

GNOME como Caso de Estudio de Ingeniería del Software Libre^{*}

Juan Jose Amor, Gregorio Robles y Jesús M. González Barahona
{jjamor, grex, jgb}@gsyc.escet.urjc.es

Grupo de Sistemas y Comunicaciones
Universidad Rey Juan Carlos (Móstoles, Madrid)

Resumen El software libre ha experimentado en los últimos años un imparable avance en la universidad, industria y administraciones públicas. Sin embargo, aún estando en boca de todos, todavía son pocos los estudios que se centran en estudiar este fenómeno de manera pormenorizada. El hecho de que se pueda acceder de manera pública y no intrusiva al código fuente y a otros subproductos del desarrollo de software libre (como son el repositorio de versiones, las listas de correo o el sistema de gestión de errores) ofrece la posibilidad de realizar análisis cuantitativos a gran escala. El proyecto GNOME es un interesante caso de estudio en este sentido, ya que además de ser uno de los entornos gráficos libres más utilizados en la actualidad ha sabido aglutinar una activa comunidad de desarrolladores a su alrededor. Este artículo es un repaso a los diferentes estudios que nuestro grupo de investigación de ingeniería de software libre ha realizado sobre GNOME desde diferentes puntos de vista como son el tamaño del software, el crecimiento de los programas, la contribución de los desarrolladores y el tipo de contribuciones, etc.

1. Introducción

El software libre a veces tiene ciertas características que lo hacen parecer mágico. Sin negar que el hecho de que cientos, quizás miles de personas distribuidos geográficamente, y que probablemente nunca se hayan visto cara a cara, consigan crear un software a la altura del estado del arte es de por sí algo que a muchos entusiasma, para la definitiva implantación del software libre es necesario conocer y entender los mecanismos que llevan al desarrollo de software libre para poder aplicarlos de manera efectiva en el futuro.

Precisamente por ello varios grupos de investigación de varias universidades vienen realizando desde hace unos años diversos estudios sobre software libre. Entre ellos, nuestro grupo de investigación en ingeniería del software libre [9] se ha especializado en realizar estudios de caracterización de proyectos de software libre dentro de la acción coordinada CALIBRE [1], impulsada por la Comisión Europea.

^{*} Este trabajo ha sido financiado en parte por la Comisión Europea, dentro de la acción coordinada CALIBRE dentro del programa IST, número de contrato 004337, por el proyecto CICYT TIN2004-07296 y por la Universidad Rey Juan Carlos con el proyecto PPR-2004-42.

1.1. El modelo de desarrollo de software libre

Aunque bien es cierto que el software libre en realidad es aquél cuya licencia permite al que lo recibe hacer uso de las cuatro libertades, hemos podido observar en los últimos años una tendencia en los proyectos de software libre que han tenido éxito (en tamaño, en número de contribuciones, en interés 'mediático') hacia un modelo de desarrollo lo más abierto posible y, en definitiva, a través de Internet.

Esto es así por dos causas: la primera es por la distribución geográfica de los desarrolladores, mientras que la segunda es para facilitar el acceso a nuevas colaboraciones (ya sea mediante código, informes de error, traducción, documentación o de cualquier otro tipo).

Por eso, además de ofrecer el código fuente, existen en paralelo diversas herramientas que se utilizan para la intercomunicación y el soporte en el desarrollo distribuido de software libre. Y debido a que para abrir al máximo el proceso de desarrollo estos datos están disponibles públicamente en Internet, también pueden ser obtenidos y analizados (incluso de manera semi-automática). Tenemos, por tanto, una ingente cantidad de artefactos que en realidad son subproducto del proceso de desarrollo que podemos analizar para conocer mejor el desarrollo de un proyecto específico.

A continuación incluimos una lista, posiblemente incompleta, de elementos y herramientas disponibles del desarrollo de software libre que nos pueden servir como fuente pública de datos sobre un proyecto:

- Código fuente: Paquetes binario y fuente (*tarballs*)
- Sistemas de control de versiones: CVS, Subversion, etc.
- Sistemas de seguimiento de fallos: Bugzilla, etc.
- Documentación: Páginas man, documentación en Docbook, etc.
- Listas de correo y foros públicos
- Estadísticas de uso (como la Debian Popcon [10])
- Encuestas: como por ejemplo FLOSS [7]

Todas estas fuentes de datos nos serán útiles para estudiar la actividad de un proyecto: desde estudios de evolución en tamaño o en el uso de lenguajes de programación hasta la actividad de los desarrolladores a partir de lo que se publica en ciertas listas de correo o los registros que tienen lugar en los sistemas de control de versiones.

