Ingeniería del Software Libre. Una visión alternativa a la ingeniería del software tradicional

Gregorio Robles
Universidad Rey Juan Carlos

grex@scouts-es.org

Historial de revisiones
Revisión 0.94 - versión V congreso Hi-10 de octubre de 2002
spaLinux

La ingeniería del software no ha podido madurar convenientemente debido a las prácticas propietarias que predominan (todavía) en el mundo del software comercial. En cambio, el software libre y los métodos de desarrollo asociados con frecuencia a proyectos de software libre brindan una gran oportunidad para que la ingeniería del software se encamine hacia lo que realmente pretende ser: una actividad ingenieril aplicada a las técnicas de desarrollo, funcionamiento y mantenimiento del software. Partiendo de esta idea, esta ponencia pretende presentar un enfoque cuantitativo basado en la extracción, procesado y análisis de información proveniente del software libre. Además de ver las aportaciones y posibilidades que esta nueva visión puede ofrecer a la ingeniería del software y al software libre, se presentará el estado del arte en este campo.

Tabla de contenidos

1. Introducción

Desde hace cuatro décadas, la ingeniería del software se ha venido consolidando como una rama importante dentro de la informática en busca de métodos de desarrollo y técnicas que permitan producir software de gran calidad con unos recursos limitados. Según la definición del IEEE, la ingeniería del software es "un enfoque sistemático y cuantificable al desarrollo, operación (funcionamiento) y mantenimiento del software: es decir, la aplicación de la ingeniería del software". [IEEE 1993]

A pesar de que la ingeniería del software ha conseguido indudablemente notables éxitos, también es cierto que ha sucumbido a lo que se ha venido a llamar la "crisis del software". Prueba de ello es que a día de hoy todavía sigue sin ser posible cuantificar con exactitud los plazos, costes, recursos humanos y técnicas que lleven a un desarrollo exitoso del software, tal y como otras ramas de la ingeniería en otros campos sí han sido capaces de hacer. Es más, incluso en algunos puntos de la ingeniería del software se puede observar una tendencia a retomar viejos caminos bajo nuevas fórmulas, como podemos ver con la incipiente expansión de las técnicas de programación evolutiva que se basan en gran parte en principios y técnicas conocidos y usados en la década de los 70. Argumentar que la ingeniería del software se encuentra estancada y falta de ideas es una consideración que, por lo tanto, podemos tomar como muy válida.

2. La crisis de la ingeniería del software tradicional

Uno de los grandes problemas de la ingeniería del software ha sido y es que no ha sabido adaptarse consecuentemente a su propia definición. Esto es algo que se puede considerar como una especie de traición a sí misma, a sus propios fundamentos. El enfoque sistemático y cuantificable ha tenido siempre como barreras las propias de las formas en las que el software se ha desarrollado y distribuido. El formato binario del software, la opacidad en los modelos de negocios, los secretos y barreras comerciales se encuentran entre las principales causas que han imposibilitado estudios cuantitativos y cualitativos a gran escala del software cuyos resultados pudieran ser verificados sistemáticamente por equipos de investigación independientes. Las "verdades" que han sido enunciadas son, con frecuencia, experiencias puntuales que han sido generalizadas y dadas por válidas ante la falta de alternativas. La propia forma de desarrollar, distribuir y comercializar software ha sido la que ha llevado a la ingeniería del software a la crisis.

Y es aquí donde el software libre puede dar nuevos aires a la ingeniería del software. Desde hace más de una década, el software libre ha venido experimentando un gran auge en cuanto a uso, aceptación y, por supuesto, desarrollo. La implantación de Internet junto con las características de las licencias que "invitan" a todo el mundo a formar parte del

equipo de desarrollo, han propiciado que a día de hoy no sólo podamos contar con el código fuente (un gran avance para su estudio en contraposición al del software propietario), sino tomar medidas de los archivos de las listas de correo donde viene plasmada la comunicación del proyecto, los repositorios de versiones gracias a los cuales podemos ver la evolución, etc. De todas estas fuentes se puede extraer una gran cantidad de información de interés, en la mayoría de casos incluso de manera automática. Por tanto, el software libre ofrece la oportunidad de conocer más a fondo el proceso de concepción de software, aportándonos nuevas evidencias y experiencias.

Podemos concluir que la apertura tanto del código como de la información asociada al proceso de desarrollo que ofrece el software libre es clave para que pueda ser analizado, estudiado y discutido de manera totalmente contrastable y abierta.

