Software libre

Jesús M. González Barahona
Joaquín Seoane Pascual
Gregorio Robles

P07/M2101/02709

UOC
www.uoc.edu

Índice

1. Introducción

> "If you have an apple and I have an apple and we exchange apples, then you and I will still each have one apple. But if you have an idea and I have an idea and we exchange these ideas, then each of us will have two ideas."
>
> "Si tú tienes una manzana y yo tengo una manzana y las intercambiamos, seguiremos teniendo una manzana cada uno. Pero si tú tienes una idea y yo tengo una idea y las intercambiamos, cada uno de nosotros tendrá dos ideas."
>
> Atribuido a Bernard Shaw

¿Qué es el software libre? ¿Qué es y qué implicaciones tiene la licencia de un programa libre? ¿Cómo se está desarrollando el software libre? ¿Cómo se financian los proyectos de software libre y qué modelos de negocio relacionados con ellos se están experimentando? ¿Qué motiva a los desarrolladores, especialmente a los que son voluntarios, a involucrarse en proyectos de software libre? ¿Cómo son estos desarrolladores? ¿Cómo se coordinan en sus proyectos y cómo es el software que producen? En resumen, ¿cuál es el panorama general del software libre? Éste es el tipo de preguntas que trataremos de responder en este texto. Porque aunque el software libre está cada vez más presente en los medios de comunicación y en las conversaciones de los profesionales de la informática, y aunque incluso empieza a estar en boca de los ciudadanos en general, aún es un desconocido para muchos. Y los que lo conocen muchas veces no saben más que de algunos de sus aspectos, y desconocen completamente otros.

Para empezar, en este capítulo vamos a presentar los aspectos específicos del software libre, centrándonos fundamentalmente en explicar sus bases para los que se aproximen al tema por primera vez y en motivar su importancia. Entre estas bases nos detendremos en la definición del término (para saber de qué vamos a hablar) y en las consecuencias principales del uso (y de la mera existencia) del software libre.

1.1. El concepto de *libertad* en el software

Desde principios de los años setenta nos hemos acostumbrado a que quien comercializa un programa pueda imponer (e imponga) las condiciones bajo las que puede usarse. Puede, por ejemplo, prohibir que sea prestado a un tercero. A pesar de que el software es el elemento tecnológico más flexible y adaptable que tenemos, puede imponerse (y es común imponer) la imposibilidad de adaptarlo a unas necesidades concretas, o de corregir sus errores, sin el permiso explícito del productor, que normalmente se reserva en exclusiva estas posibilidades. Pero ésta es sólo una de las posibilidades que ofrece la legislación actual: el *software libre*, por el contrario, otorga las libertades que el *software privativo* niega.

Software privativo

En este texto utilizaremos el término *software privativo* para referirnos a cualquier programa que no pueda considerarse software libre, de acuerdo con la definición que se ofrece más adelante.

1.1.1. Definición

Así pues, el término *software libre* (o *programas libres*), tal como fue concebido por Richard Stallman en su definición (Free Software Foundation, "Free software definition" –http://www.gnu.org/philosophy/free-sw.html– [120]), hace referencia a las libertades que puede ejercer quien lo recibe, concretamente cuatro:

1) Libertad para ejecutar el programa en cualquier sitio, con cualquier propósito y para siempre.

2) Libertad para estudiarlo y adaptarlo a nuestras necesidades. Esto exige el acceso al código fuente.

3) Libertad de redistribución, de modo que se nos permita colaborar con vecinos y amigos.

4) Libertad para mejorar el programa y publicar sus mejoras. Esto también exige el código fuente.

El mecanismo que se utiliza para garantizar estas libertades, de acuerdo con la legalidad vigente, es la distribución mediante una licencia determinada, como veremos más adelante (capítulo 3). En ella el autor plasma su permiso para que el receptor del programa pueda ejercer esas libertades, y también las restricciones que pueda querer aplicar (como dar crédito a los autores originales en caso de redistribución). Para que la licencia sea considerada libre, estas restricciones no pueden ir en contra de las libertades mencionadas.

La ambigüedad de *free*

El término original en inglés para *programas libres* es *free software*. Sin embargo, *free*, además de 'libre', significa 'gratis', lo que genera gran confusión. Por ello a menudo en inglés se toman prestadas palabras españolas y se habla de *libre software*, en contraposición a *gratis software*, al igual que nosotros tomamos prestada la palabra *software*.

Así pues, las definiciones de software libre no hacen ninguna referencia a que pueda conseguirse gratuitamente: el software libre y el software gratuito son cosas bien distintas. Sin embargo, dicho esto, hay que explicar también que debido a la tercera libertad, cualquiera puede redistribuir un programa sin pedir contraprestación económica ni permiso, lo que hace prácticamente imposible obtener grandes ganancias simplemente por la distribución de software libre: cualquiera que lo haya obtenido puede a su vez redistribuirlo a precio más bajo, o incluso gratis.

Nota

A pesar de que cualquiera puede comercializar un programa dado a cualquier precio, y eso hace que teóricamente el precio de redistribución tienda hacia el coste marginal de copia, existen modelos de negocio basados precisamente en vender software, porque hay muchas circunstancias en las que el consumidor está dispuesto a pagar si recibe ciertas contraprestaciones, como por ejemplo una cierta garantía, aunque sea subjetiva, sobre el software que adquiere o un valor añadido en forma de selección, actualización y organización de un conjunto de programas.

Desde un punto de vista práctico, hay varios textos que definen más precisamente qué condiciones tiene que cumplir una licencia para ser considerada como de software libre. Entre ellos, destacan por su importancia histórica la definición de software libre de la Free Software Foundation (http://www.gnu.org/philosophy/free-sw.html) [120], las directrices de Debian para decidir si un programa es libre (http://www.debian.org/social_contract.html#guidelines) [104] y la definición del término *open source* por la Open Source Initiative (http://www.opensource.org/docs/definition_plain.html) [215], muy similar a las anteriores.

Nota

Por ejemplo, las directrices de Debian entran en el detalle de permitir que el autor exija que los códigos fuente distribuidos no sean modificados directamente, sino que los originales se acompañen de parches separados, y que los programas binarios se generen con nombres distintos del original. Además exigen que las licencias no contaminen otros programas distribuidos en el mismo medio.

1.1.2. Términos relacionados

Equivalente de *software libre* es el término *open source software* ('programas de fuente abierta'), promovido por Eric Raymond y la Open Source Initiative. Filosóficamente, el término es muy distinto, ya que hace énfasis en la disponibilidad de código fuente, no en la libertad, pero su definición es prácticamente la misma que la de Debian ("The open source definition", 1998 –http://www.opensource.org/docs/definition_plain.html–) [183]. Este nombre es más políticamente aséptico y recalca un aspecto técnico que puede dar lugar a ventajas técnicas, como mejores modelos de desarrollo y negocio, mayor seguridad, etc. Fuertemente criticado por Richard Stallman ("Why *free software* is better than *open source*") [204] y la Free Software Foundation (http://www.fsf.org) [27], ha encontrado mucho más eco en la literatura comercial y en las estrategias de las empresas que de una manera u otra apoyan el modelo.

Otros términos relacionados de algún modo con el software libre son los siguientes:

Freeware	Son programas gratuitos. Normalmente se distribuyen sólo en binario, y se pueden obtener sin coste. A veces se consigue también permiso de redistribución, pero otras no, de manera que entonces sólo se pueden obtener del sitio "oficial" mantenido a ese efecto. Es habitual que se usen para promocionar otros programas (típicamente con funcionalidad más completa) o servicios. Ejemplos de este tipo de programas son Skype, Google Earth o Microsoft Messenger.
Shareware	No es siquiera software gratis, sino un método de distribución, ya que los programas, generalmente sin códigos fuente, se pueden copiar libremente, pero no usar continuadamente sin pagarlos. La exigencia de pago puede estar incentivada por funcionalidad limitada, mensajes molestos o una simple apelación a la moral del usuario. Además, las estipulaciones legales de la licencia podrían utilizarse en contra del infractor.
Charityware, careware	Se trata generalmente de *shareware* cuyo pago se pide para una organización *caritativa* patrocinada. En muchos casos, el pago no se exige, pero se solicita una contribución voluntaria. Algún software libre, como **Vim**, solicita contribuciones voluntarias de este tipo (Brian Molenaar, "What is the context of charityware?") [173].
Dominio público	El autor renuncia absolutamente a todos sus derechos en favor del común, lo cual tiene que quedar explícitamente declarado en el programa, ya que si no se dice nada, el programa es propietario y no se puede hacer nada con él. En este caso, y si además se proporcionan los códigos fuente, el programa es libre.
Copyleft	Se trata de un caso particular de software libre cuya licencia obliga a que las modificaciones que se distribuyan sean también libres.
Propietario, cerrado, no libre	Se trata de términos usados para denominar al software que no es libre ni de fuente abierta.

1.2. Motivaciones

Como hemos visto, hay dos grandes familias de motivaciones para el desarrollo de software libre, que dan lugar asimismo a los dos nombres con que se lo conoce:

- La motivación ética, abanderada por la Free Software Foundation (http://www.fsf.org) [27], heredera de la cultura *hacker* y partidaria del apelativo *libre*, que argumenta que el software es conocimiento que debe poder difundirse sin trabas y cuya ocultación es una actitud antisocial, y que afirma además que la posibilidad de modificar programas es una forma de libertad de expresión. Puede profundizarse en este aspecto en los ensayos de Stallman (*Free software, free society. Selected essays of Richard M. Stallman*) [211] o en el análisis de Pekka Himanen (*The hacker ethic and the spirit of the information age*. Random House, 2001) [144].

- La motivación pragmática, abanderada por la Open Source Initiative (http://www.opensource.org) [54] y partidaria del apelativo *fuente abier-*

ta, que argumenta ventajas técnicas y económicas que repasaremos en el apartado siguiente.

Aparte de estas dos grandes motivaciones, la gente que trabaja en software libre puede hacerlo por muchas otras razones, que van desde la diversión (Linus Torvalds y David Diamond, Texere, 2001) [217] hasta la mera retribución económica, posiblemente debida a *modelos de negocio* sustentables. En el capítulo 4 se profundiza en estas motivaciones a partir de análisis objetivos.

1.3. Consecuencias de la libertad del software

El software libre trae consigo numerosas ventajas y pocas desventajas, muchas de las cuales han sido exageradas (o falseadas) por la competencia propietaria. De ellas, la que más fundamento tiene es la económica, ya que como hemos visto, no es posible obtener mucho dinero de la distribución y ésta la puede y la suele hacer alguien distinto del autor. Por ello se necesitan modelos de negocio y otros mecanismos de financiación, que se desarrollan en el capítulo 5. Otras desventajas, como la falta de soporte o la calidad escasa, están relacionadas con la financiación, pero además en muchos casos son falsas, ya que incluso software sin ningún tipo de financiación suele ofrecer muy buen soporte gracias a foros de usuarios y desarrolladores, y muchas veces tiene gran calidad.

Teniendo presentes las consideraciones económicas, hemos de observar que el modelo de costes del software libre es muy distinto del modelo de costes del software propietario, ya que gran parte de él se ha desarrollado fuera de la economía formal monetaria, muchas veces con mecanismos de trueque: "yo te doy un programa que te interesa y tú lo adaptas a tu arquitectura y le haces mejoras que a ti te interesan." En el capítulo 7 se explican mecanismos de ingeniería de software apropiados para aprovechar estos recursos humanos no pagados y con características propias, mientras que en el capítulo 8 se estudian las herramientas usadas para hacer efectiva esta colaboración. Pero además, gran parte de los costes disminuyen por el hecho de que es libre, ya que los programas nuevos no tienen por qué empezar desde cero, sino que pueden reutilizar software ya hecho. La distribución tiene también un coste mucho menor, ya que se hace por Internet y con propaganda gratuita en foros públicos destinados a ello.

Otra consecuencia de las libertades es la calidad derivada de la colaboración voluntaria de gente que contribuye o que descubre y notifica errores en entornos y situaciones inimaginables por el desarrollador original. Además, si un programa no ofrece la calidad suficiente, la competencia puede tomarlo y mejorarlo partiendo de lo que hay. Así la *colaboración* y la *competencia*, dos poderosos mecanismos, se combinan para conseguir una mejor calidad.

Examinemos ahora las consecuencias beneficiosas según el destinatario.

1.3.1. Para el usuario final

El usuario final, ya sea individual o empresa, puede encontrar verdadera competencia en un mercado con tendencia al monopolio. En particular, no depende necesariamente del soporte del fabricante del software, ya que puede haber múltiples empresas, quizá pequeñas, que dispongan del código fuente y de conocimientos y que puedan hacer negocio manteniendo determinados programas libres.

Ya no se depende tanto de la *fiabilidad* del fabricante para intentar deducir la calidad de un producto, sino que la guía nos la dará la aceptación de la comunidad y la disponibilidad de los códigos fuente. Además, nos podemos olvidar de *cajas negras*, en las que hay que confiar "porque sí", y de las estrategias de los fabricantes, que pueden decidir unilateralmente dejar de mantener un producto.

La evaluación de productos antes de su adopción es ahora mucho más sencilla, ya que basta con instalar los productos alternativos en nuestro entorno real y probar, mientras que para software propietario hay que fiarse de informes externos o negociar pruebas con los proveedores, lo cual no siempre es posible.

Dada la libertad de modificar el programa para uso propio, el usuario puede personalizarlo o adaptarlo a sus necesidades, corrigiendo errores si los tuviera. El proceso de corrección de errores descubiertos por los usuarios en software propietario suele ser extremadamente penoso, si no imposible, ya que si conseguimos que se reparen, a menudo ello se hará en la versión siguiente, que puede tardar años en salir y que a veces, además, habrá que comprar de nuevo. En software libre, sin embargo, podemos hacer nosotros dicha reparación, si estamos cualificados, o contratar el servicio fuera. También podemos, directamente o contratando servicios, integrar el programa con otro o auditar su calidad (por ejemplo la seguridad). El control pasa, en gran medida, del proveedor al usuario.

1.3.2. Para la Administración pública

La Administración pública es un gran usuario de características especiales, ya que tiene obligaciones especiales con el ciudadano, sea proporcionándole servicios accesibles, neutrales respecto a los fabricantes, o garantizando la integridad, la utilidad, la privacidad y la seguridad de sus datos a largo plazo. Todo ello la obliga a ser más respetuosa con los estándares que las empresas privadas y a mantener los datos en formatos abiertos y manipulados con software que no dependa de estrategia de empresas, generalmente extranjeras, certificado como seguro por auditoría interna. La adecuación a estándares es una característica notable del software libre no tan respetada por el software propietario, generalmente ávido de crear mercados cautivos.

Asimismo, la Administración tiene una cierta función de escaparate y guía de la industria que le hace tener un gran impacto que debería dirigirse a la creación de un tejido tecnológico generador de riqueza nacional. Dicha riqueza puede crearse mediante el fomento de empresas cuyo negocio sea, en parte, el desarrollo de nuevo software libre para la Administración o el mantenimiento, la adaptación o la auditoría del software existente. En el capítulo 6 nos extenderemos más en esta cuestión.

1.3.3. Para el desarrollador

Para el desarrollador y productor de software, la libertad cambia mucho las reglas del juego. Con ella le es más fácil competir siendo pequeño y adquirir tecnología punta. Puede aprovecharse del trabajo de los demás, compitiendo incluso con otro producto mediante la modificación de su propio código, si bien también el competidor copiado se aprovechará de nuestro código (si es *copyleft*). Si el proyecto se lleva bien, puede conseguirse la colaboración gratuita de mucha gente, y además, se tiene acceso a un sistema de distribución prácticamente gratuito y global. No obstante, queda pendiente el problema de cómo obtener recursos económicos si el software realizado no es fruto de un encargo pagado. En el capítulo 5 se tratará en detalle este tema.

1.3.4. Para el integrador

Para el integrador, el software libre es el paraíso. Significa que ya no hay más cajas negras que intentar encajar, a menudo con ingeniería inversa. Puede limar asperezas e integrar trozos de programas para conseguir el producto integrado necesario, al disponer de un acervo ingente de software libre de donde extraer las piezas.

1.3.5. Para el que proporciona mantenimiento y servicios

Disponer del código fuente lo cambia todo y nos sitúa casi en las mismas condiciones que el productor. Si no son las mismas es porque hace falta un conocimiento profundo del programa que sólo el desarrollador posee, por lo que es conveniente que el mantenedor participe en los proyectos que se dedica a mantener. El valor añadido de los servicios es mucho más apreciado, ya que el coste del programa es bajo. Éste es actualmente el negocio más claro con software libre y con el que es posible un mayor grado de competencia.

1.4. Resumen

Este primer capítulo ha servido como toma de contacto con el mundo del software libre. El concepto, definido por Richard Stallman, se basa en cuatro libertades (libertad de ejecución, libertad de estudio, libertad de redistribución y libertad de mejora), dos de las cuales suponen el acceso al código fuente. Esta accesibilidad y sus ventajas motivan otro punto de vista menos ético y más pragmático, defendido por la Open Source Initiative, que ha dado lugar

a otro término: *software de fuente abierta*. Se han comentado también otros términos relacionados por similitud o contraposición, y que permiten aclarar los conceptos. Finalmente, se ha hablado de las consecuencias de la libertad del software para los principales actores implicados.

2. Un poco de historia

> "When I started working at the MIT Artificial Intelligence Lab in 1971, I became part of a software-sharing community that had existed for many years. Sharing of software was not limited to our particular community; it is as old as computers, just as sharing of recipes is as old as cooking. But we did it more than most. [...] We did not call our software *free software*, because that term did not yet exist; but that is what it was. Whenever people from another university or a company wanted to port and use a program, we gladly let them. If you saw someone using an unfamiliar and interesting program, you could always ask to see the source code, so that you could read it, change it, or cannibalize parts of it to make a new program."
>
> "Cuando comencé a trabajar en el Laboratorio de Inteligencia Artificial del MIT en 1971, pasé a formar parte de una comunidad de software compartido que existía desde hacía muchos años. Compartir código no era algo específico de nuestra comunidad: es algo tan antiguo como los ordenadores, como compartir recetas es tan viejo como cocinar. Pero nosotros lo hacíamos más que la mayoría. [...] No llamábamos a nuestro software *software libre* porque ese término aún no existía, pero eso es lo que era. Cuando alguien de otra universidad o de una empresa quería portar y usar un programa, nosotros le dejábamos hacerlo con gusto. Si veías a alguien utilizando un programa raro e interesante, siempre podías pedirle ver el código fuente, para poder leerlo, cambiarlo o canibalizar partes para hacer un programa nuevo."
>
> Richard Stallman, "The GNU Project" (publicado originalmente en el libro *Open sources*) [208]

Aunque todas las historias relacionadas con la informática son forzosamente breves, la del software libre es una de las más largas entre ellas. De hecho, podría decirse que en sus comienzos, prácticamente todo el software desarrollado cumplía con las definiciones de *software libre*, aunque el concepto ni siquiera existía aún. Más tarde la situación cambió por completo, y el software privativo dominó la escena, prácticamente en exclusiva, durante bastante tiempo. Fue durante esta época cuando se sentaron las bases del software libre como lo entendemos hoy en día, y cuando, poco a poco, empezaron a aparecer programas libres. Con el tiempo, estos comienzos se han convertido en una tendencia que ha ido creciendo y madurando hasta llegar a la situación actual, en la que el software libre es una posibilidad que hay que considerar en casi todos los ámbitos.

Esta historia es bastante desconocida, hasta tal punto que para gran parte de los profesionales informáticos el software privativo es el software "en su estado natural". Sin embargo, la situación es más bien la contraria, y las semillas del cambio que se empezó a entrever en la primera década de 2000 habían sido plantadas ya a principios de los años ochenta.