En este artículo veremos ejemplos de resultados de aplicar las medidas sobre las fuentes públicas de datos disponibles en el proyecto GNOME. En primer lugar, veremos los resultados que se obtienen al realizar un simple análisis del número de líneas de código fuente de algunas aplicaciones de GNOME. A continuación, presentaremos los resultados que se obtuvieron analizando los datos disponibles en el servidor CVS de GNOME. En la siguiente parte, veremos cómo la combinación de varias fuentes de datos (CVS, listas de correo y gestor de errores a la vez) pueden sugerirnos nuevos e interesantes resultados. Por último, presentaremos las conclusiones de este artículo.

2. Estudios del código fuente de GNOME

El primer análisis que vamos a ver se ha hecho con los *tarballs* de código fuente de los proyectos. Para ello, hemos utilizado métricas basadas en contar las líneas de código fuente de los paquetes. Para contar las líneas de código fuente pueden utilizarse, desde simples comandos Unix *wc* (contando directamente las líneas del fichero de texto) o buscando soluciones más avanzadas como scripts que detecten los comentarios (basados en *awk*) o incluso, utilizando contadores como SLOCCount [13] que incluyen heurísticas bastante sofisticadas para contar identificar los ficheros que contienen código y contar las líneas de código incluso en los más variados lenguajes de programación.

Usando SLOCCount, por ejemplo, podemos analizar la evolución del tamaño de las versiones de GTK+ o Gnumeric en el tiempo (ver figura 1).

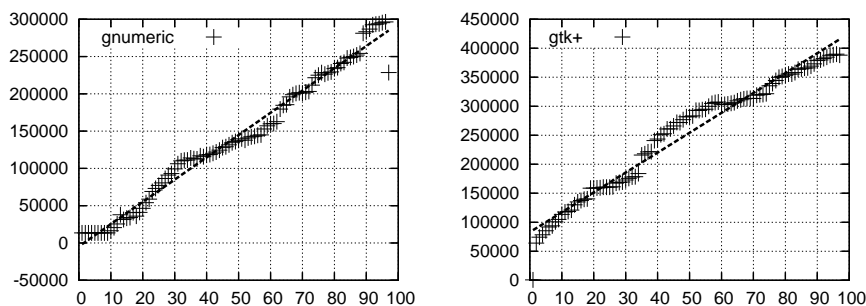


Figura 1. Evolución del número de líneas de código de Gnumeric (izquierda) y GTK+ (derecha). El eje horizontal viene dado por el tiempo (medido en meses desde que el proyecto utiliza CVS), mientras que en el eje vertical se puede ver el tamaño en líneas de código fuente. La línea recta es el ajuste lineal, que como se puede comprobar es bastante acertado en ambos casos.

Más adelante veremos que esta medida también puede hacerse a partir de los logs que se pueden obtener de un repositorio CVS. Sin embargo, los estudios de evolución de líneas de código son más precisos cuando se realizan con SLOCCount porque esta herramienta distingue lenguajes y evita contar las líneas de ficheros tales como documentos o *LEEMEs*.

Una de las cosas que se suele observar en la mayoría de los proyectos de software libre de cierto tamaño como en el caso de GNOME, es que la curva de crecimiento tiende a la linealidad, y en algunos casos se ha observado crecimiento super-lineal como, por ejemplo, en Linux [8]. Esto es, que el crecimiento del software conforme pasa el tiempo rara vez se ralentiza y en ocasiones se acelera, siendo éste un modelo de crecimiento que no se suele observar en proyectos que siguen un modelo de desarrollo privativo, que suelen presentar una curva de crecimiento sub-lineal, similar a una raíz cuadrada. Aunque todavía es pronto

para sacar conclusiones, esto puede deberse a que la forma de gestión y de organización de los proyectos de software libre es más eficiente que la de los entornos de creación de software clásico.

