instrumentada.

El contenido del fichero es simple. En la figura C.2 se puede ver un extracto en el que aparecen tres elementos a resaltar. El primer elemento es el identificador de *unit* que se asocia a un fichero, el elemento *scheme* se refiere al tipo de instrumentación y el último es el contenido que encierra la etiqueta *samples* y que contiene un número de llamadas para cada función. Para nuestro caso, los informes sólo contienen el patrón de compilación de *function-entries*, lo que significa que aparecerá una única entrada *unit* para cada fichero.

```
<samples unit="aa39ed8d8cafe4be0f1bdff52a5aa31a" scheme="function-entries">
0
0
0
17
28
24
0
1
6
12
</samples>
```

Figura C.2: Ejemplo de sección del informe de uso dedicado a un fichero y sus funciones

```
<sites unit="aa39ed8d8cafe4be0f1bdff52a5aa31a" scheme="function-entries">
lib/rb-util.c   38      rb_true_function        0
lib/rb-util.c   44      rb_false_function       0
lib/rb-util.c   50      rb_null_function        0
lib/rb-util.c   56      rb_copy_function        0
lib/rb-util.c   63      rb_gvalue_compare       0
lib/rb-util.c   204     rb_compare_gtimeval     0
lib/rb-util.c   220     totem_pixbuf_mirror     0
lib/rb-util.c   259     rb_image_new_from_stock 0
lib/rb-util.c   297     rb_gtk_action_popup_menu        0
lib/rb-util.c   311     get_mount_points        0
</sites>
```

Figura C.3: Ejemplo de información sobre las unidades de compilación

Para asociar los datos ofrecidos por el informe con los nombres de los ficheros y funciones, es imprescindible contar con la información obtenida en tiempo de compilación. Con esta información mostrada en la tabla C.3 y el informe de la tabla C.2 podríamos saber, entre otras cosas, que el fichero *lib/rb-util.c* tiene una función en la línea 311, llamada *get_mount_point* que según el informe fue llamada 12 veces.

Apéndice D

Medidas calculadas para las funciones

Aplicación	Media	Mediana	Des. Estándar	Coe. Variación
EOG 2.18.1	11283.34	1	72365.09	6.41
Evince 0.4.0	1618.62	6	11554.80	7.13
Evince 0.8.1	253283,90	77	2822204,35	11,14
Evolution 2.6.3	10553,49	0	173904,79	16,47
Evolution 2.10.1	429991,55	0	8090953,40	18,81
Gedit 2.14.4	7113,12	0	76946,59	10,81
Gedit 2.18.1	1717,40	2	17360,06	10,10
Gnome Terminal 2.14.2	374,02	0	2499,64	6,68
Rhythmbox 0.9.6	39757.94	0	409474.44	10,29
Rhythmbox 0.10.0	34417,79	3	396404,33	11,51

Cuadro D.1: Medidas calculadas sobre el número de llamadas a funciones del estudio

Apéndice E

Medidas calculadas para los ficheros

Aplicación	Media	Mediana	Des. Estándar	Coe. Variación
EOG 2.18.1	221268.69	161	515690.51	2.33
Evince 0.4.0	24299,93	277	77421	3,18
Evince 0.8.1	6879798,28	28125	23765917,27	3,45
Evolution 2.6.3	237873,15	24	1635001,10	6,87
Evolution 2.10.1	9659286,93	1049,5	76033086,59	7,87
Gedit 2.14.4	174056,06	1702,5	639383,71	3,67
Gedit 2.18.1	41757.4	1056.0	142996.78	3,42
Gnome Terminal 2.14.2	13524,10	71	30545,24	2,25
Rhythmbox 0.9.6	1062008,91	1036	4958038,79	4,66
Rhythmbox 0.10.0	964277,96	972	4387210,51	4,54

Cuadro E.1: Medidas calculadas sobre el número de llamadas a funciones acumuladas por los ficheros del estudio

Aplicación	Media	Mediana	Des. Estándar	Coe. Variación
EOG 2.18.1	4,11	2	7,27	1,76
Evince 0.4.0	11,82	4	30,18	2,55
Evince 0.8.1	6,34	2	15,46	2,43
Evolution 2.6.3	2,66	1	3,97	1,49
Evolution 2.10.1	2,25	1	3,41	1,51
Gedit 2.14.4	3,17	2	3,32	1,04
Gedit 2.18.1	3,59	2	4,40	1,22
GNOME Terminal 2.14.2	5,21	2	6,28	1,20
Rhythmbox 0.9.6	18,5	10	22,96	1,24
Rhythmbox 0.10.0	9,43	5	11,46	1,21