Bibliografía

No hay muchas historias exhaustivas sobre el software libre, y las que hay son artículos normalmente limitados al objeto de su estudio. En cualquier caso, el lector interesado puede completar lo expuesto en este capítulo consultando "Open Source Initiative. History of the OSI" [146] (http://www.opensource.org/docs/history.php), que pone el énfasis en el impacto del software libre en el mundo empresarial entre los años 1998 y 1999; "A brief history of free/open source software movement" [190], de Chris Rasch, que cubre la historia del software libre hasta el año 2000, o "The origins and future of open source software" (1999) [177], de Nathan Newman, que se centra en gran medida en la promo-

ción indirecta que el Gobierno de EE.UU. ha hecho del software libre o sistemas similares durante las décadas de 1970 y 1980.

2.1. El software libre antes del software libre

El software libre como concepto no apareció hasta principios de la década de 1980. Sin embargo, su historia puede trazarse desde bastantes años antes.

2.1.1. Y en el principio fue libre

Durante los años sesenta el panorama de la informática estaba dominado por los grandes ordenadores, instalados fundamentalmente en empresas y centros gubernamentales. IBM era el principal fabricante, con gran diferencia sobre sus competidores. En esta época, cuando se adquiría un ordenador (el hardware), el software venía como un acompañante. Mientras se pagase el contrato de mantenimiento, se tenía acceso al catálogo de software que ofrecía el fabricante. Además, no era común la idea de que los programas fueran algo "separado" desde un punto de vista comercial.

En esta época el software se distribuía habitualmente junto con su código fuente (en muchos casos sólo como código fuente), y en general, sin restricciones prácticas. Los grupos de usuarios como SHARE (usuarios de sistemas IBM) o DECUS (usuarios de DEC) participaban de estos intercambios, y hasta cierto punto, los organizaban. La apartado "Algorithms" de la revista *Communications of the ACM* era otro buen ejemplo de foro de intercambio. Podría decirse que durante estos primeros años de la informática el software era libre, al menos en el sentido de que los que tenían acceso a él podían disponer habitualmente del código fuente, estaban acostumbrados a compartirlo, a modificarlo y a compartir también estas modificaciones.

El 30 de junio de 1969 IBM anunció que, a partir de 1970, iba a vender parte de su software por separado (Burton Grad, 2002) [131]. Ello supuso que sus clientes ya no podían obtener, incluido en el precio del hardware, los programas que necesitaban. El software se comenzó a percibir como algo con valor intrínseco y, como consecuencia, se hizo cada vez más habitual restringir escrupulosamente el acceso a los programas y se limitaron, en la medida de lo posible (tanto técnica como legalmente), las posibilidades de los usuarios para compartir, modificar o estudiar el software. Dicho de otro modo, se pasó a la situación que sigue siendo habitual en el mundo del software a principios del siglo XXI.

Bibliografía

El lector interesado en esta época de transición puede leer, por ejemplo, "How the ICP Directory began" [226] (1998), donde Larry Welke comenta cómo nació uno de los primeros catálogos de software no ligados a un fabricante, y cómo en este proceso descubrió que las empresas estaban dispuestas a pagar por programas no hechos por el fabricante de sus ordenadores.

A mediados de la década de 1970 era ya absolutamente habitual, en cualquier ámbito informático, encontrarse con software privativo. Ello supuso un gran cambio cultural entre los profesionales que trabajaban con software y fue el origen del florecimiento de un gran número de empresas dedicadas al nuevo negocio. Faltaba aún casi una década para que empezase a aparecer, de forma organizada y como reacción a esta situación, lo que hoy conocemos como *software libre*.

2.1.2. Década de los setenta y principios de la década de los ochenta

Incluso cuando la tendencia abrumadoramente mayoritaria era explorar el modelo de software privativo, había iniciativas que mostraban algunas características de lo que luego se consideraría software libre. De hecho, alguna de ellas llegó a producir software libre tal como lo definimos hoy en día. Entre ellas cabe destacar SPICE, TeX y Unix, cuyo caso es mucho más complejo.

SPICE (Simulation Program with Integrated Circuit Emphasis) es un programa desarrollado en la Universidad de California, en Berkeley, para simular las características eléctricas de un circuito integrado. Fue desarrollado y puesto en el dominio público por su autor, Donald O. Pederson, en 1973. SPICE fue originalmente una herramienta docente, y como tal se extendió rápidamente a muchas universidades de todo el mundo. En ellas fue usado por muchos estudiantes de la por aquel entonces incipiente disciplina del diseño de circuitos integrados. Puesto que estaba en el dominio público, SPICE podía redistribuirse, modificarse, estudiarse... Incluso se podía adaptar a unas necesidades concretas y vender esa versión como producto privativo (lo que han hecho durante su historia docenas de veces una gran cantidad de empresas). Con estas características, SPICE tenía todas las papeletas para convertirse en el estándar de la industria, con sus diferentes versiones. Y efectivamente, eso fue lo que ocurrió. Probablemente éste fue el primer programa con características de software libre que durante un tiempo copó un mercado, el de los simuladores de circuitos integrados, y sin duda pudo hacerlo precisamente por tener estas características (además de sus innegables cualidades técnicas).

Bibliografía

Se puede consultar más información sobre la historia de SPICE en "The life of SPICE", trabajo presentado en el 1996 Bipolar Circuits and Technology Meeting, Minneapolis, MN, EE.UU., en septiembre de 1996 [175].

La página web de SPICE puede encontrarse en http://bwrc.eecs.berkeley.edu/Classes/IcBook/SPICE/.

Donald Knuth comenzó a desarrollar TeX durante un año sabático, en 1978. TeX es un sistema de tipografía electrónica muy utilizado para la producción de documentos de calidad. Desde el principio, Knuth utilizó una licencia que hoy sería considerada como de software libre. Cuando el sistema se consideró

razonablemente estable, en 1985, mantuvo esa licencia. En esa época, TeX era uno de los sistemas más grandes y más conocidos que se podía considerar software libre.

Bibliografía

Algunos hitos de la historia de TeX pueden consultarse en línea en http://www.math.utah.edu/software/plot79/tex/history.html [39]. Para saber más detalles, también es útil el artículo correspondiente de Wikipedia, http://www.wikipedia.org/wiki/TeX [233].

2.1.3. Desarrollo temprano de Unix

Unix, uno de los primeros sistemas operativos portables, fue creado originalmente por Thompson y Ritchie (entre otros) en los Bell Labs de AT&T. Su desarrollo ha continuado desde su nacimiento, hacia 1972, dando lugar a innumerables variantes comercializadas por (literalmente) decenas de empresas.

Durante los años 1973 y 1974, Unix llegó a muchas universidades y centros de investigación de todo el mundo, con una licencia que permitía su uso para fines académicos. Aunque había ciertas restricciones que impedían su libre distribución, entre las organizaciones que disponían de la licencia el funcionamiento fue muy similar al que se vio más tarde en muchas comunidades de software libre. Los que tenían acceso al código fuente de Unix tuvieron un sistema que podían estudiar, mejorar y ampliar. Alrededor de él apareció una comunidad de desarrolladores que pronto empezó a girar en torno al CSRG de la Universidad de California, en Berkeley. Esta comunidad desarrolló su propia cultura, que como veremos más adelante, fue muy importante en la historia del software libre. Unix fue, hasta cierto punto, un ensayo temprano de lo que se vio con GNU y Linux varios años más tarde. Estaba confinado a una comunidad mucho más pequeña y era necesaria la licencia de AT&T, pero en otros aspectos su desarrollo fue similar (en un mundo mucho menos comunicado).

Modos de desarrollo propios del software libre

En *Netizens. On the history and impact of Usenet and the Internet* (IEEE Computer Society Press, 1997 [139], página 139) pueden leerse unas líneas que se podrían referir a muchos proyectos libres: "Las contribuciones al valor de Unix durante su desarrollo temprano fueron muchas, gracias al hecho de que el código fuente estaba disponible. Podía ser examinado, mejorado y personalizado".

En la página 142 de la misma obra se dice lo siguiente: "Los pioneros como Henry Spencer están de acuerdo en lo importante que fue para los que pertenecían a la comunidad Unix tener el código fuente. él resalta cómo la disposición de los códigos fuente hacía posible la identificación y la reparación de los errores que descubrían. [...] Incluso en los últimos 1970 y primeros 1980, prácticamente cada sitio Unix tenía fuentes completas".

Aún más explícito es el texto de Marc Rochkind "Interview with dick haight" (*Unix Review*, mayo 1986) [198]: "ésta era una de las grandes cosas sobre Unix en los primeros días: la gente realmente compartía con los demás. [...] No sólo aprendíamos mucho aquellos días del material compartido, sino que nunca teníamos que preocuparnos sobre cómo funcionaban realmente las cosas porque siempre podíamos ir a leer el código fuente".

Con el tiempo, Unix fue también un ejemplo temprano de los problemas que podían presentar los sistemas privativos que a primera vista "tenían alguna característica del software libre". A finales de la década de 1970, y sobre todo durante la de 1980, AT&T cambió su política, y el acceso a nuevas versiones de Unix se convirtió en algo difícil y caro. La filosofía de los primeros años, que había hecho tan popular a Unix entre los desarrolladores, cambió radicalmente hasta tal punto que en 1991 AT&T puso una demanda a la Universidad de Berkeley por publicar el código de Unix BSD que el CSRG de Berkeley había creado. Pero ésa es otra historia, que retomaremos más adelante.

2.2. El comienzo: BSD, GNU

Todos los casos descritos en el apartado anterior fueron iniciativas individuales o no cumplían estrictamente los requisitos del software libre. Hasta principios de la década de 1980 no aparecieron, de forma organizada y consciente, los primeros proyectos para la creación de sistemas compuestos de software libre. En esa época se empezaron también a fijar (lo que probablemente es más importante) los fundamentos éticos, legales e incluso económicos, que se han ido desarrollando y completando hasta hoy en día. Y como el nuevo fenómeno necesitaba un nombre, durante esos años se acuñó también el propio término *software libre*.

2.2.1. Richard Stallman, GNU, FSF: nace el movimiento del software libre

A principios de 1984, Richard Stallman, en aquella época empleado en el AI Lab del MIT, abandonó su trabajo para comenzar el proyecto GNU. Stallman se consideraba un *hacker* de los que gozaban compartiendo sus inquietudes tecnológicas y su código. Veía con desagrado cómo su negativa a firmar acuerdos de exclusividad y de no compartición le estaban convirtiendo en un extraño en su propio mundo, y cómo el uso de software privativo en su entorno le dejaba impotente ante situaciones que antes podía solventar fácilmente.

Su idea al abandonar el MIT era construir un sistema de software completo, de propósito general, pero totalmente libre ("The GNU Project", DiBona *et al.*) [208]. El sistema (y el proyecto que se encargaría de hacerlo realidad) se llamó GNU (acrónimo recursivo, "GNU's not Unix"). Aunque desde el principio el proyecto GNU incluyó en su sistema software ya disponible (como TeX o, más adelante, el sistema X Window), había mucho que construir. Richard Stallman comenzó por escribir un compilador de C (GCC) y un editor (Emacs), ambos aún en uso (y muy populares).

Desde el principio del proyecto GNU, Richard Stallman se preocupó por las libertades que tendrían los usuarios de su software. Estaba interesado en que no sólo los que recibieran los programas directamente del proyecto GNU, sino cualquiera que lo recibiera después de cualquier número de redistribuciones y (quizás) modificaciones, siguiera pudiendo disfrutar de los mismos de-

rechos (modificación, redistribución, etc.). Para ello escribió la licencia GPL, probablemente la primera licencia de software diseñada específicamente para garantizar que un programa fuera libre en este sentido. Al mecanismo genérico que utilizan las licencias de tipo GPL para conseguir estas garantías, Richard Stallman lo llamó *copyleft*, que sigue siendo el nombre de una gran familia de licencias de software libre (Free Software Foundation, GNU General Public License, versión 2, junio 1991) [118].

Richard Stallman también fundó la Free Software Foundation (FSF) para conseguir fondos, que dedica al desarrollo y la protección del software libre, y sentó sus fundamentos éticos con "The GNU Manifesto" (Free Software Foundation, 1985) [117] y "Why software should not have owners" (Richard Stallman, 1998) [207].

Desde el punto de vista técnico, el proyecto GNU fue concebido como un trabajo muy estructurado y con metas muy claras. El método habitual estaba basado en grupos relativamente pequeños de personas (normalmente voluntarios) que desarrollaban alguna de las herramientas que luego encajarían perfectamente en el rompecabezas completo (el sistema GNU). La modularidad de Unix, en la que se inspiraba el desarrollo de este proyecto, coincidía totalmente con esta idea. El método de trabajo generalmente implicaba el uso de Internet, pero ante su escasa implantación en aquellos días, la Free Software Foundation también vendía cintas en las que grababa las aplicaciones, de manera que fue probablemente una de las primeras organizaciones que se benefició económicamente (aunque de manera bastante limitada) de la creación de software libre.

A principios de la década de 1990, unos seis años después de su nacimiento, el proyecto GNU estaba muy cerca de tener un sistema completo similar a Unix. Aun así, hasta ese momento aún no había producido una de las piezas fundamentales: el núcleo del sistema (también llamado *kernel*, la parte del sistema operativo que se relaciona con el hardware, lo abstrae, y permite que las aplicaciones compartan recursos y, en el fondo, funcionen). Sin embargo, el software de GNU era muy popular entre los usuarios de las distintas variantes de Unix, por aquel entonces el sistema operativo más usado en las empresas. Además, el proyecto GNU había conseguido ser relativamente conocido entre los profesionales informáticos, y muy especialmente entre los que trabajaban en universidades. En esa época, sus productos ya gozaban de una merecida reputación de estabilidad y calidad.

2.2.2. El CSRG de Berkeley

El CSRG (Computer Science Research Group) de la Universidad de California, en Berkeley, fue desde 1973 uno de los centros donde más desarrollo relacionado con Unix se hizo durante los años 1979 y 1980. No sólo se portaron aplicaciones y se construyeron otras nuevas para su funcionamiento sobre Unix, sino que se llevaron a cabo importantes mejoras en el núcleo y se le añadió

mucha funcionalidad. Por ejemplo, durante la década de los ochenta, varios contratos de DARPA (dependiente del Ministerio de Defensa de EE.UU.) financiaron la implementación de TCP/IP que ha sido considerada hasta nuestros días como la referencia de los protocolos que hacen funcionar Internet (vinculando, de paso, el desarrollo de Internet y la expansión de las estaciones de trabajo con Unix). Muchas empresas utilizaron como base de sus versiones de Unix los desarrollos del CSRG dando lugar a sistemas muy conocidos en la época, como SunOS (Sun Microsystems) o Ultrix (Digital Equipment). De esta manera, Berkeley se convirtió en una de las dos fuentes fundamentales de Unix, junto con la "oficial", AT&T.

Para poder utilizar todo el código que producía el CSRG (y el de los colaboradores de la comunidad Unix que ellos de alguna manera coordinaban), hacía falta la licencia de Unix de AT&T, que cada vez era más difícil (y más cara) de conseguir, sobre todo si se quería el acceso al código fuente del sistema. Tratando de evitar en parte este problema, en junio de 1989 el CSRG liberó la parte de Unix relacionada con TCP/IP (la implementación de los protocolos en el núcleo y las utilidades), que no incluía código de AT&T. Fue la llamada *Networking Release 1* (Net-1). La licencia con que se liberó fue la famosa *licencia BSD*, que salvo ciertos problemas con sus cláusulas sobre obligaciones de anuncio, ha sido considerada siempre un ejemplo de licencia libre minimalista (que además de permitir la redistribución libre, permitía también su incorporación a productos privativos). Además, el CSRG probó un novedoso método de financiación (que ya estaba probando con éxito la FSF): vendía cintas con su distribución por 1.000 dólares. Aunque cualquiera podía a su vez redistribuir el contenido de las cintas sin ningún problema (ya que la licencia lo permitía), el CSRG vendió cintas a miles de organizaciones, con lo que consiguió fondos para seguir desarrollando.

Viendo el éxito de la distribución de Net-1, Keith Bostic propuso reescribir todo el código que aún quedaba del Unix original de AT&T. A pesar del escepticismo de algunos miembros del CSRG, realizó un anuncio público en el que pedía ayuda para realizar esta tarea, y poco a poco las utilidades (reescritas a partir de especificaciones) comenzaron a llegar a Berkeley. Mientras tanto, se realizó el mismo proceso con el núcleo, de manera que se reescribió de forma independiente la mayor parte del código que no habían realizado Berkeley ni colaboradores voluntarios. En junio de 1991, después de conseguir el permiso de la administración de la Universidad de Berkeley, se distribuyó la *Networking Release 2* (Net-2), con casi todo el código del núcleo y todas las utilidades de un sistema Unix completo. De nuevo el conjunto se distribuyó bajo la licencia BSD y se vendieron miles de cintas al precio de 1.000 dólares la unidad.

Sólo seis meses después de la liberación de Net-2, Bill Jolitz escribió el código que faltaba en el núcleo para que funcionase sobre arquitectura i386, liberando 386BSD, que fue distribuida por Internet. A partir de este código surgieron más tarde, en sucesión, todos los sistemas de la familia *BSD: primero apareció NetBSD, como una recopilación de los parches que se habían ido aportan-

do en la Red para mejorar 386BSD; más adelante apareció FreeBSD, como un intento de soportar fundamentalmente la arquitectura i386; varios años más tarde se formó el proyecto OpenBSD, con énfasis en la seguridad. Y también hubo una distribución propietaria basada en Net-2 (aunque era ciertamente original, ya que ofrecía a sus clientes todo el código fuente como parte de la distribución básica), que realizó de forma independiente la desaparecida empresa BSDI (Berkeley Software Design Inc.).

En parte como reacción a la distribución hecha por BSDI, Unix System Laboratories (USL), subsidiaria de AT&T que tenía los derechos de la licencia de Unix, puso una demanda judicial, primero a BSDI y luego a la Universidad de California. En ella los acusaba de distribuir su propiedad intelectual sin permiso. Después de varias maniobras judiciales (que incluyeron una contrademanda de la Universidad de California contra USL), Novell compró los derechos de Unix a USL, y en enero de 1994 llegó a un acuerdo extrajudicial con la Universidad de California. Como resultado de este acuerdo, el CSRG distribuyó la versión 4.4BSD-Lite, que pronto fue utilizada por todos los proyectos de la familia *BSD. Poco después (tras liberar aún la versión 4.4BSD-Lite Release 2), el CSRG desapareció. En ese momento hubo quien temió que fuera el fin de los sistemas *BSD, pero el tiempo ha demostrado que siguen vivos y coleando, con una nueva forma de gestión más típica de proyectos libres. Aún en la década de 2000 los proyectos que gestionan la familia *BSD son de los más antiguos y consolidados en el mundo del software libre.

Bibliografía

La historia de Unix BSD es muy ilustrativa de una forma peculiar de desarrollar software durante los años setenta y ochenta. Quien esté interesado en ella puede disfrutar de la lectura de "Twenty years of Berkeley Unix" (Marshall Kirk McKusick, 1999) [170], en la que se puede seguir su evolución desde la cinta que llevó Bob Fabry a Berkeley con la idea de hacer funcionar en un PDP-11 una de las primeras versiones del código de Thompson y Ritchie (comprada conjuntamente por los departamentos de informática, estadística y matemáticas), hasta las demandas judiciales de AT&T y las últimas liberaciones de código que dieron lugar a la familia de sistemas operativos libres *BSD.

2.2.3. Los comienzos de Internet

Casi desde su nacimiento, a principios de la década de 1970, Internet tuvo mucha relación con el software libre. Por un lado, desde sus comienzos, la comunidad de desarrolladores que la construyeron tuvo claros varios principios que luego se harían clásicos en el mundo del software libre; por ejemplo, la importancia de que los usuarios puedan ayudar a depurar errores o la compartición de código. La importancia de BSD Unix en su desarrollo (al proporcionar durante los años ochenta la implementación más popular de los protocolos TCP/IP) hizo que muchas costumbres y formas de funcionamiento pasasen fácilmente de una comunidad, la de desarrolladores alrededor del CSRG, a otra, la de los que estaban construyendo lo que entonces era NSFNet y luego sería Internet, y viceversa. Muchas de las aplicaciones básicas en el desarrollo

de Internet, como Sendmail (servidor de correo) o BIND (implementación del servicio de nombres) fueron libres y, en gran medida, fruto de esta colaboración entre comunidades.