2.1. Ingeniería del software tradicional e ingeniería del software libre

Este nuevo enfoque de la ingeniería del software no es ni mucho menos contradictorio con el tradicional que todos conocemos; es más, se puede considerar que ambos son compatibles, incluso complementarios. Esto viene a significar que la nueva perspectiva puede ser utilizada para comparar dos técnicas de programación diferentes, ayudando a sacar conclusiones cuantitativas (e incluso cualitativas). Es muy probable que estos resultados puedan ayudar a mejorar esas técnicas, al mismo tiempo que nuevas técnicas pueden implicar nuevas medidas. No se trata de decir que lo que conocemos en la actualidad como ingeniería del software no sirve para nada o está condenado al fracaso, sino de abrir nuevos caminos y nuevas formas de proceder que, en paralelo con las ya existentes, amplíen nuestro conocimiento sobre los métodos de creación, funcionamiento y mantenimiento del software.

Y es que, como se discutirá a continuación, mediante el análisis del software libre se ganan una serie de factores que difícilmente ha podido conseguir la ingeniería del software tradicional.

El primero de ellos es la vertiente temporal que se añade al análisis. Y es que no se puede olvidar que el proceso de creación de software ha cambiado según han cambiado los paradigmas tecnológicos, de educación, de programación, etc. Algo que se enunció hace 30 años puede ser muy válido en la actualidad (o no), aunque no cabe duda de que es muy probable que necesite ser adaptado e incluso mejorado. De la evolución histórica se puede sacar mucha e interesante información, y no solamente de carácter técnico. Para muchas decisiones es de gran importancia saber los lenguajes de programación en alza, la evolución en cuanto a colaboradores de ciertos proyectos (por ejemplo, de proyectos que se dediquen a crear aplicaciones p2p), etc. Mediante un análisis temporal continuo estaremos en disposición de tener un termómetro permanente de lo que está ocurriendo en el mundo del software (libre).

Por otro lado, el análisis de software libre no plantea problemas de granularidad. La ingeniería del software se ha basado con frecuencia en el análisis de unos pocos proyectos de software debido en gran parte a la facilidad de acumular experiencias en entornos corporativos propios. COCOMO, un modelo de cálculo de costes y tiempos para proyectos software, es un claro ejemplo de esto, ya que fue creado en un departamento de la NASA a raíz de la experiencia en poco más de un centenar de proyectos. Mediante el uso de información extraída del software libre estamos capacitados para obtener un (nuevo) modelo del tipo "COCOMO", pero mucho más global (o dependiente del lenguaje de programación o de otras circunstancias) a la vez que más ajustado a la realidad actual. Incluso,

añadiéndole la vertiente temporal como se ha indicado en el párrafo anterior, podríamos ver la evolución de este modelo en el tiempo.

Tomando como analogía las ciencias económicas, podríamos decir que con el método COCOMO estamos hablando de un microanálisis. Por otra parte deberíamos tener, por tanto, el macroanálisis, que trataría de cuantificar y estudiar el software a gran escala. Este análisis ha sido históricamente ignorado por la ingeniería del software tradicional y es el software libre el que da la posibilidad de que se pueda ver la evolución de muchos parámetros. Así se facilitará información relevante a la hora de tomar decisiones en entornos empresariales y en proyectos de software libre.

Gracias a la ingeniería del software libre será posible, por consiguiente, medir un proyecto dentro de entornos cerrados (microanálisis) y globales (macroanálisis) de manera simultánea, lo que puede ser de gran ayuda para medir la salud del mismo. Por ejemplo, se podría analizar Evolution, el cliente de correo más completo del entorno GNOME, dentro del propio proyecto GNOME que engloba unos 200 proyectos y mil colaboradores y dentro del resto de software (libre) en general. Esto nos dará información desde dos puntos de vista que, a buen seguro, enriquecerán el enfoque.

2.2. Software libre e ingeniería del software libre

Cabe añadir, sin embargo, que la ingeniería del software libre no sólo pretende ser beneficiosa para la ingeniería del software; también lo pretende ser, en gran medida, para el software libre. Si a día de hoy los cálculos de plazos y de costes en los proyectos de software estudiados tradicionalmente (en su gran mayoría de software propietario) son difícilmente cuantificables, dentro del mundo del software libre son prácticamente utópicos.