Cuadro E.2: Medidas calculadas sobre el número de commits en el año previo a la publicación

Aplicación	Media	Mediana	Des. Estándar	Coe. Variación
EOG 2.18.1	20,91	8,0	39,79	1,90
Evince 0.8.1	21,82	4	60,85	2,78
Evolution 2.10.1	36,82	16	61,92	1,68
Gedit 2.18.1	17,42	8	28,12	1,61
GNOME Terminal 2.14.2	31,10	14	39,07	1,15
Rhythmbox 0.10.0	31,93	11	62,41	1,95

Cuadro E.3: Medidas calculadas sobre el número de commits

Apéndice F

Ejemplo de uso

Este apéndice ofrece un ejemplo de cómo se ha realizado el estudio que pretende facilitar aún más el uso a terceros de la herramienta:

Una vez hemos obtenido los *informes* y contando con toda la información necesaria de los *sites*:

```
luiz@colossus:~/mucode/tools$ ./report_parser.py ../reports/ ../sites/
Total calls = 7107368766
Number of reports = 5526
**evolution_2.10.1_ubuntu-feisty-i386 = 417 reports
417 reports - 417 sparsity - 5351244961 calls
**gnome-terminal_2.14.2_debian-etch = 420 reports
420 reports - 7443 sparsity - 256958 calls
**eog_2.16.3_debian-etch-i386 = 68 reports
68 reports - 334 sparsity - 41797826 calls
**gedit_2.14.4_debian-etch-i386 = 83 reports
83 reports - 273 sparsity - 68776487 calls
**evince_0.4.0_debian_etch-i386 = 149 reports
149 reports - 2191 sparsity - 10738578 calls
**nautilus_2.14.3_debian_etch-i386 = 109 reports
109 reports - 2203 sparsity - 16343006 calls
**eog_2.18.1_ubuntu-feisty-i386 = 154 reports
154 reports - 154 sparsity - 230316764 calls
**evince_0.8.1_ubuntu-feisty-i386 = 3611 reports
3611 reports - 7259 sparsity - 380835188 calls
**evolution_2.6.3_debian_etch-i386 = 99 reports
99 reports - 1671 sparsity - 128689375 calls
**gedit_2.18.1_ubuntu-feisty-i386 = 84 reports
84 reports - 84 sparsity - 2926567 calls
**rhythmbox_0.9.6_debian-etch-i386 = 43 reports
43 reports - 214 sparsity - 125317052 calls
**gaim_2.0.0+beta5_debian-etch-i386 = 59 reports
59 reports - 306 sparsity - 53932502 calls
**rhythmbox_0.10.0_ubuntu-feisty-i386 = 40 reports
```

```
40 reports - 40 sparsity - 250896921 calls
**gaim_2.0.0+beta6_ubuntu-feisty-i386 = 237 reports
237 reports - 237 sparsity - 497709437 calls
```

A continuación se inyecta el SQL obtenido en la base de datos:

```
luiz@colossus:~/mucode/tools$ mysql < reports.sql
```

Mientras tanto, en otro frente iremos utilizando CVSAnalY para obtener la historia del código en un otro fichero de sintaxis SQL. Para cada una de la apps, obtenemos el dump que será insertado también en la base de datos:

```
luiz@colossus:~/srccode/EVINCE_0_4_0$ cvsanaly --user root --password root > SQL
luiz@colossus:~/srccode/EVINCE_0_4_0$ mysql < SQL
```

Para finalizar y con las base de datos de los informes de uso y el CVSAnalY creadas, llamamos al script *db_analyzer.py* para que relacione los datos y ofrezca estos en formato HTML en el fichero de salida indicado. Al acabar la llamada al script, ejecutamos un script de R que será el encargado de dibujar las gráficas a partir de unos ficheros creados especialmente por el script Python antes mencionado.

```
luiz@colossus:~ ./db_analyzer.py root root mucode_reports 4
 etch/evince_0_4_0/evince_0_4_0.html cvsanaly_evince_0_4_0
luiz@colossus:~ cd etch/evince_0_4_0
luiz@colossus:~ R --vanilla < ../../script.r
```