Por último, a finales de los años ochenta y en la década de los noventa, la comunidad del software libre fue una de las primeras que exploró hasta el fondo las nuevas posibilidades que permitía Internet para la colaboración entre grupos geográficamente dispersos. Esta exploración hizo posible, en gran medida, la propia existencia de la comunidad BSD, la FSF o el desarrollo de GNU/Linux.

Uno de los aspectos más interesantes del desarrollo de Internet, desde el punto de vista del software libre, fue la gestión completamente abierta de sus documentos y sus normas. Aunque hoy pueda parecer algo normal (pues es lo habitual, por ejemplo, en el IETF o en el World Wide Web Consortium), en su época la libre disposición de todas las especificaciones y documentos de diseño, incluidas las normas que definen los protocolos, fue algo revolucionario y fundamental para su desarrollo. En *Netizens. On the history and impact of Usenet and the Internet* [139] (página 106) podemos leer:

> "Este proceso abierto promovía y llevaba al intercambio de información. El desarrollo técnico tiene éxito sólo cuando se permite que la información fluya libre y fácilmente entre las partes interesadas. La promoción de la participación es el principio fundamental que hizo posible el desarrollo de la Red."

Podemos observar cómo este párrafo podría ser suscrito, casi con toda seguridad, por cualquier desarrollador al referirse al proyecto de software libre en el que colabora.

En otra cita, en "The evolution of packet switching" [195] (página 267) podemos leer:

> "Como ARPANET era un proyecto público que conectaba muchas de las principales universidades e instituciones de investigación, los detalles de implementación y rendimiento se publicaban ampliamente."

Obviamente, esto es lo que suele ocurrir en los proyectos de software libre, donde toda la información relacionada con el proyecto (y no sólo la implementación) suele ser pública.

En este ambiente, y antes de que Internet, ya bien entrados los años noventa, se convirtiese sobre todo en un negocio, la comunidad de usuarios y su relación con los desarrolladores era clave. En aquella época muchas organizaciones aprendieron a confiar no en una única empresa proveedora del servicio de comunicación de datos, sino en una compleja combinación de empresas de servicios, fabricantes de equipos, desarrolladores profesionales y voluntarios, etc. Las mejores implementaciones de muchos programas no eran las que venían con el sistema operativo que se compraba con el hardware, sino implementaciones libres que rápidamente las sustituían. Los desarrollos más inno-

vadores eran fruto no de grandes planes de investigación en empresas, sino de estudiantes o profesionales que probaban ideas y recogían la realimentación que les enviaban cientos de usuarios de sus programas libres.

Como ya se ha dicho, Internet también proporcionó al software libre las herramientas básicas para colaborar a distancia. El correo electrónico, los grupos de noticias, los servicios de FTP anónimo (que fueron los primeros almacenes masivos de software libre) y, más tarde, los sistemas de desarrollo integrados basados en web han sido fundamentales (e imprescindibles) para el desarrollo de la comunidad del software libre tal como la conocemos y, en particular, para el funcionamiento de la inmensa mayoría de los proyectos de software libre. Desde el principio, proyectos como GNU o BSD hicieron un uso masivo e intenso de todos estos mecanismos, desarrollando, a la vez que las usaban, nuevas herramientas y sistemas que a su vez mejoraban Internet.

Bibliografía

El lector interesado en una historia de la evolución de Internet, escrita por varios de sus protagonistas, puede consultar "A brief history of the Internet" (comunicación de la ACM, 1997) [166].

2.2.4. Otros proyectos

Durante la década de 1980 vieron la luz otros importantes proyectos libres. Entre ellos destaca, por su importancia y proyección futura, el X Window (sistema de ventanas para sistemas de tipo Unix), desarrollado en el MIT, que fue uno de los primeros ejemplos de financiación a gran escala de proyectos libres con recursos de un consorcio de empresas. También merece la pena mencionar Ghostscript, un sistema de gestión de documentos PostScript desarrollado por una empresa, Aladdin Software, que fue uno de los primeros casos de búsqueda de un modelo de negocio basado en la producción de software libre.

A finales de los años ochenta hay ya en marcha toda una constelación de pequeños (y no tan pequeños) proyectos libres. Todos ellos, junto con los grandes proyectos mencionados hasta aquí, sentaron las bases de los primeros sistemas libres completos, que aparecieron a principios de la década de 1990.

2.3. Todo en marcha

Hacia 1990, gran parte de los componentes de un sistema informático completo estaban ya listos como software libre. Por un lado, el proyecto GNU y las distribuciones BSD habían completado la mayor parte de las aplicaciones que componen un sistema operativo. Por otro, proyectos como X Window o el propio GNU habían construido desde entornos de ventanas hasta compiladores, que muchas veces estaban entre los mejores de su género (por ejemplo, muchos administradores de sistemas SunOS o Ultrix sustituían para sus usuarios las aplicaciones propietarias de su sistema por las versiones libres de GNU o de BSD). Para tener un sistema completo construido sólo con software libre faltaba únicamente un componente: el núcleo. Dos esfuerzos separados e independientes vinieron a rellenar este hueco: 386BSD y Linux.

2.3.1. En busca de un núcleo

A finales de los ochenta y principios de los noventa, el proyecto GNU contaba con una gama básica de utilidades y herramientas que permitía tener el sistema operativo al completo. Ya entonces, muchas aplicaciones libres, entre las que fue especialmente interesante el caso de X Window, eran las mejores en su campo (utilidades Unix, compiladores...). Sin embargo, para completar el rompecabezas faltaba una pieza esencial: el núcleo del sistema operativo. El proyecto GNU estaba buscando esa pieza con un proyecto llamado Hurd, que pretendía construir un núcleo con tecnologías modernas.

2.3.2. La familia *BSD

Prácticamente en la misma época, la comunidad BSD estaba también en camino hacia un núcleo libre. En la distribución Net-2 sólo faltaban seis ficheros para tenerlo (el resto ya había sido construido por el CSRG o sus colaboradores). A principios de 1992, Bill Jolitz completó esos ficheros y distribuyó 386BSD, un sistema que funcionaba sobre arquitectura i386 y que con el tiempo daría lugar a los proyectos NetBSD, FreeBSD y OpenBSD. El desarrollo durante los meses siguientes fue rápido, y a finales de año ya era lo bastante estable como para ser usado en producción en entornos no críticos, que incluían, por ejemplo, un entorno de ventanas gracias al proyecto XFree (que había portado X Window a la arquitectura i386) o un compilador de gran calidad, GCC. Aunque había componentes que usaban otras licencias (como los procedentes del proyecto GNU, que usaban la GPL), la mayor parte del sistema se distribuía bajo la licencia BSD.

Bibliografía

Algunos de los episodios de esta época son ilustrativos de la potencia de los modelos de desarrollo de software libre. Es bien conocido el caso de Linus Torvalds, que desarrolló Linux mientras era estudiante de segundo curso de la Universidad de Helsinki. Pero no es el único caso de un estudiante que se abrió camino gracias a sus desarrollos libres. Por ejemplo, el alemán Thomas Roel portó X11R4 (una versión del sistema X Window) a un PC basado en un 386. Este desarrollo lo llevó a trabajar en Dell, y más adelante, a ser fundador de los proyectos X386 y XFree, fundamentales para que GNU/Linux y los *BSD tuvieran pronto un entorno de ventanas. Puede leerse más sobre la historia de XFree y el papel de Roel en ella en "The history of xFree86" (*Linux Magazine*, diciembre 1991) [135].

Luego vino la demanda de USL, que hizo que muchos usuarios potenciales temieran ser a su vez demandados si la Universidad de California perdía el juicio, o simplemente que el proyecto se parara. Quizás ésta fue la razón de que más adelante la base instalada de GNU/Linux fuera mucho mayor que la de todos los *BSD combinados. Pero eso es algo difícil de asegurar.

2.3.3. GNU/Linux entra en escena

En julio de 1991 Linus Torvalds (estudiante finés de veintiún años) pone el primer mensaje donde menciona su (por entonces) proyecto de hacer un sistema libre similar a Minix. En septiembre libera la primerísima versión (0.01), y cada pocas semanas aparecen nuevas versiones. En marzo de 1994 apareció

la versión 1.0, la primera denominada *estable*, pero el núcleo que había construido Linus era utilizable desde hacía bastantes meses. Durante este período, literalmente cientos de desarrolladores se vuelcan sobre Linux, integrando a su alrededor todo el software de GNU, XFree y muchos otros programas libres. A diferencia de los *BSD, el núcleo de Linux y gran parte de los componentes que se integran alrededor de él se distribuyen con la licencia GPL.

Bibliografía

La historia de Linux es probablemente una de las más interesantes (y conocidas) en el mundo del software libre. Se pueden encontrar muchos enlaces a información sobre ella en las páginas del décimo aniversario de su anuncio, aunque probablemente la más interesante es "History of Linux", de Ragib Hasan [138]. Como curiosidad, puede consultarse el hilo en el que Linus Torvalds anunciaba que estaba empezando a crear lo que luego fue Linux (en el grupo de noticias comp.os.minix) en http://groups.google.com/groups?th=d161e94858c4c0b9. Allí explica cómo lleva trabajando en su núcleo desde abril y cómo ya ha portado algunas herramientas del proyecto GNU sobre él (concretamente, menciona Bash y GCC).

Entre los muchos desarrollos aparecidos alrededor de Linux, uno de los más interesantes es el concepto de *distribución*[1]. Las primeras distribuciones aparecieron pronto, en 1992 (MCC Interim Linux, de la Universidad de Manchester; TAMU, de Texas A&M, y la más conocida, SLS, que más tarde dio lugar a Slackware, que aún se distribuye en la década de 2000), y han supuesto la entrada de la competencia en el mundo del empaquetamiento de sistemas alrededor de Linux. Cada distribución trata de ofrecer un GNU/Linux listo para usar, y partiendo todas del mismo software, han de competir en mejoras que su base de usuarios considere importantes. Además de proporcionar paquetes precompilados y listos para usar, las distribuciones suelen ofrecer sus propias herramientas para gestionar la selección, la instalación, la sustitución y la desinstalación de estos paquetes, así como la instalación inicial en un ordenador, y la gestión y la administración del sistema operativo.

(1)Este concepto se explica en detalle en la entrada correspondiente de Wikipedia, www.wikipedia.org/wiki/Linux_distribution

Con el tiempo, unas distribuciones han ido sucediéndose a otras como las más populares. Entre todas ellas, cabe destacar las siguientes:

1) Debian, desarrollada por una comunidad de desarrolladores voluntarios.

2) Red Hat Linux, que primero desarrolló internamente la empresa Red Hat, pero que más adelante adoptó un modelo más comunitario, dando lugar a Fedora Core.

3) Suse, que dio lugar a OpenSUSE, en una evolución similar a la de Red Hat.

4) Mandriva (sucesora de Mandrake Linux y de Conectiva).

5) Ubuntu, derivada de Debian y producida a partir de ella por la empresa Canonical.

2.4. Tiempos de maduración

A mediados de la década de 2000, GNU/Linux, OpenOffice.org o Firefox tienen una presencia relativamente habitual en los medios de comunicación. La inmensa mayoría de empresas utiliza software libre al menos para algunos de sus procesos informáticos. Es difícil ser un estudiante de informática y no utilizar software libre en grandes cantidades. El software libre ha dejado de ser una nota a pie de página en la historia de la informática para convertirse en algo muy importante para el sector. Las empresas de informática, las del sector secundario (las que utilizan intensivamente software, aunque su actividad principal es otra) y las administraciones públicas empiezan a considerarlo como algo estratégico. Y está llegando, poco a poco pero con fuerza, a los usuarios domésticos. En líneas generales, se entra en una época de maduración.

Y en el fondo, empieza a surgir una pregunta muy importante, que de alguna forma resume lo que está ocurriendo: "¿estamos ante un nuevo modelo de industria software?". Aún podría ocurrir, quizás, que el software libre no llegara a ser más que una moda pasajera que con el tiempo sólo fuera recordada con nostalgia. Pero también podría ser (y ello parece cada vez más plausible) un nuevo modelo que está aquí para quedarse, y tal vez para cambiar radicalmente una de las industrias más jóvenes, pero también de las más influyentes.

2.4.1. Finales de la década de los noventa

A mediados de la década de los noventa, el software libre ofrecía ya entornos completos (distribuciones de GNU/Linux, sistemas *BSD...) que permitían el trabajo diario de mucha gente, sobre todo de desarrolladores de software. Aún había muchas asignaturas pendientes (la principal, disponer de mejores interfaces gráficas de usuario en una época en la que Windows 95 era considerado el estándar), pero ya había unos cuantos miles de personas en todo el mundo que sólo usaban software libre en su trabajo diario. Los anuncios de nuevos proyectos se sucedían y el software libre comenzaba su largo camino hacia las empresas, los medios de comunicación y, en general, el conocimiento público.

De esta época es también el despegue de Internet como red para todos, en muchos casos de la mano de programas libres (sobre todo en su infraestructura). La llegada del web a los hogares de millones de usuarios finales consolida esta situación, al menos en lo que se refiere a servidores: los servidores web (HTTP) más populares siempre han sido libres (primero el servidor del NCSA, luego Apache).

Quizás el comienzo del camino del software libre hasta su puesta de largo en la sociedad pueda situarse en el célebre ensayo de Eric Raymond, "La catedral y el bazar" (Eric S. Raymond, 2001) [192]. Aunque mucho de lo expuesto en él era ya bien conocido por la comunidad de desarrolladores de software libre, reunirlo en un artículo y darle una gran difusión lo convirtió en una influyente

herramienta de promoción del concepto de *software libre* como mecanismo de desarrollo alternativo al usado por la industria del software tradicional. Otro artículo muy importante en esta época fue "Setting up shop. The Business of open-source software" [141], de Frank Hecker, que por primera vez expuso los modelos de negocio posibles en torno al software libre y que fue escrito para influir en la decisión sobre la liberación del código del navegador de Netscape.

Si el artículo de Raymond supuso una gran herramienta de difusión de algunas de las características fundamentales del software libre, la liberación del código del navegador de Netscape fue el primer caso en que una empresa relativamente grande, de un sector muy innovador (la entonces naciente industria del web), tomaba la decisión de liberar como software libre uno de sus productos. En aquella época, Netscape Navigator estaba perdiendo la batalla de los navegadores web frente al producto de Microsoft (Internet Explorer), en parte por las tácticas de Microsoft de combinarlo con su sistema operativo. Para muchos, Netscape hizo lo único que podía hacer: tratar de cambiar las reglas para poder competir con un gigante. Y de este cambio de reglas (tratar de competir con un modelo de software libre) nació el proyecto Mozilla. Este proyecto, no sin problemas, ha llevado varios años después a un navegador que, si bien no ha recuperado la enorme cuota de mercado que tuvo en su día Netscape Navigator, parece que técnicamente es al menos tan bueno como sus competidores privativos.

En cualquier caso, y con independencia de su éxito posterior, el anuncio de Netscape de liberar el código de su navegador supuso un fuerte impacto en la industria del software. Muchas empresas comenzaron a considerar el software libre como digno de consideración.

También los mercados financieros se empezaron a ocupar del software libre. En plena euforia de las puntocom, varias empresas de software libre se convierten en objetivo de inversores. Quizás el caso más conocido es el de Red Hat, una de las primeras empresas que reconocieron que la venta de CD con sistemas GNU/Linux listos para usar podía ser un modelo de negocio. Red Hat comenzó distribuyendo su Red Hat Linux, haciendo gran énfasis (al menos para lo habitual en la época) en la facilidad de manejo y mantenimiento del sistema por personas sin conocimientos específicos de informática. Con el tiempo, fue diversificando su negocio, manteniéndose en general en la órbita del software libre, y en septiembre de 1998 anunció que Intel y Netscape habían invertido en ella. "Si es bueno para Intel y Netscape, seguro que es bueno para nosotros", debieron de pensar muchos inversores. Cuando Red Hat salió a bolsa en el verano de 1999, la oferta pública de acciones fue suscrita completamente, y pronto el valor de cada título subió espectacularmente. Fue la primera vez que una empresa consiguía financiación del mercado de valores con un modelo basado en el software libre. Pero no fue la única: lo mismo hicieron más tarde otras como VA Linux o Andover.net (que más tarde fue adquirida por VA Linux).

Nota

Red Hat proporciona una lista de hitos históricos relacionados con su empresa en http://fedora.redhat.com/about/history/.

En esta época nacieron también muchas empresas basadas en modelos de negocio en torno al software libre. Sin salir a bolsa y sin lograr tan estupendas capitalizaciones, fueron sin embargo muy importantes para el desarrollo del software libre. Por ejemplo, aparecieron muchas empresas que empezaron distribuyendo sus propias versiones de GNU/Linux, como SuSE (Alemania), Conectiva (Brasil) o Mandrake (Francia), que más tarde se uniría a la anterior para formar Mandriva. Otras proporcionaban servicios a empresas que ya demandaban mantenimiento y adaptación de productos libres: LinuxCare (EE.UU.), Alcove (Francia), ID Pro (Alemania); y muchas más.

Por su parte, los gigantes del sector también empezaron a posicionarse ante el software libre. Algunas empresas, como IBM, lo incorporaron directamente en su estrategia. Otras, como Sun Microsystems, mantuvieron con él una curiosa relación, a veces de apoyo, a veces de indiferencia, a veces de enfrentamiento. La mayoría (como Apple, Oracle, HP, SGI, etc.) exploraron el modelo del software libre con diversas estrategias, que iban desde la liberación selectiva de software hasta el simple porte a GNU/Linux de sus productos. Entre estos dos extremos, hubo otras muchas líneas de acción, como la utilización más o menos intensa de software libre en sus productos (como fue el caso del Mac OS X) o la exploración de modelos de negocio basados en el mantenimiento de productos libres.

Desde el punto de vista técnico, lo más destacable de esta época fue probablemente la aparición de dos ambiciosos proyectos al objeto de conseguir llevar el software libre al entorno de escritorio (*desktop*) de los usuarios no muy versados en la informática: KDE y GNOME. Dicho de forma muy simplista, el objetivo final era que no hubiera que usar la línea de órdenes para interaccionar con GNU/Linux o *BSD ni con los programas sobre esos entornos.

KDE fue anunciado en octubre de 1996. Utilizando las bibliotecas gráficas Qt (por aquel entonces un producto privativo de la empresa Trolltech, pero gratuito para su uso sobre GNU/Linux[2]), iniciaron la construcción de un conjunto de aplicaciones de escritorio que funcionasen de forma integrada y que tuviesen un aspecto uniforme. En julio de 1998 liberaron la versión 1.0 del K Desktop Environment, que fue pronto seguida de nuevas versiones cada vez más completas y maduras. Las distribuciones de GNU/Linux pronto incorporaron KDE como escritorio para sus usuarios (o al menos como uno de los entornos de escritorio que sus usuarios podían elegir).

[(2)]Más tarde, Qt pasó a ser distribuido bajo una licencia libre, la QPL (Qt Public License), no compatible con la GPL, lo que causaba algún problema, puesto que la mayor parte de KDE estaba distribuida bajo la GPL. Con en el tiempo, finalmente Trolltech decidió distribuir Qt bajo la licencia GPL, con lo que estos problemas terminaron.

En gran medida como reacción a la dependencia que tenía KDE de la biblioteca propietaria Qt, en agosto de 1997 se anunció el proyecto GNOME (Miguel de Icaza, "The story of the GNOME Project") [101], con fines y características muy similares a los de KDE, pero con el objetivo explícito de que todos sus

componentes fueran libres. En marzo de 1999 se liberó GNOME 1.0, que con el tiempo también se iría mejorando y estabilizando. A partir de ese momento, la mayor parte de las distribuciones de sistemas operativos libres (y muchos derivados de Unix privativos) ofrecieron como opción el escritorio de GNOME o el de KDE y las aplicaciones de ambos entornos.