Quizás una de los métodos más interesantes que se pueden aplicar a partir del número de líneas de código sean las estimaciones de costes con técnicas como COCOMO [4]. La herramienta SLOCCount devuelve de forma predeterminada el coste estimado del proyecto analizado de acuerdo con los cálculos del modelo básico de COCOMO. La tabla 1 nos muestra el esfuerzo estimado que se necesitaría para crear algunos proyectos desde cero en hombres-año, así como el tiempo de desarrollo mínimo en años. Con estos dos resultados, que obtiene el método COCOMO aplicando algunos cálculos al número de líneas de código fuente producidas, es fácil obtener el esfuerzo en horas totales de trabajo estimado, y traducir éste a cantidades monetarias que nos darían el coste de desarrollo de un proyecto.

Software	Fecha	Líneas	Esfuerzo	Tiempo desarrollo	Coste Total
Dia	Agosto 2004	122,722	31.23 a-h	1.98 años	\$ 4,218,868
Evolution	Marzo 2005	207,507	54.19 a-h	2.44 años	\$ 7,320,252
Gnumeric	Enero 2005	294,547	78.28 a-h	2.81 años	\$ 10,574,357
Gtk+	Abril 2005	418,927	113.31 a-h	3.23 años	\$ 15,306,888

Cuadro 1. Coste estimado de proyectos de GNOME según COCOMO. La fecha indica el mes en el que se han tomado los datos de tamaño (en número de líneas de código fuente). El esfuerzo da el número de años-hombre necesarios para construir un software de ese tamaño, mientras que el tiempo de desarrollo es el tiempo en años en el que se estima que se puede crear el software. Finalmente el coste total en dólares americanos viene dado por multiplicar el esfuerzo por el salario de los desarrolladores, así como otros costes que hay que tener en cuenta en el desarrollo de software (hardware, personal de apoyo, etc.).

Otro resultado interesante que nos puede proporcionar el estudio de los paquetes de código fuente, es la proporción en el uso de los lenguajes de programación y cómo evolucionan éstos. En el caso de GNOME hemos encontrado que la importancia del lenguaje C sigue haciendo despreciable el crecimiento de otros lenguajes. Por ejemplo, la figura 2 nos muestra la proporción de lenguaje C en Evolution frente al resto (principalmente, scripts en shell y perl). Es de esperar, sin embargo, que otros lenguajes como Python vayan ganando terreno en el futuro.

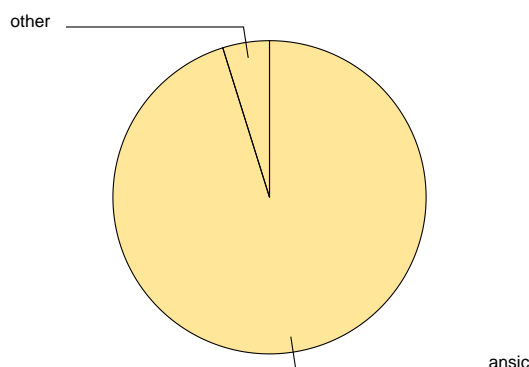


Figura 2. Lenguajes de programación en Evolution 2.1.3. Se puede observar cómo el lenguaje predominante es C, mientras que la presencia de otros lenguajes -shell, perl, etc.- es pequeña (en torno al 5%).

3. El repositorio de versiones como fuente de datos sobre GNOME

GNOME utiliza un sistema de control de versiones CVS, ya que facilita el desarrollo distribuido y es utilizado por los usuarios avanzados para obtener la versión más reciente del código, incluso antes de que se publique oficialmente. En esta parte veremos diversos resultados obtenidos del análisis del CVS de GNOME. Los análisis han sido realizados mediante la herramienta CVSanaly, desarrollada por el grupo de Ingeniería del Software Libre de la Universidad Rey Juan Carlos [11] [6]. Los resultados para GNOME se pueden consultar de manera pública en Internet [2], existiendo la posibilidad de hacerlo por módulo e incluso por desarrollador.

Como completo sistema de control de versiones, CVS almacena las diferentes versiones de cada fichero, así como información sobre el desarrollador que hizo los cambios y la fecha de éstos. De este modo, es posible utilizar estos datos para realizar análisis relacionados con la evolución del software o la participación de los desarrolladores.

La herramienta CVSanaly obtiene la mayor parte de la información acerca del proyecto de software libre a partir de los históricos que almacena (*logs*). En estos históricos se almacenan datos de cada transacción: líneas cambiadas en qué fichero (*diffs*), fecha y hora del cambio y desarrollador que lo hizo.