Apéndice G

Licencia Reconocimiento-CompartirIgual 2.5 España

Licencia

LA OBRA (SEGÚN SE DEFINE MÁS ADELANTE) SE PROPORCIONA BAJO LOS TÉRMINOS DE ESTA LICENCIA PÚBLICA DE CREATIVE COMMONS ("CCPL" O "LICENCIA"). LA OBRA SE ENCUENTRA PROTEGIDA POR LA LEY ESPAÑOLA DE PROPIEDAD INTELECTUAL Y/O CUALESQUIERA OTRAS NORMAS RESULTEN DE APLICACIÓN. QUEDA PROHIBIDO CUALQUIER USO DE LA OBRA DIFERENTE A LO AUTORIZADO BAJO ESTA LICENCIA O LO DISPUESTO EN LAS LEYES DE PROPIEDAD INTELECTUAL.

MEDIANTE EL EJERCICIO DE CUALQUIER DERECHO SOBRE LA OBRA, USTED ACEPTA Y CONSIENTE LAS LIMITACIONES Y OBLIGACIONES DE ESTA LICENCIA. EL LICENCIADOR LE CEDE LOS DERECHOS CONTENIDOS EN ESTA LICENCIA, SIEMPRE QUE USTED ACEPTE LOS PRESENTES TÉRMINOS Y CONDICIONES.

1. Definiciones

1. La "obra" es la creación literaria, artística o científica ofrecida bajo los términos de esta licencia.

2. El "autor" es la persona o la entidad que creó la obra.

3. Se considerará "obra conjunta" aquella susceptible de ser incluida en alguna de las siguientes categorías:

    a) "Obra en colaboración", entendiendo por tal aquella que sea resultado unitario de la colaboración de varios autores.
    b) "Obra colectiva", entendiendo por tal la creada por la iniciativa y bajo la coordinación de una persona natural o jurídica que la edite y divulgue bajo su nombre y que esté constituida por la reunión de aportaciones de diferentes autores cuya contribución personal se funde en una creación única y autónoma, para la cual haya sido concebida sin que sea posible atribuir separadamente a cualquiera de ellos un derecho sobre el conjunto de la obra realizada.

c) "Obra compuesta e independiente", entendiendo por tal la obra nueva que incorpore una obra preexistente sin la colaboración del autor de esta última.

4. Se considerarán "obras derivadas" aquellas que se encuentren basadas en una obra o en una obra y otras preexistentes, tales como: las traducciones y adaptaciones; las revisiones, actualizaciones y anotaciones; los compendios, resúmenes y extractos; los arreglos musicales y, en general, cualesquiera transformaciones de una obra literaria, artística o científica, salvo que la obra resultante tenga el carácter de obra conjunta en cuyo caso no será considerada como una obra derivada a los efectos de esta licencia. Para evitar la duda, si la obra consiste en una composición musical o grabación de sonidos, la sincronización temporal de la obra con una imagen en movimiento ("synching") será considerada como una obra derivada a los efectos de esta licencia.

5. Tendrán la consideración de "obras audiovisuales" las creaciones expresadas mediante una serie de imágenes asociadas, con o sin sonorización incorporada, así como las composiciones musicales, que estén destinadas esencialmente a ser mostradas a través de aparatos de proyección o por cualquier otro medio de comunicación pública de la imagen y del sonido, con independencia de la naturaleza de los soportes materiales de dichas obras.

6. El "licenciador" es la persona o la entidad que ofrece la obra bajo los términos de esta licencia y le cede los derechos de explotación de la misma conforme a lo dispuesto en ella.

7. "Usted" es la persona o la entidad que ejercita los derechos cedidos mediante esta licencia y que no ha violado previamente los términos de la misma con respecto a la obra, o que ha recibido el permiso expreso del licenciador de ejercitar los derechos cedidos mediante esta licencia a pesar de una violación anterior.

8. La "transformación" de una obra comprende su traducción, adaptación y cualquier otra modificación en su forma de la que se derive una obra diferente. Cuando se trate de una base de datos según se define más adelante, se considerará también transformación la reordenación de la misma. La creación resultante de la transformación de una obra tendrá la consideración de obra derivada.

9. Se entiende por "reproducción" la fijación de la obra en un medio que permita su comunicación y la obtención de copias de toda o parte de ella.

10. Se entiende por "distribución" la puesta a disposición del público del original o copias de la obra mediante su venta, alquiler, préstamo o de cualquier otra forma.