Simultáneamente, los principales proyectos de software libre que ya estaban en marcha continúan con buena salud y surgen nuevos proyectos cada día. En varios nichos de mercado, se observa cómo la mejor solución (reconocida por casi todo el mundo) es software libre. Por ejemplo, Apache ha mantenido casi desde su aparición en abril de 1995 la mayor cuota de mercado entre los servidores web; XFree86, el proyecto libre que desarrolla X Window, es con diferencia la versión de X Window más popular (y por tanto, el sistema de ventanas para sistemas de tipo Unix más extendido); GCC es reconocido como el compilador de C más portable y uno de los de mejor calidad; GNAT, sistema de compilación para Ada 95, se hace con la mayor parte del mercado de compiladores Ada en pocos años; y así sucesivamente.

En 1998 se creó la Open Source Initiative (OSI), que decidió adoptar el término *open source software* ('software de fuente abierta') como una marca para introducir el software libre en el mundo comercial, tratando de evitar la ambigüedad que en inglés supone el término *free* (que significa tanto 'libre' como 'gratis'). Esta decisión supuso (y aún supone) uno de los debates más enconados del mundo del software libre, ya que la Free Software Foundation y otros consideraron que era mucho más apropiado hablar de *software libre* (Richard Stallman, "Why *free software* is better than *open source*", 1998) [206]. En cualquier caso, la OSI realizó una fructífera campaña de difusión de su nueva marca, que ha sido adoptada por muchos como la forma preferida de hablar del software libre, sobre todo en el mundo anglosajón. Para definir el software *open source*, la OSI utilizó una definición derivada de la que utiliza el proyecto Debian para definir el software libre ("Debian free software guidelines", http://www.debian.org/social_contract.html#guidelines) [104], que por otra parte refleja con bastante aproximación la idea de la FSF al respecto ("Free software definition", http://www.gnu.org/philosophy/free-sw.html) [120], por lo que desde el punto de vista práctico casi cualquier programa que es considerado software libre es también considerado *open source* y viceversa. Sin embargo, las comunidades de software libre y de software de fuente abierta (o al menos las personas que se identifican como parte de una o de otra) pueden ser profundamente diferentes.

2.4.2. Década de 2000

A principios de la década de 2000 el software libre es ya un serio competidor en el segmento de servidores y comienza a estar listo para el escritorio. Sistemas como GNOME, KDE, OpenOffice.org y Mozilla Firefox pueden ser utilizados por usuarios domésticos y son suficientes para las necesidades de muchas em-

presas, al menos en lo que a ofimática se refiere. Los sistemas libres (y sobre todo los basados en Linux) son fáciles de instalar, y la complejidad para mantenerlos y actualizarlos es comparable a la de otros sistemas privativos.

En estos momentos, cualquier empresa de la industria del software tiene una estrategia con respecto al software libre. La mayoría de las grandes multinacionales (IBM, HP, Sun, Novell, Apple, Oracle...) incorpora el software libre con mayor o menor decisión. En un extremo podríamos situar empresas como Oracle, que reaccionan simplemente portando sus productos a GNU/Linux. En el otro podría situarse IBM, que tiene la estrategia más decidida y ha realizado las mayores campañas de publicidad sobre GNU/Linux. Entre los líderes del mercado informático, sólo Microsoft se ha significado con una estrategia claramente contraria al software libre y en particular al software distribuido bajo licencia GPL.

En cuanto al mundo del software libre en sí mismo, a pesar de los debates que de vez en cuando sacuden a la comunidad, el crecimiento es enorme. De día en día hay más desarrolladores, más proyectos de software libre activos, más usuarios... Cada vez más el software libre está pasando de ser algo marginal a convertirse en un competidor que hay que tener en cuenta.

Ante este desarrollo, aparecen nuevas disciplinas que estudian específicamente el software libre, como la ingeniería del software libre. A partir de sus investigaciones comenzamos, poco a poco, a entender cómo funciona el software libre en sus diferentes aspectos: modelos de desarrollo, modelos de negocio, mecanismos de coordinación, gestión de proyectos libres, motivación de desarrolladores, etc.

En estos años empiezan también a verse los primeros efectos de la *deslocalización* que permite el desarrollo de software libre: países considerados "periféricos" participan en el mundo del software libre de forma muy activa. Por ejemplo, es significativo el número de desarrolladores mexicanos o españoles (ambos países con poca tradición de industria software) en proyectos como GNOME (Lancashire, "Code, culture and cash: the fading altruism of open source development", 2001) [164]. Y es aún más interesante el papel de Brasil, que está teniendo una gran cantidad de desarrolladores y expertos en tecnologías de software libre y un decidido apoyo por parte de las administraciones públicas. Mención aparte merece el caso de gnuLinEx, muy significativo de cómo una región con poca tradición de desarrollo de software puede tratar de cambiar la situación con una estrategia agresiva de implantación de software libre.

Desde el punto de vista de la toma de decisiones a la hora de implantar soluciones software, hay que destacar el hecho de que hay ciertos mercados (como los servicios de Internet o la ofimática) en los que el software libre se ha convertido en una opción natural cuya no consideración es difícil de justificar cuando se está estudiando qué tipo de sistema utilizar.

En el lado negativo, estos años han visto cómo el entorno legal en el que se mueve el software libre está cambiando rápidamente en todo el mundo. Por una parte, las patentes de software (patentes de programación) son consideradas cada vez en más países. Por otro, las nuevas leyes de protección de derechos de autor dificultan o hacen imposible el desarrollo de aplicaciones libres en algunos ámbitos, el más conocido de los cuales es el de los visores de DVD (debido al algoritmo CSS de ofuscación de imágenes que se utiliza en esa tecnología).

gnuLinEx

A principios de 2002 la Junta de Extremadura dio a conocer públicamente el proyecto gnuLinEx. La idea era simple: promover la creación de una distribución basada en GNU/Linux con el objetivo fundamental de utilizarla en los miles de ordenadores que se van a instalar en los centros educativos públicos de toda la región. Extremadura, situada en la parte occidental de España, fronteriza con Portugal, cuenta con aproximadamente un millón de habitantes y nunca se había destacado por sus iniciativas tecnológicas. De hecho, la región prácticamente carecía de industria de software.

En este contexto, gnuLinEx ha supuesto una aportación muy interesante en el panorama del software libre a escala mundial. Mucho más allá de ser una nueva distribución de GNU/Linux basada en Debian (lo que no deja de ser algo relativamente anecdótico), y más allá de su enorme impacto en los medios de comunicación (es la primera vez que Extremadura ha sido portada de *The Washington Post* y una de las primeras que lo ha sido un producto de software libre), lo extraordinario es la (al menos aparentemente) sólida apuesta de una administración pública por el software libre. La Junta de Extremadura decidió probar un modelo diferente en cuanto al software usado para la enseñanza y más adelante a todo el uso de la informática dentro de sus competencias. Esto la ha convertido en la primera administración pública de un país desarrollado que toma decididamente este camino. En torno a la iniciativa de la Junta se ha producido también mucho movimiento, tanto dentro como fuera de Extremadura: hay academias que enseñan informática con gnuLinEx; se han escrito libros para apoyar esta enseñanza; se venden ordenadores con gnuLinEx preinstalado. En general, se intenta crear alrededor de esta experiencia todo un tejido docente y empresarial que le proporcione soporte. Y la experiencia se ha exportado. A principios del siglo XXI varias comunidades autónomas en España han apostado (de una u otra forma) por el software libre para la enseñanza, y en general, su importancia para las administraciones públicas es ampliamente reconocida.

Knoppix

Desde finales de los años noventa hay distribuciones de GNU/Linux que se instalan fácilmente, pero probablemente Knoppix, cuya primera versión apareció en 2002, ha llevado este concepto a su máxima expresión. Se trata de un

CD que arranca prácticamente en cualquier PC, convirtiéndolo (sin tener siquiera que formatear el disco, ya que permite su uso "en vivo") en una máquina GNU/Linux completamente funcional, con una selección de las herramientas más habituales. Knoppix une una buena detección automática de hardware con una buena selección de programas y un funcionamiento "en vivo". Permite, por ejemplo, una experiencia rápida y directa de qué puede suponer trabajar con GNU/Linux. Y está dando lugar a toda una familia de distribuciones del mismo tipo, especializadas para necesidades específicas de un perfil de usuarios.

OpenOffice.org

En 1999 Sun Microsystems compró una empresa alemana llamada Stardivision, cuyo producto estrella era StarOffice, un juego de herramientas ofimático similar en funcionalidad a Office, el juego de herramientas de Microsoft. Un año más tarde, Sun distribuyó gran parte del código de StarOffice bajo una licencia libre (la GPL), dando lugar al proyecto OpenOffice.org. Este proyecto liberó la versión 1.0 de OpenOffice.org en mayo de 2002. OpenOffice.org se ha convertido en un juego de aplicaciones ofimáticas de calidad y funcionalidad similar a la de cualquier otro producto ofimático y, lo que es más importante, interopera muy bien con los formatos de datos de Microsoft Office. Estas características han hecho de ella la aplicación de referencia del software libre en el mundo de la ofimática.

La importancia de OpenOffice.org, desde el punto de vista de extensión del software libre a un gran número de usuarios, es enorme. Por fin ya es posible cambiar, prácticamente sin traumas, de los entornos privativos habituales en ofimática (sin duda la aplicación estrella en el mundo empresarial) a entornos completamente libres (por ejemplo, GNU/Linux más GNOME y/o KDE más OpenOffice.org). Además, la transición puede hacerse de forma muy suave: como OpenOffice.org funciona también sobre Microsoft Windows, no es preciso cambiar de sistema operativo para experimentar en profundidad el uso de software libre.

Mozilla, Firefox y los demás

Prácticamente desde su aparición en 1994 hasta 1996, Netscape Navigator fue el líder indiscutible del mercado de navegadores web, con cuotas de hasta el 80%. La situación empezó a cambiar cuando Microsoft incluyó Internet Explorer en su Windows 95, lo que supuso que poco a poco Netscape Navigator fuera perdiendo cuota de mercado. A principios de 1998 Netscape anunció que iba a distribuir gran parte del código de su navegador como software libre, cosa que efectivamente hizo en marzo del mismo año, lanzando el proyecto Mozilla. Durante bastante tiempo dicho proyecto estuvo rodeado de incerti-

dumbre, e incluso de pesimismo (por ejemplo cuando su líder, Jamie Zawinski, lo abandonó), dado que pasaba el tiempo y no aparecía ningún producto como resultado de su lanzamiento.

En enero de 2000, el proyecto liberó Mozilla M13, que fue considerada la primera versión razonablemente estable. Pero sólo en mayo de 2002 se publicó finalmente la versión 1.0, la primera oficialmente estable, más de cuatro años después de la liberación del primer código de Netscape Navigator.

Bibliografía

En "Netscape Navigator", de Brian Wilson, [234], puede consultarse una reseña detallada de las principales versiones de Netscape Navigator y Mozilla, así como de sus principales características.

Por fin Mozilla era una realidad, aunque quizás demasiado tarde, si tenemos en cuenta las cuotas de mercado que tuvo Internet Explorer durante 2002 ó 2003 (en los que fue líder indiscutible relegando a Mozilla y a otros a una posición completamente marginal). Pero el proyecto Mozilla, a pesar de tardar tanto, dio sus frutos; no sólo los esperados (el navegador Mozilla), sino muchos otros que podrían considerarse "colaterales", como por ejemplo Firefox, otro navegador basado en el mismo motor de HTML, que con el tiempo se ha convertido en el producto principal, y que desde su aparición en 2005 está consiguiendo erosionar poco a poco las cuotas de mercado de otros navegadores.

El proyecto Mozilla ha ayudado a completar un gran hueco en el mundo del software libre. Antes de la aparición de Konqueror (el navegador del proyecto KDE), no había muchos navegadores libres con interfaz gráfica. A partir de la publicación de Mozilla, han ido apareciendo gran cantidad de proyectos basados en él que han producido una buena cantidad de navegadores. Por otro lado, la combinación de Mozilla Firefox y OpenOffice.org permite usar software libre para las tareas más cotidianas, incluso en un entorno Microsoft Windows (ambos funcionan no sólo sobre GNU/Linux, *BSD y otros sistemas de tipo Unix, sino que también lo hacen sobre Windows). Esto permite, por primera vez en la historia del software libre, que la transición de software privativo a software libre en entornos de oficina sea simple: se puede empezar utilizando estas dos aplicaciones sobre Windows, sin cambiar de sistema operativo (en el caso de los que lo usan habitualmente), y con el tiempo eliminar la única pieza no libre y pasar a GNU/Linux o a FreeBSD.

El caso SCO

A principios de 2003 la corporación SCO (anteriormente Caldera Systems y Caldera International) interpuso una demanda contra IBM por supuesta infracción de sus derechos de propiedad intelectual. Aunque la demanda era compleja, estaba centrada en la acusación de que IBM había contribuido al núcleo de Linux con código que era de SCO. En mayo de 2007 el asunto todavía no estaba resuelto y se había complicado aún más con más demandas (de IBM y Red Hat contra SCO, de SCO contra AutoZone y DaimlerChrysler, dos grandes usuarios informáticos) y con campañas de SCO en las que amenazaba con demandar a grandes empresas que usan Linux, etc.

Aunque el ganador de esta gran batalla legal aún no se conoce, el asunto ha puesto de relieve ciertos aspectos legales del software libre. En particular, muchas empresas se han planteado los problemas a los que se pueden enfrentar al usar Linux y otros programas libres y qué garantías tienen de que al hacerlo no están infringiendo derechos de propiedad intelectual o industrial de terceros.

De alguna manera, este caso y algunos otros (como los relacionados con la validez de la GPL que se han resuelto en Alemania en 2005) pueden interpretarse también como un síntoma de la madurez del software libre. Ya ha dejado de ser un elemento extraño al mundo empresarial y ha entrado a formar parte de muchas de sus actividades (incluidas las que tienen que ver con estrategias legales).

Ubuntu, Canonical, Fedora y Red Hat

Aunque Canonical (la empresa que produce y comercializa Ubuntu) podría considerarse casi como una recién llegada al negocio de las distribuciones GNU/Linux, sus actividades merecen que le dediquemos atención. En relativamente poco tiempo, Ubuntu se ha establecido como una de las distribuciones más conocidas y utilizadas, con fama de buena calidad y mucha simplicidad de instalación y uso. Ubuntu también se caracteriza por tener mucho más cuidado en incluir fundamentalmente software libre que la mayoría de las demás distribuciones producidas por empresas.

Sin embargo, la característica probablemente fundamental de Ubuntu (y de la estrategia de Canonical) ha sido basarse en Debian, una distribución creada y mantenida por voluntarios. De hecho, Ubuntu no ha sido el primer caso de distribución basada en Debian (otro caso bien conocido es gnuLinEx), pero quizás sí que ha sido, entre todas ellos, el que más recursos ha recibido. Por ejemplo, Canonical ha contratado a una gran cantidad de expertos en Debian (muchos de los cuales participan en el proyecto) y ha seguido una estrategia que parece buscar la colaboración con el proyecto voluntario. De alguna manera, Canonical ha tratado de completar lo que considera que falta en Debian para tener una aceptación por el usuario medio.

Red Hat, por su parte, ha seguido un camino diferente para llegar a una situación bastante similar. Partiendo de una distribución realizada completamente con sus propios recursos, decidió colaborar con Fedora, un grupo de voluntarios que ya estaba trabajando con distribuciones basadas en Red Hat, para producir Fedora Core, su distribución "popular". Red Hat mantiene su versión para empresas, pero esta colaboración con voluntarios es, en el fondo, similar a la que ha dado lugar a Ubuntu.

Quizás todos estos movimientos no son más que el fruto de la feroz competencia que tiene lugar en el mercado de distribuciones GNU/Linux y de una tendencia de más calado: la colaboración de empresas con voluntarios (con *la comunidad*) para producir software libre.

Las distribuciones particularizadas

Desde que Linux entró en escena, muchos grupos y empresas han creado su propia distribución basada en él. Pero durante estos años, el fenómeno se ha extendido a muchas organizaciones y empresas que quieren tener una distribución particularizada para sus propias necesidades. El abaratamiento del proceso de particularización de una distribución y la amplia disposición del conocimiento técnico para hacerlo han permitido la expansión de esta actividad, que se ha convertido incluso en un nicho de negocio para algunas empresas.

Quizás uno de los casos más conocidos de distribuciones particularizadas es el de las distribuciones autonómicas en España. La Junta de Extremadura comenzó con su gnuLinEx una tendencia que han seguido muchas otras comunidades autónomas. El proceso es tan común que varias de ellas convocan de forma regular concursos públicos para la creación y el mantenimiento de las nuevas versiones de sus distribuciones.

La creación de distribuciones particularizadas hace real una tendencia de la que se hablaba desde hacía tiempo en el mundo del software libre: la adaptación de los programas a las necesidades específicas de los usuarios, sin que haga falta que los productores originales tengan necesariamente que realizar esta adaptación.

Bibliografía

Algunas de las distribuciones autonómicas más conocidas de GNU/Linux:

- gnuLinEx: http://linex.org (Extremadura)
- Guadalinex: http://guadalinex.org (Andalucía)
- Lliurex: http://lliurex.net (Comunidad Valenciana)
- Augustux: http://www.zaralinux.org/proy/augustux/ (Aragón)
- MAX: http://www.educa.madrid.org/web/madrid_linux/ (Madrid)
- MoLinux: http://molinux.info (Castilla-La Mancha)

Colaboración de empresas con empresas y de voluntarios con empresas

Desde prácticamente el principio del software libre ha habido empresas que colaboraban con voluntarios en el desarrollo de aplicaciones. Sin embargo, durante estos años en los que parece que se está llegando a la madurez, cada vez son más las empresas que usan el software libre como parte de su estrategia para colaborar con otras empresas, cuando eso les resulta interesante. Dos

de los casos más significativos, organizados específicamente con este fin, son ObjectWeb (alianza nacida en Francia que con el tiempo pasó a ser claramente internacional) y Morfeo (en España). En ambos casos, un grupo de empresas se ha puesto de acuerdo para desarrollar un conjunto de sistemas libres que les resultan interesantes y que deciden distribuir como software libre.

En otros casos, las empresas buscan activamente bien colaborar en proyectos libres promovidos por voluntarios, o bien tratar de que sean los voluntarios los que vayan a colaborar a sus propios proyectos libres. Ejemplos de la primera situación son la GNOME Foundation o el ya mencionado Ubuntu con respecto a Debian. Entre los segundos, se puede destacar el caso de Sun y OpenOffice.org y OpenSolaris, o el de Red Hat con Fedora Core.

Extensión a otros ámbitos

El software libre ha demostrado que, en el campo de la producción de programas, es posible otra forma de hacer las cosas. Se ha visto en la práctica cómo otorgando las libertades de distribución, modificación y uso se puede conseguir la sostenibilidad, bien mediante trabajo voluntario, bien incluso mediante una generación de negocio que permita la supervivencia de las empresas.

Con el tiempo, esta misma idea se está trasladando a otros campos de producción de obra intelectual. Las licencias Creative Commons han permitido simplificar el proceso de liberación en campos como la literatura, la música o el vídeo. Wikipedia está mostrando que en un área tan particular como la producción de enciclopedias se puede recorrer un camino muy interesante. Y cada vez hay más autores literarios, grupos musicales e incluso productoras de películas interesados en modelos libres de producción y distribución.

En todos estos dominios queda mucho camino por recorrer, y en casi todos ellos la práctica aún no ha demostrado completamente que es posible la creación sostenible con modelos libres. Pero no se puede negar que la experimentación al respecto está entrando en estado de ebullición.

El software libre como objeto de estudio

Aunque algunos trabajos, como el conocido "La catedral y el bazar" empezaron a abrir el camino del estudio del software libre como tal, no fue hasta el año 2001 y siguientes cuando la comunidad académica comenzó a considerar el software libre como un objeto digno de estudio. Con el tiempo, la gran disponibilidad de datos (casi todo en el mundo del software libre es público y está disponible en almacenes de información públicos) y las novedades que se observan en él han ido centrando la atención de muchos grupos. A mediados de la década de 2000 hay ya varios congresos internacionales que se dedican

específicamente al software libre, las revistas más prestigiosas le dedican con cierta regularidad monográficos, y las agencias que financian la investigación están abriendo líneas orientadas específicamente a él.