En cierta medida, la ingeniería del software libre pretende desposeer de esa "magia" que parece que es intrínseca a los desarrollos de software libre y cuantificar unos parámetros que nos permitan predecir con exactitud costes, plazos y recursos humanos. Como consecuencia, aunque podemos considerar que en la actualidad el software libre adolece de estos métodos en contraposición a las formas de desarrollo tradicionales, también es cierto que, por los motivos que se están desarrollando en esta ponencia, no le falta precisamente potencial para que esta situación cambie en el futuro.

La comparación entre diferentes proyectos de software libre, así como el análisis de aquéllos que tienen éxito también debe servir para que la experiencia en la creación de software libre quede plasmada en conocimiento para proyectos futuros. En este sentido, paradójicamente, la ingeniería del software libre puede ser la puerta de entrada de métodos auspiciados por la ingeniería del software tradicional que se muestren exitosos o eficaces en proyectos de software libre.

3. Hacia un sistema de medición y análisis de software libre

La medición y el análisis de datos relacionados con el desarrollo de software libre se hace imprescindible para alcanzar los objetivos que la ingeniería del software libre persigue. Además, es de capital importancia que los procesos que se desarrollen puedan ser verificados por terceros, por lo que las herramientas utilizadas deberían tener una licencia de software libre.

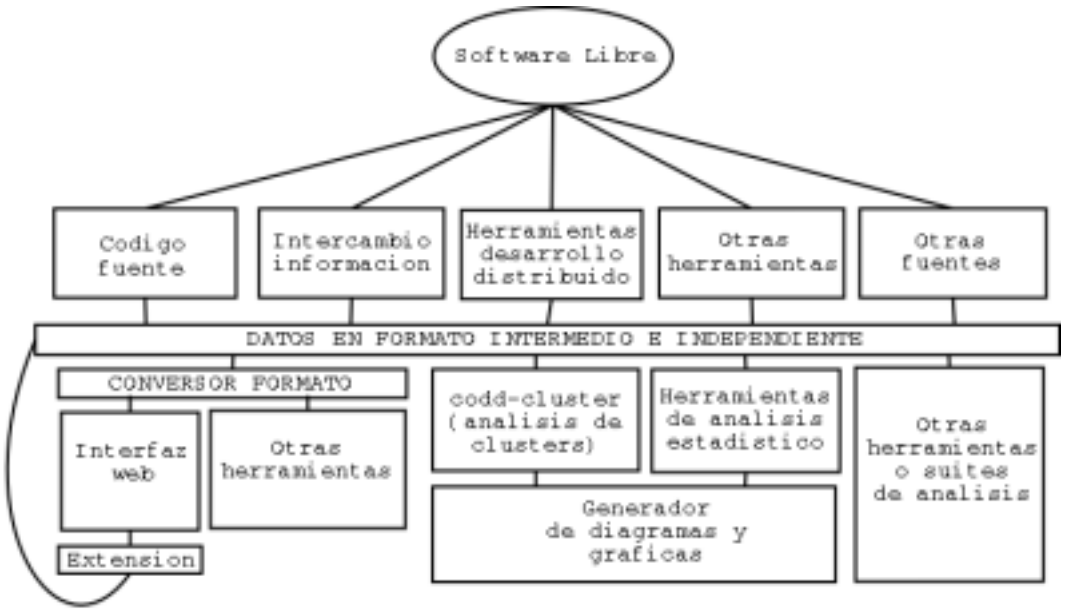
Para hacer la medición y el análisis lo más versátil posible, se han diferenciado varias etapas, tal y como se puede observar en la figura al final de este punto.

Es importante denotar que todos los datos provienen directa o indirectamente de parámetros y características de software libre. Esto se debe a que suelen ser accesibles gracias a que se tiende a seguir un modelo de desarrollo lo más abierto posible. Mediante el uso de varias herramientas independientes entre sí se pretende obtener los datos de diferentes fuentes. Es frecuente que un mismo parámetro se pueda obtener de varias formas diferentes, lo que puede ser útil para la realización de comprobaciones.

Los resultados de las diferentes herramientas se almacenarán en un formato intermedio e independiente de las mismas. De esta forma, la segunda fase se facilita sobremanera, ya que los datos guardados en ese formato intermedio podrán analizarse convenientemente por medio de herramientas realizadas al efecto o, si es necesario, pueden ser fácilmente convertidos a otro tipo de formatos.