11. Se entenderá por "comunicación pública" todo acto por el cual una pluralidad de personas pueda tener acceso a la obra sin previa distribución de ejemplares a cada una de ellas. No se considerará pública la comunicación cuando se celebre dentro de un ámbito estrictamente doméstico que no esté integrado o conectado a una red de difusión de cualquier tipo. A efectos de esta licencia se considerará comunicación pública la puesta a disposición del público de la obra por procedimientos alámbricos o inalámbricos, incluida la puesta a disposición del público de la obra de tal forma que cualquier persona pueda acceder a ella desde el lugar y en el momento que elija.

12. La "explotación" de la obra comprende su reproducción, distribución, comunicación pública y transformación.

13. Tendrán la consideración de "bases de datos" las colecciones de obras ajenas, de datos o de otros elementos independientes como las antologías y las bases de datos propiamente dichas que por la selección o disposición de sus contenidos constituyan creaciones intelectuales, sin perjuicio, en su caso, de los derechos que pudieran subsistir sobre dichos contenidos.

14. Los "elementos de la licencia" son las características principales de la licencia según la selección efectuada por el licenciador e indicadas en el título de esta licencia: Reconocimiento de autoría (Reconocimiento), Compartir de manera igual (CompartirIgual).

2. Límites y uso legítimo de los derechos. Nada en esta licencia pretende reducir o restringir cualesquiera límites legales de los derechos exclusivos del titular de los derechos de propiedad intelectual de acuerdo con la Ley de Propiedad Intelectual o cualesquiera otras leyes aplicables, ya sean derivados de usos legítimos, tales como el derecho de copia privada o el derecho a cita, u otras limitaciones como la derivada de la primera venta de ejemplares.

3. Concesión de licencia. Conforme a los términos y a las condiciones de esta licencia, el licenciador concede (durante toda la vigencia de los derechos de propiedad intelectual) una licencia de ámbito mundial, sin derecho de remuneración, no exclusiva e indefinida que incluye la cesión de los siguientes derechos:

1. Derecho de reproducción, distribución y comunicación pública sobre la obra.

2. Derecho a incorporarla en una o más obras conjuntas o bases de datos y para su reproducción en tanto que incorporada a dichas obras conjuntas o bases de datos.

3. Derecho para efectuar cualquier transformación sobre la obra y crear y reproducir obras derivadas.

4. Derecho de distribución y comunicación pública de copias o grabaciones de la obra, como incorporada a obras conjuntas o bases de datos.

5. Derecho de distribución y comunicación pública de copias o grabaciones de la obra, por medio de una obra derivada.

6. Para evitar la duda, sin perjuicio de la preceptiva autorización del licenciador, y especialmente cuando la obra se trate de una obra audiovisual, el licenciador renuncia al derecho exclusivo a percibir, tanto individualmente como mediante una entidad de gestión de derechos, o varias, (por ejemplo: SGAE, Dama, VEGAP), los derechos de explotación de la obra, así como los derivados de obras derivadas, conjuntas o bases de datos, si dicha explotación pretende principalmente o se encuentra dirigida hacia la obtención de un beneficio mercantil o la remuneración monetaria privada.

   Los anteriores derechos se pueden ejercitar en todos los medios y formatos, tangibles o intangibles, conocidos o por conocer. Los derechos mencionados incluyen el derecho a efectuar las modificaciones que sean precisas técnicamente para el ejercicio de los derechos en otros medios y formatos. Todos los derechos no cedidos expresamente por el licenciador quedan reservados.

4. Restricciones. La cesión de derechos que supone esta licencia se encuentra sujeta y limitada a las restricciones siguientes:

1. Usted puede reproducir, distribuir o comunicar públicamente la obra solamente bajo los términos de esta licencia y debe incluir una copia de la misma, o su Identificador Uniforme de Recurso (URI), con cada copia o grabación de la obra que usted reproduzca, distribuya o comunique públicamente. Usted no puede ofrecer o imponer ningún término sobre la obra que altere o restrinja los términos de esta licencia o el ejercicio de sus derechos por parte de los cesionarios de la misma. Usted no puede sublicenciar la obra. Usted debe mantener intactos todos los avisos que se refieran a esta licencia y a la ausencia de garantías. Usted no puede reproducir, distribuir o comunicar públicamente la obra con medidas tecnológicas que controlen el acceso o uso de la obra de una manera contraria a los términos de esta licencia. Lo anterior se aplica a una obra en tanto que incorporada a una obra conjunta o base de datos, pero no implica que éstas, al margen de la obra objeto de esta licencia, tengan que estar sujetas a los términos de la misma. Si usted crea una obra conjunta o base de datos, previa comunicación del licenciador, usted deberá quitar de la obra conjunta o base de datos cualquier crédito requerido en el apartado 4c, según lo que se le requiera y en la medida de lo posible. Si usted crea una obra derivada, previa comunicación del licenciador, usted deberá quitar de la obra derivada cualquier crédito requerido en el apartado 4c, según lo que se le requiera y en la medida de lo posible.