2.5. El futuro: ¿una carrera de obstáculos?

Sin duda, es difícil predecir el futuro. Y desde luego, no es algo a lo que nos vayamos a dedicar aquí. Por lo tanto, más que tratar de explicar cómo será el futuro del software libre, intentaremos mostrar los problemas que previsiblemente tendrá que afrontar (y de hecho lleva ya tiempo afrontando). De cómo sea el mundo del software libre capaz de superar estos obstáculos dependerá, indudablemente, su situación dentro de unos años.

- Técnica FUD (*fear, uncertainity, doubt*, o en español, 'miedo, desconocimiento, duda'). Se trata de una técnica bastante habitual en el mundo de las tecnologías de la información y que ha sido utilizada por los competidores de productos de software libre para tratar de desacreditarlos, con mayor o menor razón y con éxito variable. En líneas generales, el software libre, quizás debido a su complejidad y a diversos métodos de penetración en las empresas, ha resultado bastante inmune a estas técnicas.

- Disolución. Muchas empresas están probando los límites del software libre como modelo, y en particular tratan de ofrecer a sus clientes modelos que presentan algunas características similares al software libre. El principal problema que puede presentar este tipo de modelos es la confusión que genera en los clientes y los desarrolladores, que tienen que estudiar con mucho detalle la letra pequeña para darse cuenta de que lo que se les está ofreciendo no tiene las ventajas que para ellos supone el software libre. El caso más conocido de modelos de este tipo es el programa Shared Source, de Microsoft.

- Desconocimiento. En muchos casos los usuarios llegan al software libre simplemente porque creen que es gratis; o porque consideran que está "de moda". Si no profundizan más y no estudian con cierto detenimiento las ventajas que les puede ofrecer el software libre como modelo, corren el riesgo de no aprovecharse de ellas. En muchos casos, las suposiciones de partida en el mundo del software libre son tan diferentes de las habituales en el mundo del software privativo que es indispensable un mínimo análisis para comprender que lo que en un caso es habitual en el otro puede ser imposible, y viceversa. El desconocimiento, por lo tanto, no puede sino generar insatisfacciones y pérdida de oportunidades en cualquier persona u organización que se aproxime al software libre.

- Impedimentos legales. Sin duda éste es el principal problema con el que se va a encontrar el software libre en los próximos años. Aunque el entorno legal en el que se desarrolló el software libre durante la década de 1980 y la primera mitad de la de 1990 no era ideal, al menos dejaba suficien-

te espacio como para que creciese en libertad. Desde entonces, la extensión del ámbito de la patentabilidad al software (que se ha producido en muchos países desarrollados) y las nuevas legislaciones sobre derechos de autor (que limitan la libertad de creación del desarrollador de software) suponen cada vez barreras más altas a la entrada del software libre en segmentos importantes de aplicaciones.

2.6. Resumen

En este capítulo se presenta la historia del software libre. Los años sesenta fueron una etapa dominada por los grandes ordenadores e IBM en la que el software se distribuía junto al hardware, y habitualmente con el código fuente. En la década de los setenta se comenzó a vender el software por separado, y rápidamente la distribución privativa, que no incluye el código fuente y que no otorga permiso de modificación o redistribución, se convirtió casi en la única opción.

Ved también

El lector interesado podrá encontrar en el apéndice B una lista de algunas de las fechas más relevantes de la historia del software libre.

En la década de 1970 comenzó el desarrollo del sistema operativo Unix en los Bell Labs de AT&T, que dio lugar más adelante a Unix BSD. Su evolución, paralela al nacimiento de Internet, sirvió de campo de pruebas para nuevas formas de desarrollo en colaboración que fueron luego habituales en el mundo del software libre.

En 1984 Richard Stallman comenzó a trabajar en el proyecto GNU, fundó la Free Software Foundation (FSF), escribió la licencia GPL y, en general, sentó los fundamentos del software libre tal como ha sido conocido más tarde.

En la década de 1990 Internet fue madurando y proporcionando a las comunidades de software libre nuevos canales de comunicación y distribución. En 1991, Linus Torvalds comenzó a desarrollar un núcleo libre (Linux) que permitió completar el sistema GNU, que contaba ya con casi todas las piezas para convertirse en un sistema completo similar a Unix: compilador de C (GCC), editor (Emacs), sistema de ventanas (X Window), etc. Nacieron así los sistemas operativos GNU/Linux, que fructificaron en multitud de distribuciones, como Red Hat Linux y Debian GNU/Linux. A finales de la década de 1990 estos sistemas se completaban con dos entornos de escritorio: KDE y GNOME.

En la década de 2000 el software libre llega a liderar algunos sectores (como el de servidores web, dominado por Apache), y aparecen nuevas herramientas que cubren gran cantidad de necesidades informáticas.

3. Aspectos legales

"The licenses for most software are designed to take away your freedom to share and change it."

"Las licencias de la mayoría de los programas están diseñadas para quitarte la libertad de compartirlos y cambiarlos."

GNU General Public License, versión 2

En este capítulo se presentan los principales aspectos legales relacionados con el software libre. Para ponerlos en contexto, se empieza por una pequeña introducción a los conceptos más básicos de la propiedad intelectual e industrial, antes de exponer la definición detallada de *software libre*, *software de fuente abierta* y otros conceptos relacionados. Se analizan también con cierto detalle las licencias de software libre más habituales y su impacto sobre los modelos de negocio (tema que se tratará con más detalle en el capítulo 5) y los modelos de desarrollo.

3.1. Breve introducción a la propiedad intelectual

El término *propiedad intelectual* tiene varias acepciones según el contexto y quién lo utiliza. Hoy en día se usa en muchos foros para agrupar distintos privilegios que se otorgan sobre bienes intangibles con valor económico. Entre ellos podemos destacar los de *copyrigh*t (derechos de autor) y similares, que protegen de la copia no autorizada trabajos literarios o artísticos, programas de ordenador, recopilaciones de datos, diseños industriales, etc.; las marcas, que protegen símbolos; las indicaciones geográficas, que protegen denominaciones de origen; los secretos industriales, que respaldan la ocultación de información, y las patentes, que otorgan monopolios temporales sobre invenciones a cambio de desvelarlas. Sin embargo, en muchas tradiciones legales, entre ellas la hispana, se distingue entre la *propiedad intelectual*, que se refiere exclusivamente a los derechos de autor, y la *propiedad industrial,* que abarca las figuras restantes.

En cualquier caso, la legislación que se aplica en todos estos aspectos es una de las más coordinadas en prácticamente todo el mundo. Por un lado, la OMPI (Organización Mundial de la Propiedad Intelectual, WIPO según sus siglas en inglés) promueve ambos tipos de propiedad en todos sus aspectos. Por otro, el acuerdo TRIPS (aspectos comerciales de la propiedad intelectual) establece unos mínimos de protección y obliga a todos los países miembros de la OMC (Organización Mundial del Comercio, WTO) a desarrollarlos en unos ciertos plazos, que dependen del nivel de desarrollo del país[3] .

(3)El acuerdo TRIPS fue firmado por la presión de los países industrializados (especialmente Estados Unidos y Japón).

La Declaración Universal de los Derechos Humanos reconoce, en su artículo 27, el derecho a que se protejan los intereses morales y materiales que correspondan a cualquier persona por razón de las producciones científicas, literarias o artísticas de que sean autores. Sin embargo, en muchos casos (y de forma habitual en el caso del software), este derecho suele ser transferido en la práctica a las empresas que emplean a los creadores o que comercializan sus creaciones. No obstante, la propiedad intelectual se justifica no sólo por razones morales, sino también por razones prácticas, para dar cumplimiento a otro derecho: el de la sociedad a beneficiarse de las creaciones, incentivándolas con beneficios y protegiendo las inversiones para la creación, la investigación y el desarrollo. Para armonizar ambos derechos, la propiedad intelectual es temporal, y caduca cuando ha cumplido su función de promoción.

Pero la caducidad no es la única característica que diferencia la propiedad intelectual de la ordinaria. Hoy en día, los objetos de la misma pueden copiarse fácil y económicamente, sin pérdida de calidad. La copia no perjudica a quien ya disfruta de lo copiado, al contrario del robo, que sí que priva del objeto al poseedor original. La copia sí que puede perjudicar al propietario, ya que lo priva potencialmente de los ingresos de una venta. El control de la copia de intangibles es mucho más complicado que el del robo de bienes tangibles y puede llevarnos a una sociedad policial, que necesite controlar todas las copias de información, y a una gran inseguridad jurídica, porque aumentan las posibilidades de violación accidental de derechos. Además la creatividad es incremental: al crear siempre se copia algo, y la línea divisoria entre la copia burda y la inspiración es sutil.

Para profundizar más en todo esto, en los siguientes apartados se repasan algunas de las categorías de la propiedad intelectual. En cualquier caso, se puede adelantar ya que el software libre propone un nuevo punto de equilibrio en este ámbito, primando los beneficios de la copia y la innovación incremental frente al control exclusivo de una obra por parte de su autor.

3.1.1. Derechos de autor

Los derechos de autor (*copyright*) protegen la expresión de un contenido, no el contenido en sí mismo. Se desarrollaron para recompensar a los autores de libros o de arte. Las obras protegidas pueden expresar ideas, conocimientos o métodos libremente utilizables, pero se prohíbe reproducirlas sin permiso, total o parcialmente, con o sin modificaciones. Esta protección es muy sencilla, ya que entra automáticamente en vigor con ámbito casi universal en el momento de publicación de la obra. Modernamente se ha extendido a los programas de ordenador y (en algunas áreas geográficas) a recopilaciones de datos.

La Ley de Propiedad Intelectual (LPI) en España, y leyes similares en otros países, desarrolladas sobre la base del Convenio de Berna para la protección de trabajos literarios y artísticos de 1886, regulan los derechos de autor. Estos derechos se dividen en derechos morales y derechos patrimoniales. Los primeros

garantizan al autor el control sobre la divulgación de su obra, con nombre o seudónimo, el reconocimiento de autoría, el respeto a la integridad de la obra y el derecho de modificación y retirada. Los segundos le dan derecho a explotarla económicamente y pueden ser cedidos total o parcialmente, de forma exclusiva o no, a un tercero. Los derechos morales son vitalicios o indefinidos, mientras que los patrimoniales tienen una duración bastante larga (setenta años después de la muerte del autor, si es una persona física, en el caso de la ley española).

La cesión de derechos se especifica mediante un contrato denominado *licencia*. En el caso de programas privativos, éstos generalmente se distribuyen por medio de licencias de uso "no exclusivo", que se entiende que se aceptan automáticamente al abrir o instalar el producto. No es necesario pues firmar el contrato, ya que, en el caso de no aceptarlo el receptor, rigen automáticamente los derechos por omisión de la ley, es decir, ninguno. Las licencias no pueden restringir algunos derechos que otorga la legislación vigente, como el de hacer copias privadas de arte o música, lo que permite regalar una copia de una grabación a un amigo, pero este derecho no es aplicable a los programas. Según la LPI de 1996 (Ley de Propiedad Intelectual. Real Decreto Legislativo 1/1996, de 12 de abril) [77], modificada en 2006 (Ley de Propiedad Intelectual. Ley 23/2006, de 7 de julio) [79], respecto de los programas siempre se puede hacer una copia de seguridad, se pueden estudiar para hacer programas interoperables y se pueden corregir y adaptar a nuestras necesidades (cosa difícil, porque no solemos disponer de los códigos fuente). Estos derechos no pueden ser restringidos por licencias, aunque las leyes están en proceso de revisión, en una tendencia aparentemente imparable a reducir los derechos de los usuarios. Las recopilaciones organizadas de obras o datos ajenos también están sometidas a derechos de autor, si bien los términos son distintos y la duración menor.

Las nuevas tecnologías de la información, y en especial la Red, han trastocado profundamente la protección de los derechos de autor, ya que las expresiones de contenidos son mucho más fáciles de copiar que los contenidos mismos. Y en el caso de los programas y algunas obras de arte (música, imágenes, películas, e incluso literatura), "funcionan" automáticamente en el ordenador, sin necesidad de un esfuerzo humano apreciable. En cambio, los diseños o inventos hay que construirlos, y posiblemente ponerlos en producción. Esta posibilidad de crear riqueza sin coste ha llevado a gran parte de la sociedad, en particular a los países pobres, a duplicar programas sin pagar licencia, sin que exista una conciencia social de que eso sea una "mala acción" (como sí que la suele haber con respecto al robo de bienes físicos, por ejemplo). Por otro lado, los fabricantes de programas, solos o en coalición (por ejemplo la BSA, Business Software Alliance), presionan fuertemente para que las licencias se paguen y los gobiernos persigan lo que se ha dado en llamar *piratería*.

Nota

La palabra *piratería* se ha popularizado como sinónimo de 'violación de cualquier forma de propiedad intelectual, especialmente en el caso de la copia ilegal de programas, música y películas'. El término parece exagerado, y en el diccionario de la Real Academia Española de la Lengua aparece como una acepción en sentido figurado, ya que el término original se refiere a 'robo con violencia en el mar'. Por ello Richard Stallman recomienda evitarlo ("Some confusing or loaded words and phrases that are worth avoiding", 2003) [212].

Precisamente para proteger los derechos de autor de aquellos contenidos con licencias privativas, nacen los llamados sistemas DRM (*digital rights management*, 'gestión de derechos digitales'), con el fin de controlar el acceso y la utilización de datos en soporte digital o de restringir su uso a ciertos dispositivos. El empleo de sistemas DRM se ha criticado fuertemente en muchos sectores, puesto que trata de proteger derechos de autor imponiendo restricciones más allá de las suficientes, por lo que algunos, como la Free Software Foundation, recomiendan interpretar las siglas como *digital restrictions management* ('gestión de restricciones digitales'), intentando evitar la utilización de la palabra *derechos* (en inglés, *rights*), al considerar que se privan excesivos derechos de los usuarios para lograr satisfacer los derechos de los autores.

3.1.2. Secreto comercial

Otro de los recursos que tienen las empresas para rentabilizar sus inversiones es el secreto comercial, protegido por las leyes de propiedad industrial, siempre que las empresas tomen las medidas suficientes para ocultar la información que no quieren desvelar. En el caso de productos químicos o farmacéuticos que requieran aprobación gubernamental, el Estado se compromete a no desvelar los datos entregados que no sea obligatorio hacer públicos.

Una de las aplicaciones más conocidas del secreto comercial se encuentra en la industria del software propietario, que generalmente comercializa programas compilados sin dar acceso al código fuente, para así impedir el desarrollo de programas derivados.

A primera vista parece que la protección del secreto comercial es perversa, ya que puede privar indefinidamente a la sociedad de conocimientos útiles. En cierto modo así lo entienden algunas legislaciones, permitiendo la ingeniería inversa para desarrollar productos sustitutos, aunque la presión de las industrias ha conseguido que en muchos países ésta sea una actividad prohibida y en otros sólo esté permitida en aras de la compatibilidad.

Sea perverso o no el secreto comercial, en muchos casos es mejor que una patente, ya que da una ventaja competitiva al que pone un producto en el mercado mientras la competencia trata de imitarlo con ingeniería inversa. Cuanto más sofisticado sea el producto, más costará a la competencia reproducirlo, mientras que si es trivial, lo copiará rápidamente. La imitación con mejoras

ha sido fundamental para el desarrollo de las que hoy son superpotencias (Estados Unidos y Japón) y es muy importante para la independencia económica de los países en vías de desarrollo.

3.1.3. Patentes y modelos de utilidad

La alternativa al secreto comercial es la patente. A cambio de un monopolio de diecisiete a veinticinco años y un determinado coste económico, un *invento* es revelado públicamente, de forma que sea fácilmente reproducible. Con ella se pretende promover la investigación privada, sin coste para el contribuyente y sin que el resultado se pierda. El poseedor de una patente puede decidir si permite a otros utilizarla y el precio que debe pagar por la licencia.

La doctrina oficial es que el sistema de patentes promueve la innovación, pero cada vez más se hacen oír voces que afirman que la dificulta, bien porque opinan que el sistema está mal implementado, o bien porque creen que es perverso en sí mismo (François-René Rideau, "Patents are an economic absurdity", 2000) [194].

Lo que se considera un invento ha ido variando con el tiempo, y existen grandes presiones para ampliar la cobertura del sistema, que incluyen algoritmos, programas, modelos de negocio, sustancias naturales, genes y formas de vida, incluidas plantas y animales. TRIPS exige que el sistema de patentes no discrimine ningún ámbito del saber. Las presiones de la Organización Mundial de la Propiedad Intelectual (OMPI o WIPO) pretenden eliminar la necesidad de que el invento tenga aplicación industrial, y también rebajar los estándares de inventiva exigibles en una patente. Estados Unidos está a la cabeza de los países con un mínimo de exigencias sobre lo que es patentable, y es además el más beligerante para que otros países adopten sus estándares, sin acordarse de que él mismo se negó a aceptar las patentes extranjeras cuando era un país subdesarrollado.

Una vez obtenida una patente, los derechos del poseedor son independientes de la calidad del invento y del esfuerzo invertido en obtenerlo. Dado el coste de mantenimiento de una patente y los costes de litigación, solamente las grandes empresas pueden mantener y mantienen una amplia cartera de patentes que las sitúan en una posición que les permite ahogar cualquier competencia. Dada la facilidad para colocar patentes sobre soluciones triviales o de gran aplicabilidad, pueden monopolizar para sí un espacio muy amplio de actividad económica.

Con patentes, muchas actividades, especialmente la programación, se hacen extremadamente arriesgadas, ya que es muy fácil que en el desarrollo de un programa complicado se viole accidentalmente alguna patente. Cuando dos o más empresas están investigando para resolver un problema, es muy probable que lleguen a una solución similar casi al mismo tiempo, pero sólo una (generalmente la que tenga más recursos) logrará patentar su invento, de manera

que las otras perderán toda posibilidad de rentabilizar su inversión. Todo desarrollo técnico complejo puede convertirse en una pesadilla si para cada una de las soluciones de sus partes es necesario investigar si la solución encontrada está patentada (o en trámite), para intentar obtener la licencia o para buscar una solución alternativa. Este problema es especialmente grave en el software libre, donde las violaciones de patentes de algoritmos son evidentes por simple inspección del código.

Aunque en Europa aún es ilegal patentar un algoritmo, se podrá hacer en muy breve plazo, quizá cuando el lector lea estas líneas.

3.1.4. Marcas y logotipos registrados

Las marcas y logotipos son nombres y símbolos que representan un acervo de calidad (o una gran inversión en publicidad). No tienen gran importancia dentro del software libre, posiblemente porque registrarlos tiene un coste. Así, solamente algunos nombres importantes, como Open Source (por Open Source Foundation), Debian (por Software in the Public Interest), GNOME (por GNOME Foundation), GNU (por Free Software Foundation) u OpenOffice.org (por SUN Microsystems) están registrados, y sólo en algunos países. Sin embargo, el no registro de nombres ha provocado problemas. Por ejemplo, en Estados Unidos (1996) y en Corea (1997) ha habido personas que han registrado el nombre Linux y han demandado dinero por su uso. La resolución de estas disputas supone costes legales y la necesidad de demostrar un uso del nombre anterior a la fecha del registro.

3.2. Licencias en el software libre

Legalmente hablando, la situación de los programas libres respecto de los privativos no es muy diferente: también se distribuyen bajo licencia. Lo que los diferencia es precisamente lo que permite esa licencia. En el caso de las licencias de programas libres, que no restringen precisamente el uso, la redistribución y la modificación, lo que pueden imponer son condiciones que hay que satisfacer precisamente en caso de que se quiera redistribuir el programa. Por ejemplo, pueden exigir que se respeten las indicaciones de autoría o que se incluya el código fuente si se quiere redistribuir el programa listo para ejecutar.