3.1. Desigualdad en las contribuciones al proyecto

Un hecho habitual es que la mayor parte del trabajo la realiza una minoría de desarrolladores. Una forma de establecer una medida del grado de *desigualdad*

en las aportaciones al proyecto por parte de los desarrolladores, es expresarlo mediante el coeficiente de Gini [5].

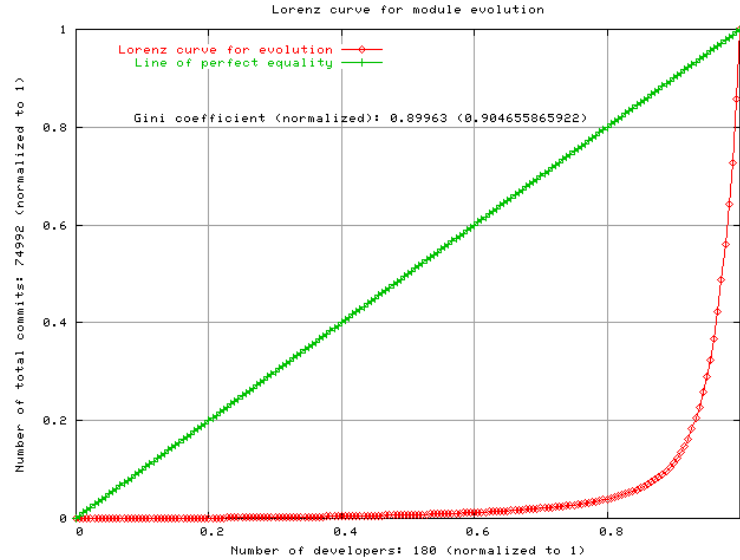


Figura 3. Curva de Lorenz y coeficiente de Gini para Evolution (datos de abril de 2004). El área entre las dos curvas nos da el coeficiente de Gini, que es 0.899.

En la figura 3 mostramos la gráfica de desigualdad para el caso de Evolution. La línea recta (verde) nos indica el caso de igualdad, que se daría si todos los desarrolladores hubieran contribuido la misma cantidad; mientras que la curva (en rojo) indica la distribución real de aportaciones (de manera agregada). El coeficiente de Gini es una medida de la relación entre las áreas de la recta y la curva con los ejes, que tiende a uno conforme la desigualdad sea mayor. Los proyectos de software libre, y Evolution no es una excepción, suelen tender a tener un coeficiente de Gini bastante alto, lo que implica que hay una gran desigualdad en cuanto a las contribuciones. Algunos autores hablan de que se cumple la Ley de Pareto: el 20 % más activo realiza el 80 % del trabajo, mientras el 80 % restante sólo es responsable del 20 % restante.

3.2. Contribución de cada desarrollador

Las posibilidades de un análisis sobre el CVS no terminan aquí. Es posible analizar las contribuciones a cada proyecto (qué desarrolladores colaboran) o las contribuciones de cada desarrollador en cada proyecto (entendiendo por proyecto cada uno de los módulos del repositorio CVS).