Mientras que el objetivo de la primera fase es extraer el mayor número de parámetros cuantificables, la segunda fase es un amplio terreno aún por explorar; veremos que es ahí donde tienen cabida desde simples análisis de los datos hasta la utilización de complejos algoritmos estadísticos que permitan ir conociendo más a fondo el software libre.

Antes de mostrar pormenorizadamente las herramientas existentes para las cada una de las fases, se debe mencionar que aún cuando la arquitectura completa del sistema puede parecer compleja, esto no es así. Existe una gran modularidad e independencia y el "pegamento" que da sentido a todo esto es la capa donde viene especificado que los datos se almacenarán en un formato intermedio e independiente. Esto quiere decir que una aplicación de extracción de parámetros de código fuente es totalmente independiente de otra que toma datos debidos al desarrollo distribuido. Es más incluso lo es de otra que también se encarga de estudiar el código fuente. Lo que debe preocupar a las aplicaciones es ofrecer sus resultados en el formato intermedio, haciendo uso de filtros si es conveniente realizar conversiones a otros formatos.



(esta imagen se encuentra ampliada - del tamaño de un folio - al final del artículo)

4. Extracción de datos (Primera fase)

El primer paso engloba agrupar, ordenar y analizar convenientemente el código fuente y los flujos de información existentes en los proyectos de software libre. La finalidad principal es conseguir que todo esto se haga lo más automáticamente posible. En realidad, se pretende recabar todo tipo de información para poder ser analizada y estudiada detenidamente con posterioridad. Como se ve, se trata de un proceso iterativo, ya que los resultados de los primeros análisis nos dirán por dónde seguir buscando y cuáles deben ser los siguientes pasos lógicos dentro del estudio del software libre.

4.1. Código Fuente

El código fuente es, con diferencia, el mayor continente de información en cuanto al desarrollo de proyectos de software libre se refiere. De él se pueden extraer no sólo parámetros globales como el tamaño, el número de ficheros, sino que puede ser investigado con la finalidad de encontrar parámetros de participación (número de desarrolladores), de programación (lenguaje de programación, además de ofrecer la posibilidad de utilizar diferentes métricas de programación), de líneas de código (tanto lógicas como físicas), número de comentarios, etc. etc.

Una de las primeras aproximaciones existentes a día de hoy es el cálculo del número de líneas físicas de proyectos de software libre y el uso del modelo COCOMO (clásico) para obtener resultados en cuanto al tiempo, al coste y a los recursos humanos necesarios para su desarrollo. Evidentemente, este primer análisis se encuentra en una fase bastante primitiva, pero la correlación con otras fuentes permitirá mejorar (y/o adaptar) los resultados en el futuro.

Existe en la actualidad una gama de herramientas que permiten realizar parcialmente estas tareas. SLOCcount reconoce el lenguaje de programación utilizado, cuenta líneas de código (físicas) y hace una estimación aproximada de los costes y el tiempo que el desarrollo ha supuesto mediante el uso de COCOMO. Por su parte, CODD recorre todos los ficheros con código fuente y asigna autoría a las personas que lo han desarrollado. A estas dos aproximaciones habría que sumar la información que pueden proporcionar diferentes métricas de software.

4.2. Intercambio de información directa entre desarrolladores

El intercambio más importante de información no incluido en el código corre a cargo de listas de correo electrónico, canales IRC y documentación. En el caso de las listas de correo-e, los mensajes son almacenados en archivos que deben ser analizados. En cuanto a la documentación y al IRC todavía no está muy claro lo que buscamos y sobre todo, cómo hacerlo de forma automática.

Para nuestro acometido existe una herramienta llamada MailListStats que analiza estadísticamente la participación en las listas de correo. Existen también herramientas que analizan las bitácoras de los canales IRC.

4.3. Herramientas de desarrollo distribuido

El desarrollo de software libre se basa en gran parte en unas herramientas que permiten sincronizarse con el trabajo de los diferentes desarrolladores del proyecto, de manera que la distribución geográfica no suponga un problema. Los sistemas de control de versiones y los gestores de erratas (también usados ocasionalmente para tareas de planificación) se han convertido en herramientas imprescindibles para proyectos de software libre grandes, y no tan grandes. Estos sistemas suelen registrar las interacciones con los desarrolladores y, por tanto, una vez que se consiguen estos registros puede monitorizarse de manera bastante sencilla todo el proceso de desarrollo.