Aunque en esencia *software libre* y *software propietario* se diferencien en la licencia con la que los autores publican sus programas, es importante hacer hincapié en que esta diferencia se refleja en condiciones de uso y redistribución totalmente diferentes. Como se ha visto a lo largo de los últimos años, esto ha originado no sólo métodos de desarrollo totalmente diferentes, sino incluso formas prácticamente opuestas (en muchos sentidos) de entender la informática.

Las leyes sobre propiedad intelectual aseguran que en ausencia de permiso explícito no se puede hacer casi nada con una obra (en nuestro caso, un programa) que se reciba o se compre. Sólo el autor (o el que posea los derechos de la obra) nos puede dar ese permiso. En cualquier caso, la propiedad de la obra no cambia por otorgar una licencia, ya que ésta no supone transferencia de propiedad, sino solamente de derecho de uso y, en algunos casos (obligados en el software libre), de distribución y modificación. Las licencias de software libre se diferencian de las privativas precisamente en que en lugar de restringir cuidadosamente lo que se permite, otorgan ciertos permisos explícitos. Cuando uno recibe un programa libre puede redistribuirlo o no, pero si lo redistribuye, sólo puede hacerlo porque la licencia se lo permite. Pero para ello es preciso cumplir con la licencia. En definitiva, la licencia contiene las normas de uso a las que han de atenerse usuarios, distribuidores, integradores y otras partes implicadas en el mundo de la informática.

Para comprender plenamente todos los entresijos legales que se van a presentar en este capítulo (y que, sin duda, son muy importantes para entender la naturaleza del software libre), también es necesario saber que cada nueva versión de un programa es considerada como una nueva obra. El autor tiene, otra vez, plena potestad para hacer con ella lo que le apetezca, incluso distribuirla en términos y condiciones totalmente diferentes (o sea, con una licencia diferente a la anterior). Así, si el lector es autor único de un programa podrá publicar una versión bajo una licencia de software libre y, si le apeteciese, otra posterior bajo una licencia propietaria. En caso de que existan más autores y que la nueva versión contenga código cuya autoría les corresponda, si se va a publicar bajo otras condiciones, todos ellos habrán de dar el visto bueno al cambio de licencia.

Un tema todavía relativamente abierto es la licencia que se aplica a las contribuciones externas. Generalmente se supone que una persona que contribuya al proyecto acepta *de facto* que su contribución se ajuste a las condiciones especificadas por su licencia, aunque esto podría tener poco fundamento jurídico. La iniciativa de la Free Software Foundation de pedir mediante carta (física) la cesión de todos los derechos de *copyright* a cualquiera que contribuya con más de diez líneas de código a un subproyecto de GNU es una buena muestra de que en el mundo del software libre hay políticas más estrictas con respecto a estas contribuciones.

Partiendo de todo lo dicho, vamos a centrarnos ya en el resto de este capítulo en el análisis de diversas licencias. Para poner en contexto este estudio, hay que recordar que de ahora en adelante, cuando decimos que una licencia es de software libre, lo decimos en el sentido de que cumple las definiciones de *software libre* presentadas en el apartado 1.1.1.

3.2.1. Tipos de licencias

La variedad de licencias libres es grande, aunque por razones prácticas la mayoría de los proyectos utilizan un pequeño conjunto de cuatro o cinco. Por un lado, muchos proyectos no quieren o no pueden dedicar recursos a diseñar una licencia propia; por otro, la mayoría de los usuarios prefieren referirse a una licencia ampliamente conocida que leerse y analizar licencias completas.

Bibliografía

Se pueden ver recopiladas y comentadas tanto licencias consideradas libres como licencias consideradas no libres o libres pero incompatibles con la GPL desde el punto de vista de la FSF en Free Software Foundation, "Licencias libres" [121]. El punto de vista filosóficamente diferente de la Open Source Initiative se refleja en su listado (Open Source Initiative, "Licencias de fuente abierta") [181]. Pueden verse discrepancias en algunas licencias, como la Apple Public Source License Ver. 1.2, considerada no libre por la FSF por la obligación de publicar todos los cambios (aunque sean privados), de notificar a Apple las redistribuciones, o por la posibilidad de revocación unilateral. No obstante, la presión de esta clasificación hizo que Apple publicara la versión 2.0 en agosto de 2003, ya considerada libre por la FSF.

Es posible dividir las licencias de software libre en dos grandes familias. La primera está compuesta por las licencias que no imponen condiciones especiales en la *segunda redistribución* (esto es, que sólo especifican que el software se puede redistribuir o modificar, pero que no imponen condiciones especiales si se hace, lo que permite, por ejemplo, que alguien que reciba el programa pueda después redistribuirlo como software propietario): son las que llamaremos *licencias permisivas*. La segunda familia, que denominaremos *licencias robustas* (o *licencias* copyleft), incluye las que, al estilo de la GNU GPL, imponen condiciones en caso de que se quiera redistribuir el software, condiciones que van en la línea de forzar a que se sigan cumpliendo las condiciones de la licencia después de la *primera redistribución*. Mientras que el primer grupo hace énfasis en la libertad de quien recibe un programa, que le permite hacer casi todo lo que quiera con él (en términos de condiciones de futuras redistribuciones), el segundo hace énfasis en la libertad de cualquiera que potencialmente pueda recibir algún día un trabajo derivado del programa, que obliga a que las sucesivas modificaciones y redistribuciones respeten los términos de la licencia original.

Nota

El término *copyleft* aplicado a una licencia, usado sobre todo por la Free Software Foundation para definir las suyas, tiene implicaciones similares a las de la expresión *licencia robusta* tal como la usamos en este texto.

La diferencia entre estos dos tipos de licencias ha sido (y es) tema de debate en la comunidad del software libre. En cualquier caso, es conveniente recordar que todas ellas son licencias libres.

3.2.2. Licencias permisivas

Las licencias permisivas, a veces también llamadas *licencias liberales* o *minimalistas*, no imponen prácticamente ninguna condición sobre quien recibe el software, y sin embargo, le dan permiso de uso, redistribución y modificación. Desde cierto punto de vista, este enfoque puede entenderse como la garantía de las máximas libertades para quien recibe un programa. Pero desde otro, puede entenderse también como la máxima despreocupación con respecto al

hecho de que una vez recibido un programa por parte de alguien, se sigan garantizando las mismas libertades cuando ese programa se redistribuya. De hecho, típicamente estas licencias permiten que se redistribuya con licencia privativa un software cuyo autor distribuye con licencia permisiva.

Entre estas licencias, una de las más conocidas es la licencia BSD, hasta tal punto que muchas veces se llama a las licencias permisivas *licencias de tipo BSD*. La licencia BSD (Berkeley Software Distribution) tiene su origen en la publicación de versiones de Unix realizadas por la universidad californiana de Berkeley, en EE.UU. La única obligación que exige es dar crédito a los autores, mientras que permite tanto la redistribución binaria como la de los códigos fuente, aunque no obliga a ninguna de las dos en ningún caso. Asimismo, da permiso para realizar modificaciones y ser integrada con otros programas casi sin restricciones.

Nota

Una de las consecuencias prácticas de las licencias de tipo BSD ha sido la difusión de estándares, ya que los desarrolladores no encuentran ningún obstáculo para realizar programas compatibles con una implementación de referencia bajo este tipo de licencias. De hecho, ésta es una de las razones de la extraordinaria y rápida difusión de los protocolos de Internet y de la interfaz de programación basada en zócalos (*sockets*), porque la mayoría de los desarrolladores comerciales derivó su realización de la de la Universidad de Berkeley.

Las licencias permisivas son bastante populares, y existe toda una familia con características similares a la BSD: X Window, Tcl/Tk, Apache, etc. Históricamente estas licencias aparecieron debido a que el software correspondiente fue creado en universidades con proyectos de investigación financiados por el Gobierno de los Estados Unidos. Estas universidades prescindían de la comercialización de estos programas, asumiendo que ya habían sido pagados previamente por el Gobierno, y por tanto, con los impuestos de todos los contribuyentes, por lo que cualquier empresa o particular podía utilizar el software casi sin restricciones.

Como ya se ha comentado, a partir de un programa distribuido bajo una licencia permisiva puede crearse otro (en realidad, una nueva versión) que sea privativo. Los críticos de las licencias BSD ven en esta característica un peligro, ya que no se garantiza la libertad de versiones futuras de los programas. Sus partidarios, por el contrario, la consideran la máxima expresión de la libertad y argumentan que, a fin de cuentas, se puede hacer (casi) todo lo que se quiera con el software.

La mayoría de las licencias permisivas son una copia calcada de la original de Berkeley en la que se modifica todo lo referente a la autoría. En algunos casos, como la licencia del proyecto Apache, incluyen alguna cláusula adicional, como la imposibilidad de llamar a las versiones redistribuidas de igual manera

que a la original. Todas estas licencias suelen incluir, como la BSD, la prohibición de usar el nombre del propietario de los derechos para promocionar productos derivados.

Asimismo, todas las licencias, sean de tipo BSD o no, incluyen una *limitación de garantía* que es en realidad una *negación de garantía*, necesaria para evitar demandas legales por garantías implícitas. Aunque se ha criticado mucho esta negación de garantía en el software libre, es práctica habitual en el software propietario, que generalmente sólo garantiza que el soporte es correcto y que el programa en cuestión se ejecuta.

Esquema resumen de la licencia BSD

Copyright © *el propietario*. Todos los derechos reservados.

Se permite la redistribución en fuente y en binario, con o sin modificación, siempre que se cumplan las condiciones siguientes:

1) Las redistribuciones en fuente deben retener la nota de *copyright* y listar estas condiciones y la limitación de garantía.

2) Las redistribuciones en binario deben reproducir la nota de *copyright* y listar estas condiciones y la limitación de garantía en la documentación.

3) Ni el nombre del *propietario* ni el de los que han contribuido pueden usarse sin permiso para promocionar productos derivados de este programa.

Este programa se proporciona "tal cual", sin garantías expresas ni implícitas, tales como su aplicabilidad comercial o su adecuación para un propósito determinado. En ningún caso *el propietario* será responsable de ningún daño causado por su uso (incluida la pérdida de datos, la pérdida de beneficios o la interrupción de negocio).

A continuación describimos brevemente algunas licencias permisivas:

- Licencia de X Window, versión 11 (X11)
  (http://www.x.org/Downloads_terms.html) [73].
  Es la licencia usada para la distribución del sistema X Window, el sistema de ventanas más ampliamente utilizado en el mundo Unix, y también en entornos GNU/Linux. Es muy similar a la licencia BSD, que permite redistribución, uso y modificación prácticamente sin restricciones. A veces se la llama *licencia MIT* (con peligrosa poca precisión, porque el MIT ha usado otros tipos de licencias). Bajo esta licencia se distribuyen también trabajos derivados de X Windows, como XFree86.

- Zope Public License 2.0 (http://www.zope.org/Resources/ZPL) [76].
  Esta licencia (habitualmente llamada ZPL) se usa para la distribución de Zope (un servidor de aplicaciones) y otros productos relacionados. Es similar a la BSD, con el interesante detalle de que prohíbe expresamente el uso de marcas registradas por Zope Corporation.

- Licencia de Apache.
  Es una licencia bajo la que se distribuyen la mayor parte de los programas producidos por el proyecto Apache. Es similar a la licencia BSD.

Hay algunos programas libres que no se distribuyen con una licencia específica, sino que su autor los declara explícitamente *public domain* ('de dominio público o del común'). La principal consecuencia de esta declaración es que el autor renuncia a todos sus derechos sobre el programa, que por lo tanto, se puede modificar, redistribuir, usar, etc. de cualquier manera. A efectos prácticos, es muy similar a que el programa esté bajo una licencia de tipo BSD.

3.2.3. Licencias robustas

La Licencia Pública General de GNU (GNU GPL)

La Licencia Pública General del proyecto GNU (Free Software Foundation, 1991) [118] (más conocida por su acrónimo en inglés, GPL), que mostramos traducida en el apéndice C, es con diferencia la más popular y conocida de todas las del mundo del software libre. Su autoría corresponde a la Free Software Foundation (promotora del proyecto GNU), y en un principio fue creada para ser la licencia de todo el software generado por la FSF. Sin embargo, su utilización ha ido más allá hasta convertirse en la licencia más utilizada (por ejemplo, más del 70% de los proyectos anunciados en Freshmeat están licenciados bajo la GPL), incluso por proyectos bandera del mundo del software libre, como el núcleo Linux.

La licencia GPL es interesante desde el punto de vista legal porque hace un uso muy creativo de la legislación de *copyright*, consiguiendo efectos prácticamente contrarios a los que se suponen de la aplicación de esta legislación: en lugar de limitar los derechos de los usuarios, los garantiza. Por este motivo, en muchos casos se denomina a esta "maniobra" *copyleft* (juego de palabras en inglés que se puede traducir como 'izquierdos de autor'). Alguien con una pizca de humor llegó incluso a lanzar el eslogan "copyleft, all rights reversed".

En líneas básicas, la licencia GPL permite la redistribución binaria y la del código fuente, aunque en el caso de que redistribuya de manera binaria obliga a que también se pueda acceder a los códigos fuente. Asimismo, está permitido realizar modificaciones sin restricciones. Sin embargo, sólo se puede redistribuir código licenciado bajo GPL de forma integrada con otro código (por ejemplo, mediante enlaces o *links*) si éste tiene una licencia compatible. Esto se ha llamado *efecto viral* (aunque muchos consideran despectiva esta denominación) de la GPL, ya que un código publicado una vez con esas condiciones nunca puede cambiarlas.

Nota

Una licencia es incompatible con la GPL cuando restringe alguno de los derechos que la GPL garantiza, bien explícitamente, contradiciendo alguna cláusula, o bien implícitamente, imponiendo alguna nueva. Por ejemplo, la licencia BSD actual es compatible, pero la de Apache, que exige que se mencione explícitamente en los materiales de publicidad que el trabajo combinado contiene código de todos y cada uno de los titulares de derechos, es incompatible. Esto no implica que no se puedan usar simultáneamente programas con ambas licencias, o incluso integrarlos. Sólo supone que esos programas integrados no se pueden distribuir, pues es imposible cumplir simultáneamente las condiciones de redistribución de ambos.

La licencia GPL está pensada para asegurar la libertad del código en todo momento, ya que un programa publicado y licenciado bajo sus condiciones nunca podrá ser privativo. Es más, ni ese programa ni modificaciones del mismo pueden ser publicados con una licencia diferente de la propia GPL. Como ya se ha dicho, los partidarios de las licencias de tipo BSD ven en esta cláusula un recorte de la libertad, mientras que sus seguidores creen que es una forma de asegurarse que ese software siempre va a ser libre. Por otro lado, se puede considerar que la licencia GPL maximiza las libertades de los usuarios, mientras que las de tipo BSD maximizan las de los desarrolladores. Hay que observar, sin embargo, que en el segundo caso estamos hablando de los desarrolladores en general y no de los autores, ya que muchos autores consideran que la licencia GPL es más beneficiosa para sus intereses, puesto que obliga a sus competidores a publicar sus modificaciones (mejoras, correcciones, etc.) en caso de que redistribuyan su software, mientras que con una licencia de tipo BSD éste no tiene por qué ser el caso.

En cuanto a la naturaleza contraria al *copyright* de esta licencia, se debe a que su filosofía (y la de la Free Software Foundation) es que el software no debe tener propietarios (Richard Stallman, "Why software should not have owners", 1998) [207]. Aunque es cierto que el software licenciado con la GPL tiene un autor, que es el que a fin de cuentas permite la aplicación de la legislación de *copyright* sobre este software, las condiciones bajo las que lo publica le confieren tal carácter que podemos considerar que la propiedad del software corresponde a quien lo tiene y no a quien lo ha creado.

Por supuesto, esta licencia también incluye *negaciones de garantía* para proteger a los autores. Asimismo, y para preservar la buena fama de los autores originales, toda modificación de un fichero fuente debe incluir una nota en la que se especifique la fecha y el autor de la modificación.

La GPL tiene en cuenta también las patentes de software, y exige que si el código lleva algoritmos patentados (como hemos dicho, algo legal y usual en Estados Unidos y práctica irregular en Europa), o se concede licencia de uso de la patente libre de tasas, o no se puede distribuir bajo la GPL.

La última versión de la licencia GPL, la segunda, se publicó en 1991 (aunque en el momento de escribir este texto está en avanzado proceso de preparación la tercera). Precisamente teniendo en cuenta futuras versiones, la licencia recomienda licenciar bajo las condiciones de la segunda o de cualquier otra

posterior publicada por la Free Software Foundation, cosa que hacen muchos autores. Sin embargo, otros, entre los que destaca Linus Torvalds (creador de Linux), publican su software sólo bajo las condiciones de la segunda versión de la GPL, buscando desmarcarse de las posibles evoluciones futuras de la Free Software Foundation.

La tercera versión de la GPL (http://gplv3.fsf.org) [115] pretende llevarla al escenario actual del software, principalmente en aspectos como patentes, sistemas DRM (*digital rights management*, 'gestión de derechos digitales') y otras limitaciones de la libertad del software. Por ejemplo, en el borrador disponible en el momento de escribir este texto (mayo de 2007), no permite que un fabricante de hardware bloquee la utilización de ciertos módulos de software si no presentan una firma digital que acredite una determinada autoría. Un ejemplo de estas prácticas se da en los grabadores digitales de vídeo TiVo, que proporcionan el código fuente de todo su software (licenciado con GPLv2) al tiempo que no permiten que se utilicen modificaciones del código en dicho hardware[4].

(4)Este caso ha llegado incluso a sugerir la denominación de *tivoisation* para varios otros similares que han surgido.

La licencia tampoco permite que el software obligue a la ejecución en entorno prefijado, como ocurre cuando se prohíbe la utilización de núcleos no firmados en distribuciones cuya política de seguridad lo considere oportuno.

Nota

Hay varios puntos en la licencia GPLv3 que han despertado una cierta oposición. Uno de los grupos de opositores está compuesto por desarrolladores del núcleo Linux (entre ellos el propio Linus Torvalds). Consideran que el requisito de utilización de componentes de software firmados permite otorgar ciertas características de seguridad imposibles de otra manera, al tiempo que su prohibición explícita extendería la licencia al terreno del hardware. Además, la limitación establecida por el mecanismo de firmas se daría únicamente en las plataformas de hardware y software así diseñadas, de manera que se podría modificar el software para su utilización en otro hardware. Con respecto a este punto, la FSF está a favor del empleo de mecanismos de firmas que recomienden la no utilización de componentes no firmados por motivos de seguridad, pero cree que la no prohibición de aquellos mecanismos de firmas que imposibilitan la utilización de componentes no firmados podrían dar lugar a escenarios en los que no existiesen plataformas de hardware o software en las que ejecutar dichas modificaciones del software, por lo que en ese caso quedarían totalmente limitadas las libertades del software libre en lo que a modificación del código se refiere.

La Licencia Pública General Menor de GNU (GNU LGPL)

La Licencia Pública General Menor del proyecto GNU (Free Software Foundation, GNU Lesser General Public License, versión 2.1, febrero 1999) [119], comúnmente conocida por sus iniciales en inglés, LGPL, es la otra licencia de la Free Software Foundation. Pensada al principio para su uso en bibliotecas (la L, en sus inicios, venía de *library*, 'biblioteca'), fue modificada recientemente para ser considerada la hermana menor (*lesser*, 'menor') de la GPL.

La LGPL permite el uso de programas libres con software propietario. El programa en sí se redistribuye como si estuviera bajo la licencia GPL, pero se permite su integración con cualquier otro paquete de software sin prácticamente limitaciones.

Como se puede ver, en un principio esta licencia estaba orientada a las bibliotecas, para que se pudiera potenciar su uso y desarrollo sin tener los problemas de integración que implica la GPL. Sin embargo, cuando se vio que el efecto buscado de popularizar las bibliotecas libres no se veía compensado por la generación de programas libres, la Free Software Foundation decidió el cambio de *library* a *lesser* y desaconsejó su uso, salvo para condiciones muy puntuales y especiales. Hoy en día, existen muchos programas que no son bibliotecas licenciados bajo las condiciones de la LGPL. Por ejemplo, el navegador Mozilla o el paquete ofimático (*suite*) OpenOffice.org están licenciados, entre otros, también bajo la LGPL.