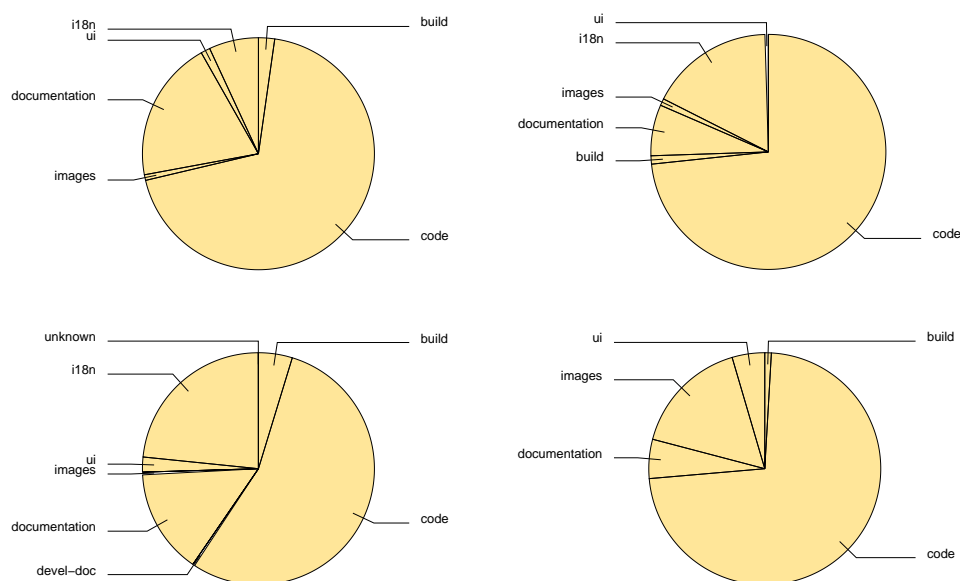


Figura 4. Tarta de contribución de algunos desarrolladores a GNOME según tipos de fichero. De izquierda a derecha y de arriba abajo, los committers seleccionados: *miquel*, *rodrigo*, *carlos* y *acs*.

Por ejemplo, se muestran a continuación en la figura 4 las tartas del tipo de contribuciones de cuatro desarrolladores hispanohablantes muy conocidos. Cada porción de la tarta viene dada por su aportación en número de commits a tipos de ficheros específicos, que van desde ficheros de código fuente (*code*), documentación (*documentation*), traducción (*i18n*), imágenes (*images*), interfaz gráfica de usuario (*ui*) y los que no hemos sabido identificar (*unknown*).

Puede verse que el trabajo del desarrollador *carlos* destaca por su contribución más importante que en el resto, a los trabajos de traducción (*i18n*) y documentación, de modo que la parte dedicada solamente al desarrollo es menor que en los demás casos.

3.3. Generaciones de desarrolladores

Otro de los resultados que se obtienen del análisis de la mayoría de los proyectos de GNOME, es que el liderato de éstos se va heredando. En otras palabras, no existe una dependencia de un único desarrollador que hace que si éste abandona el proyecto, éste último se estanque. Por contra, generalmente nos encontramos con generaciones sucesivas de desarrolladores que toman el proyecto durante un tiempo, lo hacen evolucionar y después de un tiempo lo dejan. Esta hipótesis se trata en profundidad en [3].

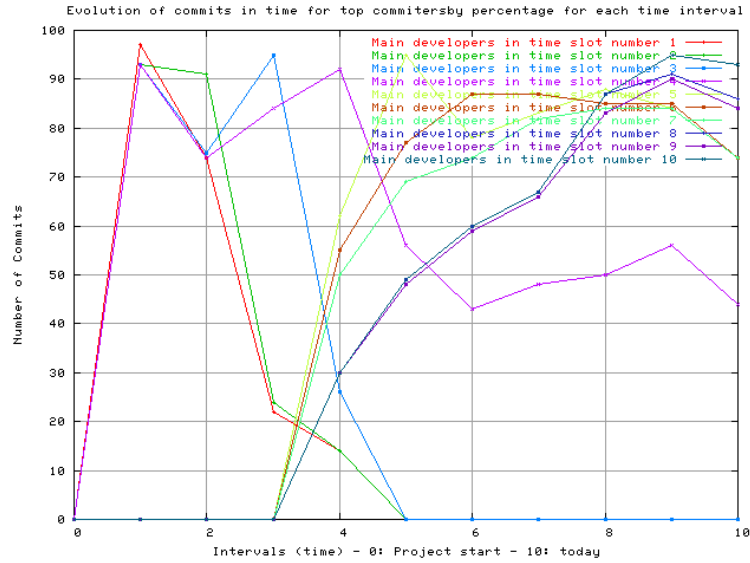


Figura 5. Generaciones de desarrolladores en Evolution. El eje horizontal viene dado por todo el tiempo del proyecto (dividido en diez partes iguales o 'slots'), mientras el horizontal por el porcentaje de commits en cada parte temporal. Para cada 'slot' se ha identificado el grupo más activo ('core') y se ha estudiado su evolución en el tiempo en los demás 'slots'. De esta manera, podemos ver que hay varias 'generaciones' de grupos más activos en el tiempo para el proyecto Evolution.

Para demostrar este hecho, hemos creado una metodología que muestra gráficamente la existencia de generaciones. Si dividimos el tiempo de desarrollo en intervalos, e identificamos al grupo de desarrolladores más activo en cada uno de estos intervalos, podemos representar la contribución de cada grupo en cada uno, dando lugar a un gráfico como el de la figura 5. Esta figura corresponde al desarrollo del proyecto Evolution, cuyo tiempo de desarrollo hasta el momento de la extracción de los datos se dividió en diez intervalos.