El análisis de las bitácoras del CVS mediante la aplicación cvstat2 permite obtener información de la interacción de los desarrolladores con el repositorio. El análisis permite obtener datos por desarrollador, por proyecto y por fichero y todo esto de manera temporal. cvstat2 tiene integrado un interfaz web que permitirá por una parte que muchos proyectos puedan instalarlo fácilmente y ver sus estadísticas y por otra que exista una extracción de datos distribuida.

La otra fuente de información interesante en este punto es obteniendo datos de los sistemas de control de erratas como BugZilla. Del mismo se pueden ver la evolución de las erratas y las diferentes interacciones que existen.

4.4. Otras Herramientas

Además de las herramientas ya presentadas, existen otro tipo de herramientas que no entran en la clasificación que hemos seguido y que, por tanto, serán mostradas a continuación.

SFparser es una herramienta que rastrea sitios web que tengan el software de SourceForge instalado. Su acometido es tomar toda la información posible que existe, como por ejemplo los desarrolladores que participan, la categorización del proyecto, si tiene listas de correo y/o usa un repositorio CVS, etc. La misma idea subyace tras el guión FMparser, que realiza idénticas extracciones de Freshmeat. Otras aplicaciones o sitios que pueden ofrecer más datos son apt, el LSM o rpmgraph.

4.5. Datos de otras fuentes

A diferencia de los proyectos de software "tradicionales", con el software libre en muchos casos se desconocen los recursos humanos. La situación socio-laboral, económica, geográfica y cultural de los integrantes de los proyectos de software puede ser muy dispar y, a buen seguro, tiene repercusión en la forma en la que un proyecto de software evoluciona.

La forma tradicional para la obtención de estos datos ha sido mediante encuestas, que han permitido obtener una imagen de las personas que están detrás del desarrollo de software libre. El problema que éstas conllevan es que las grandes consultas suelen preservar el anonimato de los desarrolladores, por lo que su correlación con las fuentes presentadas con anterioridad se dificulta sobremanera. Para estudios pormenorizados, como sería en el caso de comunidades en torno a proyectos concretos (GNOME, KDE, Linux, Apache...), se podrían conseguir y estudiar datos sobre la situación laboral y personal de los desarrolladores. Esta información podría utilizarse en los microanálisis para ver el grado de participación de profesionales en el desarrollo, cómo se dividen las funciones, etc.

5. Formato intermedio e independiente

En un principio, se intentará que todas las herramientas utilizadas devuelvan los resultados en un formato intermedio e independiente de forma que se pueda aglutinarlos de manera sencilla aún proveniendo de diferentes herramientas. El formato elegido debe ser muy flexible, ya que puede que en un futuro próximo se le añadan más parámetros. A día de hoy, lo mejor sería utilizar un formato XML o SQL, ya que cumplen todos los requisitos comentados con anterioridad y además permiten la compatibilidad hacia atrás.

También hay que tener en cuenta que los conversores de XML (SQL) a cualquier otro tipo de formato que se desee no serán muy difíciles de implementar. La conversión es muy importante, ya que se pretende utilizar en la medida de lo posible herramientas ya existentes en la segunda fase, por lo que una adaptación de los datos obtenidos en la primera fase al formato que estas herramientas requieran sería muy conveniente.

6. Análisis, procesado y visualización de los datos (Segunda fase)

Hemos visto que la primera fase trata la extracción de datos de diferentes fuentes para almacenarlos posteriormente en un formato intermedio que sea independiente de las fuentes y de las herramientas. Esta primera fase, aunque todavía incompleta, se encuentra mucho más madura que la fase que se va a presentar ahora. Parece bastante claro cuáles son las fuentes que se quieren investigar y sólo falta rellenar algunos huecos en las implementaciones para que se dé por acabada.

Una vez que tenemos los datos, se abre ante nosotros un mundo lleno de posibilidades. El volumen de datos del que disponemos y la carencia de análisis hace que se puedan vislumbrar en un futuro próximo gran cantidad de estudios en lo que se refiere al análisis, procesado e interpretación de los resultados.

Existen varias formas de tomar los datos y analizarlos, aunque seguro que en los próximos tiempos se crearán más. La más elemental es proporcionar un simple interfaz web para poder visualizar los datos de manera tabular y/o gráfica. Si los datos están en una base de datos, además se pueden ofrecer correlaciones y datos por proyecto o desarrollador.

El siguiente paso lógico es la utilización de herramientas de análisis estadístico que faciliten el estudio de grandes cantidades de datos, así como su correlación. Existen suites para la realización de estas tareas que pueden hacer todo esto un poco más fácil.