Nota

Igual que pasa con la GPL, la última versión publicada de la LGPL es la segunda, aunque ya hay un borrador de la tercera versión (http://gplv3.fsf.org/pipermail/info-gplv3/2006-July/000008.html) [116]. Esta nueva versión es más corta que la anterior, dado que refiere todo su texto a la GPLv3 y destaca únicamente sus diferencias.

Otras licencias robustas

Otras licencias robustas que puede resultar interesante comentar son las siguientes:

- Licencia de Sleepycat (www.sleepycat.com/download/oslicense.html) [59].
  Es la licencia bajo la que la empresa Sleepycat (http://www.sleepycat.com/) [60] distribuye sus programas (entre los que destaca el conocido Berkeley DB). Obliga a ciertas condiciones siempre que se redistribuya el programa o trabajos derivados del mismo. En particular, obliga a ofrecer el código fuente (incluidas las modificaciones si se trata de un trabajo derivado) y a que la redistribución imponga al receptor las mismas condiciones. Aunque mucho más corta que la GNU GPL, es muy similar a ella en sus efectos principales.

- eCos License 2.0 (http://www.gnu.org/licenses/ecos-license.html) [25].
  Es la licencia bajo la que se distribuye eCos (http://sources.redhat.com/ecos/) [24], un sistema operativo en tiempo real. Es una modificación de la GNU GPL que no considera que el código que se enlace con programas protegidos por ella queden sujetos a las cláusulas de la GNU GPL si se redistribuyen. Desde este punto de vista, sus efectos son similares a los de la GNU LGPL.

- Affero General Public License (http://www.affero.org/oagpl.html) [78].

Es una interesante modificación de la GNU GPL que considera el caso de los programas que ofrecen servicios vía web, o en general, vía redes de ordenadores. Este tipo de programas plantean un problema desde el punto de vista de las licencias robustas. Como el uso del programa no implica haberlo recibido mediante una redistribución, aunque esté licenciado, por ejemplo, bajo la GNU GPL, alguien puede modificarlo y ofrecer un servicio en la Red usándolo, sin redistribuirlo de ninguna forma, y por tanto, sin estar obligado, por ejemplo, a distribuir su código fuente. La Affero GPL tiene una cláusula que obliga a que, si el programa tiene un medio para proporcionar su código fuente vía web a quien lo use, no se pueda desactivar esa característica. Esto significa que si el autor original incluye esa capacidad en el código fuente, cualquier usuario puede obtenerlo, y además esa *redistribución* está sometida a las condiciones de la licencia. La Free Software Foundation se plantea incluir provisiones similares en la versión 3 de su GNU GPL.

- IBM Public License 1.0 (http://oss.software.ibm.com/developerworks/opensource/license10.html) [40].
  Es una licencia que permite la redistribución binaria de trabajos derivados sólo si (entre otras condiciones) se prevé algún mecanismo para que quien reciba el programa pueda recibir su código fuente. La redistribución del código fuente se ha de hacer bajo la misma licencia. Además, esta licencia es interesante porque obliga al que redistribuye el programa con modificaciones a licenciar automática y gratuitamente las patentes que puedan afectar a esas modificaciones, y que sean propiedad del redistribuidor, a quien reciba el programa.

- Mozilla Public License 1.1 (http://www.mozilla.org/MPL/MPL-1.1.html) [49].
  Se trata de un ejemplo de licencia libre con origen en una empresa. Es una evolución de la primera licencia libre que tuvo Netscape Navigator, que en su momento fue muy importante por ser la primera vez que una empresa muy conocida decidía distribuir un programa bajo su propia licencia libre.

3.2.4. Distribución bajo varias licencias

Hasta ahora se ha dado por supuesto que cada programa se distribuye bajo una única licencia en la que se especifican las condiciones de uso y redistribución. Sin embargo, un autor puede distribuir obras con distintas licencias. Para entenderlo, debemos tener en cuenta que cada publicación es una nueva obra, y que se puede dar el caso de que se distribuyan versiones que sólo difieran en la licencia. Como veremos, la mayoría de las veces esto se traduce en el hecho de que en función de lo que el usuario quiera hacer con el software se encontrará con que tiene obedecer una licencia u otra.

Uno de los ejemplos más conocidos de doble licencia es el de la biblioteca Qt, sobre la que se cimenta el entorno de escritorio KDE. Trolltech, una empresa afincada en Noruega, distribuía Qt con una licencia propietaria, aunque eximía del pago a los programas que hicieran uso de la misma sin ánimo de lucro. Por esta causa y por sus características técnicas, fue elegida a mediados de la década de los noventa por el proyecto KDE. Esto supuso una ardua polémica con la Free Software Foundation, ya que KDE dejaba de ser entonces software libre en su conjunto, al depender de una biblioteca propietaria. Tras un largo debate (durante el cual apareció GNOME como competidor libre de KDE en el escritorio), Trolltech decidió utilizar el sistema de doble licencia para su producto estrella: los programas bajo la GPL podían hacer uso de una versión de Qt GPL, mientras que si se querían integrar con programas con licencias incompatibles con la GPL (por ejemplo, licencias privativas), debían comprarles una licencia especial. Esta solución satisfizo a todas las partes, y hoy en día KDE se considera software libre.

Otros ejemplos conocidos de licencia dual son StarOffice y OpenOffice.org, o Netscape Communicator y Mozilla. En ambos casos el primer producto es propietario, mientras que el segundo es una versión libre (generalmente bajo las condiciones de varias licencias libres). Aunque en un principio los proyectos libres eran versiones limitadas de sus hermanos propietarios, con el tiempo han ido tomando su propio camino, por lo que a día de hoy tienen un grado de independencia bastante grande.

3.2.5. Documentación de programas

La documentación que viene con un programa es parte integrante del mismo, igual que los comentarios del código fuente, como reconoce, por ejemplo en España, la Ley de Propiedad Intelectual. Dado este nivel de integración, parece lógico que a la documentación se le apliquen las mismas libertades y que evolucione de la misma manera que el programa: toda modificación que se haga de un programa requiere un cambio simultáneo y consistente en su documentación.

La mayor parte de esta documentación suele estar codificada como ficheros de texto sin formato, ya que se pretende que sea universalmente accesible con un entorno de herramientas mínimo, y (en el caso de los programas libres) suele incluir una pequeña introducción al programa (`README` o `LEEME`), instrucciones de instalación (`INSTALL`), alguna historia sobre la evolución pasada y el futuro del programa (`CHANGELOG` y `TODO`), autoría y condiciones de copia (`AUTHORS` y `COPYRIGHT` o `COPYING`), así como las instrucciones de uso. Todos ellos, menos la autoría y las condiciones de copia, deberían ser libremente modificables a medida que el programa evoluciona. A la autoría sólo se le deberían añadir nombres y créditos, pero sin borrar nada, y las condiciones de copia sólo deberían modificarse si las mismas lo permiten.

Las instrucciones de uso acostumbran a estar codificadas en formatos más complejos, ya que suelen ser documentos más largos y más ricos. El software libre exige que esta documentación pueda ser modificada fácilmente, lo que a su vez obliga a usar formatos denominados *transparentes*, de especificación conocida y procesables por herramientas libres, como son, además del texto puro y limpio, los formatos de páginas de manual de Unix, TexInfo, LaTeX o DocBook, sin perjuicio de distribuir también el resultado de la transformación de esos documentos fuente en formatos más aptos para su visualización o impresión, como HTML, PDF o RTF (formatos en general más *opacos*).

Sin embargo, muchas veces se hace documentación sobre programas por parte de terceros que no han intervenido en el desarrollo. A veces es documentación de carácter didáctico que facilita la instalación y el uso de un programa concreto (HOWTO, CÓMO o recetarios); a veces es documentación más amplia, que abarca varios programas y su integración, que compara soluciones, etc., bien en forma de tutorial o bien en forma de referencia; a veces es una mera recopilación de preguntas frecuentes con sus respuestas (FAQ o PUF). Es un ejemplo notable el Proyecto de documentación Linux (http://www.tldp.org) [44]. En esta categoría podemos también incluir otros documentos técnicos, no necesariamente sobre programas, ya sean las instrucciones para cablear una red local, para construir una cocina solar, para reparar un motor o para seleccionar un proveedor de tornillos.

Estos documentos son algo intermedio entre la mera documentación de programas y los artículos o libros muy técnicos y prácticos. Sin menoscabo de la libertad de lectura, copia, modificación y redistribución, el autor puede querer verter opiniones que no desea que se tergiversen, o al menos puede querer que esas tergiversaciones no se le atribuyan; o puede querer que se conserven párrafos, como agradecimientos; o que necesariamente se modifiquen otros, como el título. Aunque estas inquietudes pueden también manifestarse con los programas en sí mismos, no se han expresado con tanta fuerza en el mundo del software libre como en el de la documentación libre.

3.3. Resumen

En este capítulo se han revisado los aspectos legales que rigen o que tienen impacto sobre el software libre. Éstos forman parte del derecho de propiedad intelectual o industrial y han sido concebidos, en principio, para estimular la creatividad recompensando a los creadores por un tiempo determinado. De todos ellos, el llamado *copyright* es el que más afecta al software libre, y convenientemente empleado, sirve para asegurar su existencia en forma de licencias libres.

Hemos podido ver la importancia que tienen las licencias dentro del mundo del software libre. Asimismo, hemos presentado la gran variedad de licencias existentes, su motivación, sus repercusiones y sus ventajas e inconvenientes. En definitiva, podemos decir que la GPL trata de maximizar las libertades que

tiene el usuario del software, tanto si lo recibe directamente de su autor como si no, mientras que las licencias de tipo BSD lo que hacen es maximizar las libertades del modificador o redistribuidor.

A la vista de lo que se ha comentado en este capítulo, se deduce que es muy importante decidir pronto qué licencia va a tener un proyecto y conocer detalladamente sus ventajas e inconvenientes, ya que una modificación posterior suele ser muy difícil, sobre todo si el número de contribuciones externas es muy grande.

Para finalizar, queremos hacer hincapié en el hecho de que el software libre y el software propietario se diferencien de manera estricta única y exclusivamente en la licencia con la que se publican los programas. En próximos capítulos veremos, sin embargo, que esta puntualización meramente legal puede tener consecuencias –o no– en la manera como se desarrolla el software, dando lugar a un nuevo modelo de desarrollo que se diferencie en mayor o menor medida, según el caso, de los métodos de desarrollo "tradicionales" utilizados en la industria del software.

4. El desarrollador y sus motivaciones

"Being a hacker is lots of fun, but it's a kind of fun that takes lots of effort. The effort takes motivation. Successful athletes get their motivation from a kind of physical delight in making their bodies perform, in pushing themselves past their own physical limits. Similarly, to be a hacker you have to get a basic thrill from solving problems, sharpening your skills and exercising your intelligence."

"Ser un *hacker* es muy divertido, pero es un tipo de diversión que supone mucho esfuerzo. El esfuerzo supone motivación. Los deportistas de éxito obtienen su motivación de un tipo de placer físico en hacer que sus cuerpos funcionen, en llevarse a sí mismos más allá de sus propios límites físicos. De forma similar, para ser un *hacker* tienes que experimentar una emoción básica al resolver problemas, al afilar tus aptitudes y al ejercitar tu inteligencia."

Eric Steven Raymond, "How to become a hacker"

4.1. Introducción

El desarrollo parcialmente anónimo y distribuido del software libre ha permitido que durante muchos años los recursos humanos con los que cuenta sean desconocidos. Consecuencia de este desconocimiento ha sido la mitificación, al menos parcial, del mundo del software libre y de la vida de los que están detrás de él, que se ampara en tópicos más o menos extendidos sobre la cultura *hacker* y los ordenadores. Desde hace unos pocos años, se ha venido realizando un gran esfuerzo por parte de la comunidad científica para conocer mejor a las personas que participan en proyectos de software libre, su procedencia, sus motivaciones, su preparación y otros aspectos que pudieran parecer interesantes. Desde el punto de vista puramente pragmático, conocer quién se implica y por qué en este tipo de proyectos puede ser de gran utilidad para la generación de software libre. Algunos científicos, principalmente economistas, psicólogos y sociólogos, han querido ir más allá y han visto en esta comunidad el germen de futuras comunidades virtuales con reglas y jerarquías propias, en muchos casos totalmente diferentes de las que conocemos en la sociedad "tradicional". Entre las incógnitas más importantes está la de conocer los motivos que llevan a los desarrolladores a ser partícipes en una comunidad de estas características, habida cuenta de que los beneficios económicos, al menos los directos, son prácticamente inexistentes, mientras que los indirectos son difícilmente cuantificables.

4.2. ¿Quiénes son los desarrolladores?

Este apartado pretende dar una visión global de las personas que dedican su tiempo y su esfuerzo a participar en proyectos de software libre. Los datos que se van a mostrar provienen en su mayoría de estudios científicos realizados en los últimos años, entre los cuales los más significativos –aunque, por supuesto,

no los únicos– han sido *Free/libre and open source software. Survey and study*, parte IV: "Survey of developers", 2002 [126], y "Who is doing it? Knowing more about libre software developers", 2001 [197].

Los desarrolladores de software libre son generalmente personas jóvenes. La media de edad está situada en torno a los veintisiete años. La varianza de edad es muy grande, ya que el grupo predominante se encuentra en una horquilla que va de los veintiuno a los veinticuatro años, y la moda –el valor que aparece con mayor frecuencia– se sitúa en los veintitrés años. Es interesante observar cómo la edad de incorporación al movimiento de software libre tiene sus máximos entre los dieciocho y los veinticinco años –y es especialmente pronunciada entre los veintiuno y los veintitrés–, lo que equivaldría a la edad universitaria. Esta evidencia contrasta con la afirmación de que el software libre es cosa principalmente de adolescentes, aunque la presencia de éstos es evidente (alrededor de un 20% de los desarrolladores tiene menos de veinte años). En definitiva, podemos ver que los desarrolladores suelen ser mayoritariamente veinteañeros, en un 60%, mientras que los menores de veinte años y los mayores de treinta se reparten a partes iguales el 40% restante.

De la edad de incorporación se puede deducir que existe una gran influencia universitaria en el software libre. Ello no es de extrañar, ya que como se ha podido ver en el capítulo de historia, el software libre –antes incluso de denominarse así– ha estado íntimamente ligado a las instituciones educativas superiores. Aún hoy, el verdadero motor del uso y la expansión del software libre siguen siendo las universidades y los grupos de usuarios estudiantiles. Por tanto, no es de extrañar que más de un 70% de los desarrolladores cuente con una preparación universitaria. El dato tiene mayor importancia si tenemos en cuenta que del 30% restante muchos no son universitarios porque todavía están en su fase escolar. Aun así, también tienen cabida –y no por ello son menos apreciados– desarrolladores que no han accedido nunca a estudios superiores, pero que son amantes de la informática.

El desarrollador de software libre es generalmente varón. Las cifras que se manejan en las diferentes encuestas sobre la presencia de mujeres en la comunidad varían entre un 1% y un 3%, y compiten con el propio margen de error de las mismas. Por otro lado, una mayoría (60%) afirma tener pareja, mientras que el número de desarrolladores con hijos sólo es de un 16%. Dados los márgenes de edades en los que están comprendidos los desarrolladores de software libre, estos datos concuerdan bastante bien con una muestra aleatoria, por lo que se pueden considerar "normales". El mito del desarrollador solitario cuya afición por la informática es lo único en su vida se muestra, como se puede ver, como una excepción a la regla.

4.3. ¿Qué hacen los desarrolladores?

Profesionalmente, los desarrolladores de software libre se definen como ingenieros de software (33%), estudiantes (21%), programadores (11%), consultores (10%), profesores de universidad (7%), etc. En el lado contrario, podemos ver que no suelen integrar ni los departamentos comerciales ni de marketing (alrededor de un 1%). Es interesante observar cómo muchos de ellos se definen a sí mismos como ingenieros de software antes que programadores –casi tres veces más–, teniendo en cuenta, como se verá en el capítulo dedicado a la ingeniería del software, que la aplicación de las técnicas clásicas de ingeniería del software (e incluso algunas modernas) no suele estar muy arraigada en el mundo del software libre.

El vínculo universitario, que ya ha sido mostrado con anterioridad, vuelve a aparecer en este apartado. Alrededor de uno de cada tres desarrolladores es estudiante o profesor de universidad, lo que viene a demostrar que existe una gran colaboración entre gente proveniente principalmente de la industria del software (los dos tercios restantes) y el ámbito académico.

Por otro lado, también se ha podido constatar una gran interdisciplinariedad: uno de cada cinco desarrolladores proviene de campos diferentes del de las tecnologías de la información. Esto, unido al hecho de que existe también un número similar de desarrolladores no universitarios, refleja la existencia de una gran riqueza en cuanto a intereses, procedencias y, en definitiva, composición de los equipos de desarrollo. Es muy difícil encontrar una industria moderna, si es que existe, donde el grado de heterogeneidad sea tan grande como el que se puede ver en el software libre.

Además del aproximadamente 20% de estudiantes, los desarrolladores suelen ser en su gran mayoría asalariados, en un 64%, mientras que el porcentaje de autónomos es del 14%. Finalmente, sólo un 3% dice encontrarse en paro, dato significativo, puesto que la encuesta fue hecha tras el comienzo de la crisis de las puntocom.

Nota

El hecho de que la financiación del software libre, al contrario de lo que pasa con el software propietario, no se pueda hacer mediante la venta de licencias ha propiciado desde siempre calientes debates en torno a cómo deben ganarse la vida los programadores. En las encuestas que se están comentando en este capítulo más de un 50% de los desarrolladores decían haberse beneficiado económicamente de manera directa o indirecta de su implicación en el software libre. Sin embargo, hay muchos que no lo ven tan claro. El propio Richard Stallman, fundador del proyecto GNU, ante la pregunta de qué es lo que debe hacer un desarrollador de software libre para ganar dinero, suele responder que puede trabajar como camarero.

4.4. Distribución geográfica

La obtención de datos geográficos de los desarrolladores es una cuestión que todavía ha de ser abordada de manera más científica. El problema que presentan los estudios cuyos resultados se muestran en este capítulo es que, al tra-

tarse de encuestas en Internet abiertas a todo aquél que quisiera participar, la participación dependía mucho de los sitios donde se hubieran anunciado, así como de la forma en la que se hubieran anunciado. Para ser estrictos, cabe mencionar que las encuestas no buscaban representatividad en este sentido, sino más bien obtener la respuesta y/o la opinión del mayor número posible de desarrolladores de software libre.

Sin embargo, podemos aventurarnos a hacer unas cuantas afirmaciones al respecto, a sabiendas de que los datos no son tan fiables como los expuestos con anterioridad y de que el margen de error es, por tanto, mucho más grande. Lo que parece un hecho constatable es que la gran mayoría de los desarrolladores de software libre provienen de países industrializados, y que es escasa la presencia de desarrolladores de países del llamado Tercer Mundo. No es de extrañar, por consiguiente, que el mapa de desarrolladores del proyecto Debian (http://www.debian.org/devel/developers.loc) [187], por poner un ejemplo, concuerde con las fotografías de la Tierra de noche: allí donde hay luz –léase "donde hay civilización industrializada"– es donde suelen concentrarse en mayor medida los desarrolladores de software libre. Esto que en un principio podría parecer lógico, contrasta con las posibilidades potenciales que el software libre ofrece para países del Tercer Mundo.