Observando la figura 5, podemos ver que en los primeros intervalos de tiempo hay un grupo muy activo que lidera el proyecto. Este grupo va cediendo en actividad a otros grupos que van apareciendo y pasando a ser los que realizan la mayor parte de las contribuciones al mismo.

4. Uso de varias fuentes de datos simultáneas

Por último, es posible realizar análisis más avanzados de la información que pueden facilitarnos varias de las fuentes públicas de datos vistas al principio de este artículo. Hasta ahora, hemos realizado análisis que incluían datos de una

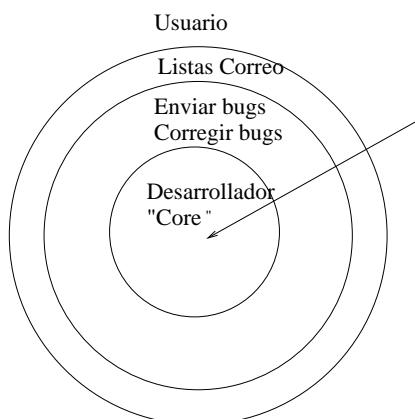


Figura 6. Modelo de la cebolla. Muestra esquemáticamente las etapas por las que pasa un colaborador que se incorpora a un proyecto como GNOME.

única fuente. Pero también podemos considerar varias fuentes simultáneamente como en el caso que se describe en el siguiente apartado.

4.1. Trayectoria del desarrollador

En el caso de GNOME, es interesante el análisis realizado de la *trayectoria seguida por los desarrolladores* para entrar en el proyecto. Veremos que se han utilizado tres fuentes de datos: el CVS, las listas de correo electrónico y el Bugzilla (sistema de seguimiento de fallos).

Básicamente, se trataba de comprobar, mediante los datos disponibles, si los desarrolladores se incorporaban al proyecto GNOME siguiendo un lógico *modelo cebolla* (ver figura 6).

La lógica de este modelo consiste en que cada desarrollador se va incorporando de forma natural a cada uno de los roles de la *cebolla* por capas, empezando por la más externa. En tal caso tendríamos,

- Cada desarrollador empieza como simple usuario “pasivo”. En esta etapa, visitará ocasionalmente la web del proyecto y puede que en algún momento se interese por consultar las listas de correo, si bien no participa aun y no puede trazarse su comportamiento.
- Cuando el desarrollador se va familiarizando con el proyecto, se suscribe a listas de correo y comienza a participar. Aquí el futuro desarrollador comienza a entender cómo funciona el proyecto (GNOME en este caso) por dentro.
- Con cierto conocimiento, ya está en condiciones de detectar por su cuenta los fallos del software que va probando y los envía. También, su conocimiento del proyecto le permite empezar a participar en fallos enviados por otras personas y sugerir sus propias correcciones.

- En otra etapa posterior, posiblemente el desarrollador acabará teniendo su propia cuenta con acceso de escritura al CVS para poder contribuir directamente al código.
- Por último, si un desarrollador colabora durante cierto tiempo con cierta intensidad, añadiendo código al CVS, podemos considerar que ha acabado formando parte del grupo central de desarrolladores (“core”).

Mientras el desarrollador no llegue como mínimo a la etapa de participación en las listas de correo, no podemos trazar su actividad. Pero a partir de ese momento, toda la actividad (en las listas, en el CVS) es perfectamente trazable. Y puede comprobarse si la incorporación del desarrollador al proyecto ha seguido este modelo o no.

En [12] se ha utilizado el proyecto GNOME como caso de estudio para evaluar la viabilidad del “modelo cebolla”. Se seleccionó un conjunto de desarrolladores que cumpliera ciertos criterios (como que lleven un tiempo suficiente con acceso de escritura al CVS y tengan un número elevado de escrituras (envíos), o que se compruebe que su contribución es principalmente código fuente, evitando aquellos cuya participación es de traductores o documentalistas).

Como resultado, se obtuvo que había un número apreciable de desarrolladores -generalmente voluntarios- que se incorporaban siguiendo este modelo por capas; desde las capas más externas de la “cebolla” hasta formar parte del núcleo necesitaban alrededor de tres años en media. Por otro lado, había otro grupo no despreciable de personas que se incorporaban “repentinamente” a la actividad. Se comprobó que estos últimos se corresponderían con el perfil de empleados en empresas que colaboran en el desarrollo de GNOME como Ximian o SUN.