Yendo un poco más allá, cabe la posibilidad de realizar un análisis de clusters, ya sea de paquetes o de desarrolladores. En este sentido se han hecho algunos esfuerzos en los últimos tiempos. Muestra de ello es la aplicación codd-cluster que toma datos de desarrolladores de CODD y crea agrupaciones de proyectos con desarrolladores en común de forma que se pueda llegar a encontrar interdependencias entre diferentes proyectos. Los clusters también pueden ser utilizados para agruparlos según lenguajes de programación, tamaño, etc. Incluso tomando varios parámetros simultáneamente.

7. Conclusiones

Empezamos esta ponencia viendo los problemas que tiene la ingeniería del software tradicional, básicamente por su falta de análisis metódicos y sistemáticos. Hemos visto que el software libre cuenta con cualidades innatas para introducir métodos que permitan conocer más a fondo todos los parámetros que influyen en la generación del mismo. Esta formalización es muy prometedora, ya que el software libre es mucho más difícil de cuantificar que la ingeniería del software tradicional. Por tanto, es probable que el conocimiento del desarrollo de software libre se pueda también dar solución a muchos problemas de la ingeniería del software tradicional. Hemos de recordar en este sentido, que se ha hecho hincapié en que el nuevo enfoque es totalmente complementario al tradicional.

Después de una introducción más bien teórica en la que se ha presentado una declaración de intenciones de la ingeniería del software libre, hemos visto una propuesta de metodología para recabar información de proyectos de software libre. De esta inmensa cantidad de información se pretende extraer experiencias con éxito que puedan ser replicadas en otros desarrollos.

También se ha mostrado que existen una serie de herramientas para la extracción de datos de diversas fuentes, desde el propio código fuente hasta rastreando páginas web de portales de desarrollo de software. Los datos se almacenarán

en un formato intermedio e independiente tanto de la fuente como de las herramientas, para que en una segunda fase, se puedan analizar, correlar, procesar y visualizar de diversas maneras. Esto último abre un campo de consecuencias difíciles de predecir, pero sin lugar a dudas muy ilusionante y prometedor.

La sensación de que el software libre parece que está propiciando uno de esos extraños momentos donde una industria entera cambia de paradigma, se puede también extrapolar a la ingeniería del software libre. Aún nos encontramos en sus comienzos y hace falta un largo camino por andar. Esperamos poder seguir contándolo.

8. Agradecimientos

A esta ponencia le faltaría algo si no expresara mi agradecimiento por las ideas y comentarios con los que me han provisto Jesús M. González de Barahona y Luis Rodero del Grupo de Sistemas y Comunicaciones de la Universidad Rey Juan Carlos y Joaquín Seoane, del Departamento de Ingeniería Telemática de la Universidad Politécnica de Madrid.

9. Bibliografía, referencias y enlaces de interés

1. Frederick P. Brooks Jr. "The Mythical Man-Month", Primera edición año 1975, Addison Wesley Publishing Company

2. Roger Pressman "Ingeniería del Software: Un enfoque práctico", 1997, McGrawHill

3. Adam Smith "Investigación sobre la naturaleza y la causa de la riqueza de las naciones", Primera edición año 1776

4. Jae Yun Moon & Lee Sproull "Essence of Distributed Work: The Case of the Linux Kernel", http://www.firstmonday.dk/issues/issue5_11/moon

5. Sandrep Krishnamurthy "Cave or Community?", http://www.firstmonday.dk/issues/issue7_6/krishnamurthy

6. David Lancashire "Code, Culture and Cash: The Fading Altruism of Open Source Development", http://firstmonday.org/issues/issue6_12/lancashire/

7. Christopher M. Kelty "Free Software/Free Science", http://firstmonday.org/issues/issue6_12/kelty/

8. Jesús M. González Barahona, José M. Ortuño Pérez y otros "Contando Patatas: El tamaño de Debian 2.2", http://congreso.hispalinux.es/congreso2001/actividades/ponencias/gonzalez/html/t1.html

9. David A. Wheeler "Why Open Source Software/Free Software (OSS/FS)? Look at the Numbers!", http://www.dwheeler.com/oss_fs_why.html

10. The FLOSS Survey - Encuesta auspiciada por un proyecto de investigación de la Comisión Europea realizada a más de 2750 desarrolladores (2002), http://floss1.infonomics.nl/stats.php