2. Usted puede reproducir, distribuir o comunicar públicamente una obra derivada solamente bajo los términos de esta licencia, o de una versión posterior de esta licencia con sus mismos elementos principales, o de una licencia iCommons de Creative Commons que contenga los mismos elementos principales que esta licencia (ejemplo: Reconocimiento-CompartirIgual 2.5 Japón). Usted debe incluir una copia de la esta licencia o de la mencionada anteriormente, o bien su Identificador Uniforme de Recurso (URI), con cada copia o grabación de la obra que usted reproduzca, distribuya o comunique públicamente. Usted no puede ofrecer o imponer ningún término respecto de las obras derivadas o sus transformaciones que alteren o restrinjan los términos de esta licencia o el ejercicio de sus derechos por parte de los cesionarios de la misma. Usted debe mantener intactos todos los avisos que se refieran a esta licencia y a la ausencia de garantías. Usted no puede reproducir, distribuir o comunicar públicamente la obra derivada con medidas tecnológicas que controlen el acceso o uso de la obra de una manera contraria a los términos de esta licencia. Lo anterior se aplica a una obra derivada en tanto que incorporada a una obra conjunta o base de datos, pero no implica que éstas, al margen de la obra objeto de esta licencia, tengan que estar sujetas a los términos de esta licencia.

3. Si usted reproduce, distribuye o comunica públicamente la obra o cualquier obra derivada, conjunta o base datos que la incorpore, usted debe mantener intactos todos los avisos sobre la propiedad intelectual de la obra y reconocer al autor original, de manera razonable conforme al medio o a los medios que usted esté utilizando, indicando el nombre (o el seudónimo, en su caso) del autor original si es facilitado, y/o reconocer a aquellas partes (por ejemplo: institución, publicación, revista) que el autor original y/o el licenciador designen para ser reconocidos en el aviso legal, las condiciones de uso, o de cualquier otra manera razonable; el título de la obra si es facilitado; de manera razonable, el Identificador

Uniforme de Recurso (URI), si existe, que el licenciador especifica para ser vinculado a la obra, a menos que tal URI no se refiera al aviso sobre propiedad intelectual o a la información sobre la licencia de la obra; y en el caso de una obra derivada, un aviso que identifique el uso de la obra en la obra derivada (e.g., "traducción castellana de la obra de Autor Original,.º "guión basado en obra original de Autor Original"). Tal aviso se puede desarrollar de cualquier manera razonable; con tal de que, sin embargo, en el caso de una obra derivada, conjunta o base datos, aparezca como mínimo este aviso allá donde aparezcan los avisos correspondientes a otros autores y de forma comparable a los mismos.

4. En el caso de la inclusión de la obra en alguna base de datos o recopilación, el propietario o el gestor de la base de datos deberá renunciar a cualquier derecho relacionado con esta inclusión y concerniente a los usos de la obra una vez extraída de las bases de datos, ya sea de manera individual o conjuntamente con otros materiales.

5. Exoneración de responsabilidad.

A MENOS QUE SE ACUERDE MUTUAMENTE ENTRE LAS PARTES, EL LICENCIADOR OFRECE LA OBRA TAL CUAL (ON AN .ᴬS-IS"BASIS) Y NO CONFIERE NINGUNA GARANTÍA DE CUALQUIER TIPO RESPECTO DE LA OBRA O DE LA PRESENCIA O AUSENCIA DE ERRORES QUE PUEDAN O NO SER DESCUBIERTOS. ALGUNAS JURISDICCIONES NO PERMITEN LA EXCLUSIÓN DE TALES GARANTÍAS, POR LO QUE TAL EXCLUSIÓN PUEDE NO SER DE APLICACIÓN A USTED.