5. Conclusiones

En este artículo hemos presentado brevemente la necesidad de que el software libre sea estudiado de manera ingenieril. Esto es una tarea difícil debido a la distribución de desarrolladores y a su carácter mayoritariamente voluntario. Sin embargo, la disponibilidad de gran cantidad de información en Internet relacionada con los proyectos de software libre, generalmente información subproducto del desarrollo de un proyecto, puede ser tratada y analizada convenientemente y se pueden obtener resultados muy interesantes.

Entre los análisis que hemos mostrado contamos con análisis de código fuente, del repositorio de versiones y uno con fuentes combinadas. Así, hemos podido ver la evolución del tamaño de Evolution y Gnumeric a partir del cual hemos podido realizar una estimación de coste utilizando el método COCOMO. Por otro lado, a partir del CVS disponemos de información sobre la desigualdad en cuanto a contribuciones de los desarrolladores que forman parte de los proyectos y del tipo de contribuciones que éstos realizan. También se ha presentado una manera de demostrar que los proyectos de software se sustentan sobre varias generaciones en el tiempo, así como el hecho de que los voluntarios sigan un proceso gradual de integración en el proyecto, mientras que los profesionales parecen seguir patrones diferentes.

6. Agradecimientos

Nos gustaría agradecer el apoyo técnico y humano del resto del equipo de CALIBRE de la Universidad Rey Juan Carlos: Israel Herraiz, Álvaro Navarro, Diego Barceló, Jorge Gascón y Teo Romera. También nuestro agradecimiento a los cuatro revisores anónimos de la II GUADEC-es por sus comentarios y sugerencias.

Referencias

1. Coordination Action in Libre Software.
<http://www.calibre.ie/>
2. Gregorio Robles, Jesús M. González-Barahona: “CVS Analysis for the GNOME project”
<http://libresoft.urjc.es/cvsanal/gnome2-cvs/>
3. Jesús M. González-Barahona y Gregorio Robles “Unmounting the ”code gods” assumption.” Proceedings of the Fourth International Conference on eXtreme Programming and Agile Processes in Software Engineering
<http://libresoft.urjc.es/html/downloads/xp2003-barahona-robles.pdf>
4. Barry Boehm: “Software Engineering Economics.” Prentice Hall, 1981
5. Conrado Gini: “On the Measure of Concentration with Espacial Reference to Income and Wealth.” Cowles Comission. 1936
6. Israel Herraiz, Gregorio Robles y Jesús M. González-Barahona: “CVSAnal: Una herramienta libre para el análisis de repositorios CVS.” IV Jornadas Andaluzas de Software Libre. Algeciras, 2004
7. Rishab Aiyer Ghosh, Gregorio Robles y Ruediger Glott: “Software Source Code Survey (Free/Libre and Open Source Software: Survey and Study).” International Institute of Infonomics. University of Maastricht. 2002.
<http://www.infonomics.nl/FLOSS/report>
8. Michael W. Godfrey y Qiang Tu: “Evolution in Open Source Software: A Case Study.” Proceedings of the International Conference on Software Maintenance (ICSM 2000). San Jose, California. 131-142. 2000.
<http://plg.uwaterloo.ca/~migod/papers/icsm00.pdf>
9. Libre Software Engineering at Universidad Rey Juan Carlos
<http://libresoft.urjc.es/>
10. Debian Popularity Contest
<http://popcon.debian.org/>
11. Gregorio Robles, Stefan Koch y Jesus M. Gonzalez-Barahona: “Remote analysis and measurement of libre software systems by means of the CVSAnalY tool.” Proceedings of the 2nd ICSE Workshop on Remote Analysis and Measurement of Software Systems (RAMSS) Edinburg, Scotland, UK, 2004.
<http://libresoft.urjc.es/html/downloads/cvsanaly-icse.pdf>
12. Gregorio Robles, Israel Herraiz y Jesus M. Gonzalez-Barahona: “Program comprehension and membership integration in large libre software projects: A case study.” (Pending publication) Preliminary results:
<http://libresoft.urjc.es/html/downloads/ArtifactComprehension-herraiz-robles.pdf>
13. David A. Wheeler: “More Than a Gigabuck: Estimating GNU/Linux’s Size.”
<http://www.dwheeler.com/sloc/redhat71-v1/redhat71sloc.html>