11. Rishab A. Ghosh, Rüdiger Glott, Bernhard Krieger & Gregorio Robles "Free/Libre Open Source Software: Survey and Study - Final report", http://floss1.infonomics.nl/finalreport/

12. The Widi Survey - Encuesta realizada a más de 5500 desarrolladores (2001), http://widi.berlios.de

13. Boston Consulting Group "Boston Consulting Group/OSDN Hacker Survey" - Encuesta a casi un millar de desarrolladores de Sourceforge (2001), http://www.osdn.com/bcg/

14. Gregorio Robles "Los desarrolladores de Software Libre",

    http://congreso.hispalinux.es/congreso2001/actividades/ponencias/robles/pdf/desarrolladores.pdf

15. Gregorio Robles, Niels Weber & otros "WIDI - Who Is Doing It?",

    http://ig.cs.tu-berlin.de/s2001/ir2/ergebnisse/OSE-study.pdf

16. Curso de doctorado "Programas Libres" - Departamento de Ingeniería Telemática UPM,

    http://curso-sobre.berlios.de

17. Rishab Aiyer Ghosh "Clustering and Dependencies in Free/Open Source Software Development: Methodology and Preliminary Analysis", http://www.idei.asso.fr/Commun/Conferences/Internet/OSS2002/Papiers/Ghosh.PDF

18. Rishab Aiyer Ghosh & Vipul Ved Prakash "The Orbiten Free Software Survey",

    http://www.firstmonday.dk/issues/issue5_7/ghosh

19. Eric S. Raymond "The Cathedral and the Bazaar", http://www.tuxedo.org/~esr/writings/cathedral-bazaar/

20. Nikolai Bezroukov "A Second Look at the Cathedral and the Bazaar",

    http://www.firstmonday.dk/issues/issue4_12/bezroukov/

21. Rishab Aiyer Ghosh "Cooking pot markets: an economic model for the trade in free goods and services on the INternet", http://www.firstmonday.dk/issues/issue3_3/ghosh/

22. Josh Lerner & Jean Tirole "The Simple Economics of Open Source",

    http://www.idei.asso.fr/Commun/Articles/Tirole/simpleeconomics-July-24-2001.pdf

23. Jesús M. González Barahona "Software libre, monopolios y otras yerbas",

    http://sinetgy.org/~jgb/articulos/soft-libre-monopolios/

24. Proyecto Gestión Libre de Hispalinux http://gestion-libre.hispalinux.es

25. CODD - Aplicación rastreadora de autores y dependencias de código e interfaces en código fuente, http://codd.berlios.de

26. CODDWeb - Interfaz web para acceder a los resultados de CODD, http://floss1.infonomics.nl/coddweb/

27. SLOCcount - Herramienta para contar líneas de código fuente en diferentes lenguajes y cálculo de costes y tiempos de desarrollo, http://www.dwheeler.com/sloccount/

28. GNOME-stats - Estadísticas (estáticas por ahora) del CVS del proyecto GNOME, http://gnome-stats.berlios.de

29. KDE-stats - Estadísticas (estáticas por ahora) del CVS del proyecto KDE, http://kde-stats.berlios.de

30. sloc2html - Aplicación que presenta los resultados de SLOCcount a través de un interfaz web agradable, http://halfdans.net/index.py?p=sloc2html

31. Estadísticas gráficas de GNOME con SLOCCount y sloc2html, http://gnome-stats.berlios.de/gnome-sloc.html

32. Free Software Foundation Europe - Distribución geográfica de desarrolladores de software libre, http://www.fsfeurope.org/coposys/index.en.html