6. Limitación de responsabilidad.

SALVO QUE LO DISPONGA EXPRESA E IMPERATIVAMENTE LA LEY APLICABLE, EN NINGÚN CASO EL LICENCIADOR SERÁ RESPONSABLE ANTE USTED POR CUALQUIER TEORÍA LEGAL DE CUALESQUIERA DAÑOS RESULTANTES, GENERALES O ESPECIALES (INCLUIDO EL DAÑO EMERGENTE Y EL LUCRO CESANTE), FORTUITOS O CAUSALES, DIRECTOS O INDIRECTOS, PRODUCIDOS EN CONEXIÓN CON ESTA LICENCIA O EL USO DE LA OBRA, INCLUSO SI EL LICENCIADOR HUBIERA SIDO INFORMADO DE LA POSIBILIDAD DE TALES DAÑOS.

7. Finalización de la licencia.

1. Esta licencia y la cesión de los derechos que contiene terminarán automáticamente en caso de cualquier incumplimiento de los términos de la misma. Las personas o entidades que hayan recibido obras derivadas, conjuntas o bases de datos de usted bajo esta licencia, sin embargo, no verán sus licencias finalizadas, siempre que tales personas o entidades se mantengan en el cumplimiento íntegro de esta licencia. Las secciones 1, 2, 5, 6, 7 y 8 permanecerán vigentes pese a cualquier finalización de esta licencia.

2. Conforme a las condiciones y términos anteriores, la cesión de derechos de esta licencia es perpetua (durante toda la vigencia de los derechos de propiedad intelectual aplicables a la obra). A pesar de lo anterior, el licenciador se reserva el derecho a divulgar o publicar la obra en condiciones distintas a las presentes, o de retirar la obra en cualquier momento. No obstante, ello no supondrá dar por concluida esta licencia (o cualquier otra licencia que haya sido concedida, o sea necesario ser concedida, bajo los términos de esta licencia), que continuará vigente y con efectos completos a no ser que haya finalizado conforme a lo establecido anteriormente.

8. Miscelánea.

1. Cada vez que usted explote de alguna forma la obra, o una obra conjunta o una base datos que la incorpore, el licenciador original ofrece a los terceros y sucesivos licenciatarios la cesión de derechos sobre la obra en las mismas condiciones y términos que la licencia concedida a usted.

2. Cada vez que usted explote de alguna forma una obra derivada, el licenciador original ofrece a los terceros y sucesivos licenciatarios la cesión de derechos sobre la obra original en las mismas condiciones y términos que la licencia concedida a usted.

3. Si alguna disposición de esta licencia resulta inválida o inaplicable según la Ley vigente, ello no afectará la validez o aplicabilidad del resto de los términos de esta licencia y, sin ninguna acción adicional por cualquiera las partes de este acuerdo, tal disposición se entenderá reformada en lo estrictamente necesario para hacer que tal disposición sea válida y ejecutiva.

4. No se entenderá que existe renuncia respecto de algún término o disposición de esta licencia, ni que se consiente violación alguna de la misma, a menos que tal renuncia o consentimiento figure por escrito y lleve la firma de la parte que renuncie o consienta.

5. Esta licencia constituye el acuerdo pleno entre las partes con respecto a la obra objeto de la licencia. No caben interpretaciones, acuerdos o términos con respecto a la obra que no se encuentren expresamente especificados en la presente licencia. El licenciador no estará obligado por ninguna disposición complementaria que pueda aparecer en cualquier comunicación de usted. Esta licencia no se puede modificar sin el mutuo acuerdo por escrito entre el licenciador y usted.

Creative Commons no es parte de esta licencia, y no ofrece ninguna garantía en relación con la obra. Creative Commons no será responsable frente a usted o a cualquier parte, por cualquier teoría legal de cualesquiera daños resultantes, incluyendo, pero no limitado, daños generales o especiales (incluido el daño emergente y el lucro cesante), fortuitos o causales, en conexión con esta licencia. A pesar de las dos (2) oraciones anteriores, si Creative Commons se ha identificado expresamente como el licenciador, tendrá todos los derechos y obligaciones del licenciador.

Salvo para el propósito limitado de indicar al público que la obra está licenciada bajo la CCPL, ninguna parte utilizará la marca registrada "Creative Commons" o cualquier marca registrada o insignia relacionada con "Creative Commons" sin su consentimiento por escrito. Cualquier uso permitido se hará de conformidad con las pautas vigentes en cada momento sobre el uso de la marca registrada por "Creative Commons", en tanto que sean publicadas su sitio web (website) o sean proporcionadas a petición previa.

Puede contactar con Creative Commons en: `http://creativecommons.org/`.