33. Alison Luo "TPM (Trinity Participation Metric) for Open Source Developers",

    http://www.cse.ucsc.edu/~alison/projects/cmpe276/index.html

34. Linux Study - Questionnaire on Linux kernel developers (141 desarrolladores),

    http://www.psychologie.uni-kiel.de/linux-study/

35. Proyecto Debian - Mapa con la distribución geográfica de los desarrolladores Debian,

    http://www.debian.org/devel/developers.loc

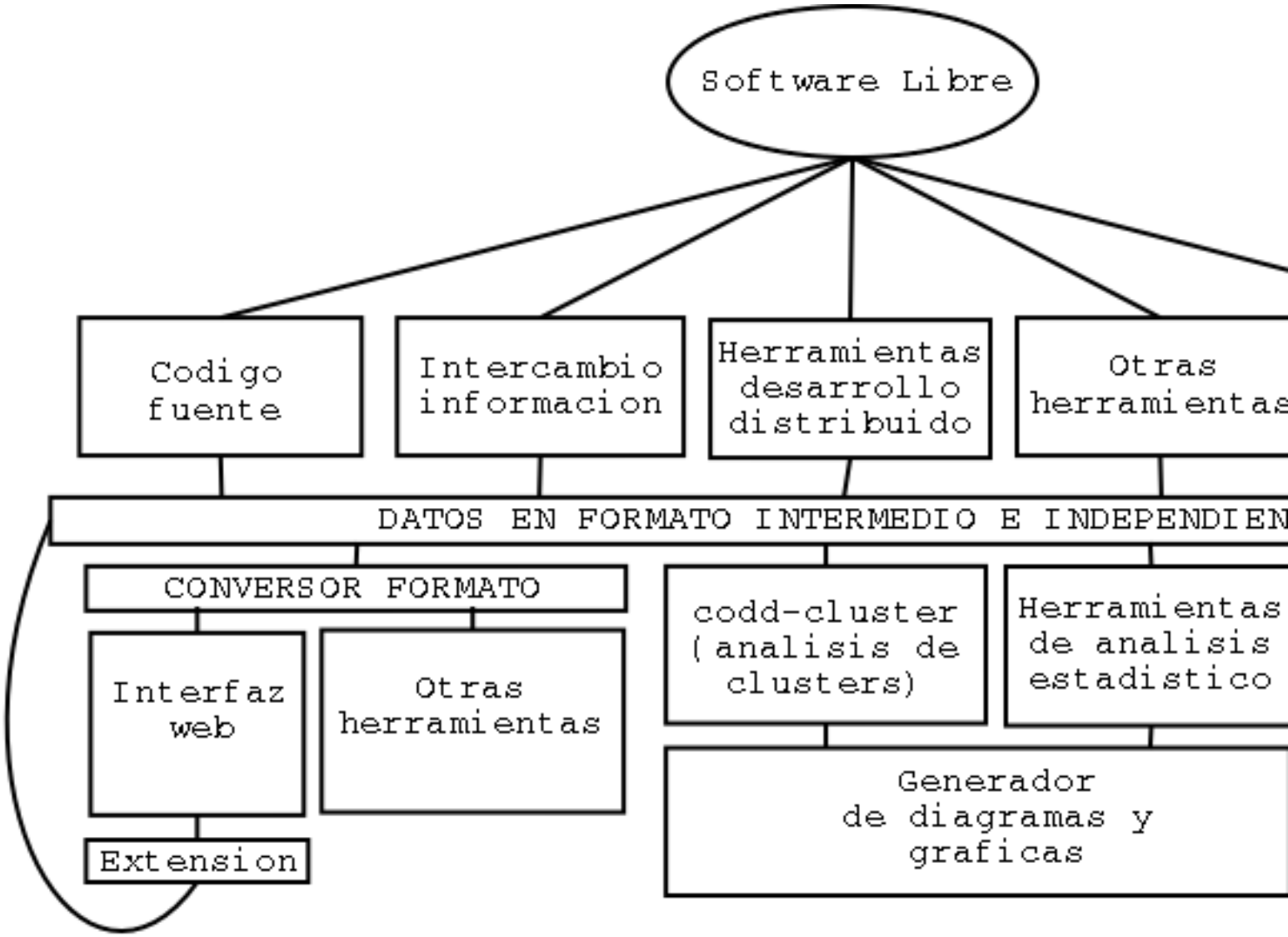
36. Edward Betts "Debian Developer Centre of Mass", http://people.debian.org/~edward/average/

37. rpmgraph - una herramienta que hace grafos de dependencias a partir de paquetes RPM,

    http://www.freshmeat.net/rpmgraph

38. Graphical Display of Package Dependencies (algo parecido a rpmgraph para Debian), http://www.debianplanet.org/node.php?id=695

10. Sistema de medición y análisis de software libre


Software Libre
Codigo fuente
Intercambio informacion
Herramientas desarrollo distribuido
Otras herramientas
DATOS EN FORMATO INTERMEDIO E INDEPENDIEN
CONVERSOR FORMATO
Interfaz web
Otras herramientas
Extension
codd-cluster (analisis de clusters)
Herramientas de analisis estadistico
Generador de diagramas y graficas