Un claro ejemplo lo podemos encontrar en la tabla siguiente, que contiene los países de origen más frecuentes para los desarrolladores del proyecto Debian a lo largo de los últimos cuatro años. Se puede observar una tendencia a la descentralización del proyecto, algo que se constata en el hecho de que el crecimiento de los desarrolladores en Estados Unidos –el país que más aporta– es inferior a la media. Y es que, por lo general, los países han conseguido doblar el número de voluntarios en los últimos cuatro años, y Francia, que ha conseguido multiplicar por cinco su presencia, es el ejemplo más claro en este sentido. Considerando que los primeros pasos de Debian tuvieron lugar en el continente americano (en particular en los Estados Unidos y el Canadá), podemos ver que en los últimos cuatro años el proyecto ha sufrido una *europeización*. Suponemos que el siguiente paso será la ansiada mundialización, con la incorporación de países sudamericanos, africanos y asiáticos (exceptuando Corea y Japón, ya bien representados), aunque los datos que manejamos (dos desarrolladores en Egipto, China o la India, y uno en México, Turquía o Colombia en junio de 2003) no son muy halagüeños en este sentido.

Dentro del mundo del software libre (y no sólo en el caso de Debian), existe una amplia discusión sobre la supremacía entre Europa y Estados Unidos. Casi todos los estudios que se han venido realizando muestran que la presencia de desarrolladores europeos es ligeramente superior a la norteamericana, efecto que queda mitigado por el hecho de que la población europea es mayor que la norteamericana. Nos encontramos entonces ante una situación de guerra de cifras, ya que el número de desarrolladores per cápita favorece a los nortea-

mericanos, pero vuelve a ser favorable a los europeos si tenemos en cuenta, en vez de las cifras de población absolutas, solamente a aquellas personas que disponen de acceso a Internet.

En cuanto a países, las zonas con mayor implantación (en número de desarrolladores dividido por población) son las del norte de Europa (Finlandia, Suecia, Noruega, Dinamarca e Islandia) y las centroeuropeas (Benelux, Alemania y Chequia), seguidas de Australia, Canadá, Nueva Zelanda y Estados Unidos. La zona mediterránea, a pesar de que es importante en magnitudes absolutas (debido a la gran población que tienen Francia, Italia y España), se encuentra, sin embargo, por debajo de la media.

Tabla 1. Países con mayor número de desarrolladores de Debian

País	01/07/1999	01/07/2000	01/07/2001	01/07/2002	20/06/2003
Estados Unidos	162	169	256	278	297
Alemania	54	58	101	121	136
Reino Unido	34	34	55	63	75
Australia	23	26	41	49	52
Francia	11	11	24	44	51
Canadá	20	22	41	47	49
España	10	11	25	31	34
Japón	15	15	27	33	33
Italia	9	9	22	26	31
Países Bajos	14	14	27	29	29
Suecia	13	13	20	24	27

4.5. Dedicación

El número de horas que los desarrolladores de software libre dedican al software libre es uno de los aspectos más desconocidos. Hay que señalar, además, que ésta es una de las grandes diferencias con el software generado por una empresa, donde tanto el equipo como la dedicación de cada miembro del equipo al desarrollo son conocidos. El tiempo que los desarrolladores de software libre dedican se puede tomar como una medida indirecta del grado de profesionalización. Antes de mostrar los datos de los que se dispone en la actualidad, es importante hacer notar que éstos han sido obtenidos de las estimaciones que los propios desarrolladores han dado en varias encuestas, por lo que a la inexactitud inherente a este tipo de recolección de datos se ha de añadir un margen de error debido principalmente a lo que cada desarrollador entienda como tiempo de desarrollo. De esta forma, es seguro que muchos desarrolla-

dores no cuenten el tiempo que dedican a leer el correo (o quizás sí) y que indiquen sólo el que dedican a programar y a depurar. Por eso, todas las cifras que se presentan a continuación han de tomarse con el debido cuidado.

Los estudios que se han realizado hasta ahora muestran que cada desarrollador de software libre dedica de media alrededor de once horas semanales ("Motivation of software developers in open source projects: an internet-based survey of contributors to the Linux kernel", 2003) [143]. Sin embargo, esta cifra puede llevar rápidamente al engaño, ya que existe una gran varianza en la dedicación de los desarrolladores de software. En el estudio *Free/libre and open source software. Survey and study*, parte IV: "Survey of developers", 2002 [126], un 22,5% de los encuestados indicó que su aportación era inferior a las dos horas semanales, cifra que subía al 26,5% para los que dedicaban entre dos y cinco horas semanales; entre seis y diez horas es el tiempo que dedica un 21,0%, mientras que el 14,1% lo hace entre once y veinte horas semanales; el 9,2% afirmaba que el tiempo que dedicaban a desarrollar software libre era de entre veinte y cuarenta horas semanales, y el 7,1%, más de cuarenta horas semanales.

Tabla 2. Dedicación en horas semanales

Horas semanales	Porcentaje
Menos de 2 horas	22,5%
Entre 2 y 5 horas	26,1%
Entre 5 y 10 horas	21,0%
Entre 10 y 20 horas	14,1%
Entre 20 y 40 horas	9,2%
Más de 40 horas	7,1%

Nota

Además de poder ver la profesionalización de los equipos de desarrollo de software libre, la dedicación en horas es un parámetro de gran importancia a la hora de realizar estimaciones de coste y hacer comparaciones con los modelos de desarrollo propietarios que se siguen en la industria. En el software libre, por ahora, sólo contamos con productos finales (nuevas entregas del software, sincronización de código nuevo en los sistemas de versiones...) que no nos permiten conocer cuánto tiempo ha necesitado el desarrollador para conseguirlos.

El análisis de estas cifras nos muestra que alrededor de un 80% de los desarrolladores realizan estas tareas en su tiempo libre, mientras que sólo uno de cada cinco podría considerarse que dedica tanto tiempo a esta actividad como un profesional. Más adelante, en el capítulo de ingeniería del software, podremos ver cómo este dato concuerda con la contribución de los desarrolladores, ya que ambos parecen seguir la ley de Pareto (*vid.* apartado 7.6).

4.6. Motivaciones

Se ha especulado y se sigue especulando mucho sobre las motivaciones que hay detrás del desarrollo de software libre, sobre todo en su vertiente de actividad de tiempo libre (que, como hemos visto, corresponde a cerca de un 80% de los desarrolladores). Como en los apartados anteriores, sólo contamos con los datos de las encuestas, por lo que es importante entender que se trata de lo que los desarrolladores responden, que puede ser más o menos coherente con la realidad. Los porcentajes que se van a mostrar a continuación superan en suma el 100% porque se daba la posibilidad a los encuestados de elegir varias respuestas.

En cualquier caso, de sus respuestas parece desprenderse que la mayoría quiere aprender y desarrollar nuevas habilidades (cerca de un 80%) y que muchos lo hacen para compartir conocimientos y habilidades (50%) o para participar en una nueva forma de cooperación (alrededor de un tercio). El primer dato no parece nada sorprendente, habida cuenta de que un profesional con mayores conocimientos se encuentra más cotizado que uno que no los posee. El segundo dato, sin embargo, no es tan fácil de explicar, e incluso parece ir en contra de la afirmación de Nikolai Bezroukov en "A second look at the cathedral and the bazaar" (diciembre, 1998) [91] que viene a decir que los líderes de los proyectos de software libre tienen buen cuidado de no compartir toda la información que poseen para perpetuar su poder. Mientras tanto, la tercera opción más frecuente es, sin lugar a dudas, fiel reflejo de que los propios desarrolladores se muestran entusiasmados por la forma en la que generalmente se crea el software libre; es difícil encontrar una industria en la que un grupo de voluntarios levemente organizados pueda plantar cara tecnológicamente a las grandes compañías del sector.

Aunque la teoría "clásica" para explicar los motivos por los que los desarrolladores de software libre se dedican a hacer aportaciones a este tipo de proyectos gira en torno a la reputación y a beneficios económicos indirectos a medio y largo plazo, parece que los propios desarrolladores no están de acuerdo con estas afirmaciones. Sólo un 5% de los encuestados responde que desarrolla software libre para ganar dinero, mientras que el número de ellos que lo hacen por obtener reputación asciende a un 9%, lejos de las respuestas que se han presentado en el párrafo anterior. En cualquier caso, parece que el estudio de las motivaciones que tienen los desarrolladores para entrar a formar parte de la comunidad del software libre es una de las tareas primordiales con las que se habrán de enfrentar sociólogos y psicólogos en los próximos tiempos.

4.7. Liderazgo

Reputación y liderazgo son dos características con las que se ha tratado de explicar el éxito del software libre, y en especial, el del modelo de bazar, tal como se verá en el capítulo dedicado a la ingeniería del software. Como hemos podido ver en otro capítulo, el dedicado a las licencias del software, existen

ciertas diferencias entre las licencias de software libre y sus homólogas en el campo de la documentación. Esas diferencias radican en la forma en la que se preservan la autoría y la opinión del autor –más acentuada en textos que en programas.

En *Free/libre and open source software. Survey and study*, parte IV: "Survey of developers" (2002) [126] se incluyó una pregunta en la que se instaba a los desarrolladores a indicar qué personas de una lista dada les eran conocidas, aunque no fuera necesariamente de manera personal. Los resultados, que se exponen en la tabla 3, muestran que estas personas se pueden aglutinar en tres grupos claramente diferenciados:

Tabla 3. Grado de conocimiento de desarrolladores importantes

Desarrollador	Conocido por
Linus Torvalds	96,5%
Richard Stallman	93,3%
Miguel de Icaza	82,1%
Eric Raymond	81,1%
Bruce Perens	57,7%
Jamie Zawinski	35,8%
Mathias Ettrich	34,2%
Jörg Schilling	21,5%
Marco Pesenti Gritti	5,7%
Bryan Andrews	5,6%
Guenter Bartsch	3,5%
Arpad Gereoffy	3,3%
Martin Hoffstede	2,9%
Angelo Roulini	2,6%
Sal Valliger	1,2%

- Un primer grupo de gente con claras connotaciones filosoficohistóricas dentro del mundo del software libre (aunque, como se puede ver, cuenten también con notables aptitudes técnicas):

1) Linus Torvalds. Creador del núcleo Linux, el sistema operativo libre más utilizado, y coautor de *Just for fun: the story of an accidental revolutionary* [217].

2) Richard Stallman. Ideólogo y fundador de la Free Software Foundation y desarrollador en varios proyectos GNU. Autor de varios escritos muy importantes dentro del mundo del software libre ("Why *free software* is better than *open source*", 1998 [206], "Copyleft: pragmatic idealism", 1998 [205], "The GNU Project" [208] y "The GNU Manifesto", 1985 [117]).

3) Miguel de Icaza. Cofundador del proyecto GNOME y de Ximian Inc., y desarrollador de parte de GNOME y de MONO.

4) Eric Raymond. Impulsor de la Open Source Initiative, autor de "La catedral y el bazar" [192] y desarrollador principal de Fetchmail.

5) Bruce Perens. Antiguo líder del proyecto Debian, impulsor (converso) de la Open Source Initiative y desarrollador de la herramienta e-fence.

6) Jamie Zawinsky. Ex desarrollador de Mozilla y famoso por una carta de 1999 en la que dejaba el proyecto Mozilla argumentando que el modelo utilizado no iba a dar frutos nunca ("Resignation and postmortem", 1999) [237].

7) Mathias Ettrich. Fundador de KDE y desarrollador de LyX y otros.

- Un segundo grupo formado por desarrolladores. Para esta encuesta se tomaron los nombres de los desarrolladores principales de los seis proyectos más populares en el índice de aplicaciones de software libre FreshMeat. Se puede ver que (a excepción de Linus Torvalds, por motivos obvios, y de Jörg Schilling) el grado de conocimiento de estos desarrolladores es pequeño:

1) Jörg Schilling, creador de cdrecord, entre otras aplicaciones.

2) Marco Pesenti Gritti, desarrollador principal de Galeon.

3) Bryan Andrews, desarrollador de Apache Toolbox.

4) Guenther Bartsch, creador de Xine.

5) Arpad Gereoffy, desarrollador de MPEGPlayer.

- Un tercer grupo compuesto por los nombres de las tres últimas "personas" de la tabla. Estos nombres fueron inventados por el equipo de la encuesta para poder comprobar el margen de error de las respuestas.

De los resultados se desprenden dos cosas: la primera es que el margen de error de las respuestas se puede considerar pequeño (menor de un 3%), y la segunda es que la mayoría de los desarrolladores de las aplicaciones de software libre

más populares son tan conocidos como personas que no existen. Este dato puede dar que pensar a los que aducen como una de las primeras causas para desarrollar software libre el hecho de buscar la fama.

4.8. Resumen y conclusiones

Este capítulo ha pretendido arrojar un poco de luz sobre el ampliamente desconocido tema de la gente que dedica su tiempo al software libre. En líneas generales se puede afirmar que el desarrollador de software libre es un varón joven con estudios universitarios (o en vías de conseguirlos). La relación del mundo del software libre con la universidad (estudiantes y profesores) es muy estrecha, aunque sigue predominando el desarrollador que no tiene nada que ver con el ámbito académico.

En cuanto a la dedicación en número de horas, se ha mostrado cómo existe una gran desigualdad al estilo de la postulada en la ley de Pareto. Las motivaciones de los desarrolladores –según ellos mismos–, lejos de ser monetarias y egocéntricas, tal como suelen sugerir economistas y psicólogos, están más bien centradas en compartir y aprender. Finalmente, se ha expuesto una tabla de los personajes del mundo del software libre más significantes (que incluía a otros que no lo eran tanto, como hemos podido ver) y se ha demostrado que la reputación en la gran comunidad del software libre suele depender de más razones que la simple codificación de una aplicación libre exitosa.

5. Economía

"Res publica non dominetur."

"Las cosas públicas no tienen dueño." (traducción libre)

Aparecido en un anuncio de IBM sobre Linux (2003)

En este capítulo se tratan algunos aspectos económicos relacionados con el software libre. Empezaremos mostrando cómo se financian los proyectos de software libre (cuando efectivamente se financian, ya que en muchos casos se desarrollan únicamente con trabajo y recursos aportados voluntariamente). A continuación expondremos los principales modelos de negocio que están poniendo en práctica las empresas relacionadas directamente con el software libre. El capítulo termina con un pequeño estudio sobre la relación entre el software libre y los monopolios en la industria del software.

5.1. Financiación de proyectos de software libre

El software libre se desarrolla de muchas formas distintas y con mecanismos para conseguir recursos que varían muchísimo de un caso a otro. Cada proyecto libre tiene su propia forma de financiarse, desde el que está formado completamente por desarrolladores voluntarios y utiliza solamente recursos cedidos de manera altruista, hasta el que es llevado a cabo por una empresa que factura el 100% de sus costes a una entidad interesada en el desarrollo correspondiente.

En este apartado nos vamos a centrar en los proyectos en los que hay financiación externa y no todo el trabajo realizado es voluntario. En estos casos, hay algún tipo de flujo de capital con origen externo al proyecto que se encarga de aportar recursos para su desarrollo. Así, el software libre construido puede considerarse, de alguna forma, como un producto de esta financiación externa. Por ello es común que sea esa fuente externa la que decida (al menos parcialmente) cómo y en qué se gastan los recursos.

En cierto sentido, esta financiación externa para proyectos libres puede considerarse como un tipo de patrocinio, aunque este patrocino no tiene por qué ser desinteresado (y habitualmente no lo es). En los siguientes apartados comentaremos los tipos de financiación externa más habituales. Mientras el lector se dedica a ellos, conviene, no obstante, no olvidar que ésta es sólo una de maneras como los proyectos que construyen software libre consiguen recursos. Pero hay otras, y entre ellas la más importante es (como se ha visto en el capítulo 4) el trabajo de muchos desarrolladores voluntarios.

5.1.1. Financiación pública

Un tipo muy especial de financiación de proyectos libres es la pública. La entidad financiadora puede ser directamente un gobierno (local, regional, nacional o incluso supranacional) o una institución pública (por ejemplo, una fundación). En estos casos, la financiación suele ser similar a la de los proyectos de investigación y desarrollo, y de hecho es muy habitual que provenga de entidades públicas promotoras de I+D. Normalmente, la entidad financiadora no busca recuperar la inversión (o al menos no de forma directa), aunque suele tener objetivos claros (como favorecer la creación de tejido industrial e investigador, promover cierta tecnología o cierto tipo de aplicaciones, etc.).

En la mayor parte de estos casos, no se encuentra explícitamente la financiación de productos o servicios relacionados con software libre, sino que ésta suele ser el subproducto de un contrato con otros fines más generales. Por ejemplo, la Comisión Europea, dentro de sus programas de investigación, financia proyectos orientados a mejorar la competitividad europea en determinadas áreas. Algunos de estos proyectos tienen como parte de sus objetivos usar, mejorar y crear software libre en su ámbito de investigación (como herramienta para la investigación o como producto derivado de ella).

Las motivaciones para este tipo de financiación son muy variadas, pero se pueden destacar las siguientes:

1) Científica. Éste es el caso más habitual en proyectos de investigación financiados con fondos públicos. Aunque su objetivo no es producir software sino investigar en un determinado campo (relacionado con la informática o no), es muy posible que para ello sea preciso desarrollar programas que se usen como herramientas necesarias para alcanzar las metas del proyecto. Normalmente el proyecto no está interesado en comercializar estas herramientas, o incluso está activamente interesado en que otros grupos las utilicen y las mejoren. En estos casos, es bastante habitual distribuirlas como software libre. De esta manera, los recursos conseguidos por el grupo que realiza la investigación se han dedicado en parte a la producción de este software, por lo que se puede decir que ha sido desarrollado con financiación pública.

2) De promoción de estándares. Tener una implementación de referencia es una de las mejores formas de promover un estándar. En muchos casos eso supone tener programas que formen parte de esa implementación (o si el estándar se refiere al campo del software, que sean la implementación ellos mismos). Para que la implementación de referencia sea de utilidad en la promoción del estándar, es preciso que esté disponible, al menos para comprobar la interoperabilidad para todos los que quieran desarrollar productos que se acojan a ese estándar. Y en muchos casos es conveniente también que los fabricantes puedan adaptar directamente la implementación de referencia para usarla en sus productos si así lo desean. De esta

manera se desarrollaron, por ejemplo, muchos de los protocolos de Internet que hoy se han convertido en norma universal. En estos casos, la liberación de esas implementaciones de referencia como software libre puede ayudar mucho en esa promoción. De nuevo, también aquí el software libre es un subproducto, en este caso de la promoción del estándar. Y habitualmente quien se encarga de esta promoción es una entidad pública (aunque a veces es un consorcio privado).

3) Social. El software libre es una herramienta de gran interés en la creación de la infraestructura básica para la sociedad de la información. Las entidades interesadas en utilizar software libre para ayudar al acceso universal a esta sociedad de la información pueden financiar proyectos relacionados con él (normalmente proyectos de desarrollo de nuevas aplicaciones o de adaptación de otras ya existentes).

Nota

Un ejemplo de financiación pública con finalidad fundamentalmente social es el caso de LinEx, promovido por la Junta de Extremadura (Extremadura, España) para promover la sociedad de la información fundamentalmente en lo que a alfabetización informática se refiere. La Junta ha financiado el desarrollo de una distribución basada en Debian para conseguir este objetivo. Otro caso similar es el de la financiación por parte del Gobierno alemán de desarrollos de GnuPG orientados a facilitar su uso para los usuarios no experimentados, con la idea de favorecer la utilización del correo seguro entre sus ciudadanos.

El desarrollo de GNAT

Un caso notorio de financiación pública para el desarrollo de software libre fue el del compilador GNAT. GNAT, compilador de Ada, fue financiado por el proyecto Ada 9X del Departamento de Defensa de EE.UU. con la idea de disponer de un compilador de la nueva versión del lenguaje de programación Ada (la que luego sería Ada 95), cuyo uso trataba de promover en aquella época. Una de las causas que se habían identificado en cuanto a la adopción de la primera versión de Ada (Ada 83) por las empresas de software había sido la tardía disposición de un compilador del lenguaje, y su alto precio cuando estuvo finalmente disponible. Por ello, trataron de que no ocurriese lo mismo con Ada 95, asegurándose de que hubiera un compilador de manera prácticamente simultánea a la publicación del nuevo estándar del lenguaje.

Para lograrlo, Ada 9X contrató un proyecto con un equipo de la Universidad de Nueva York (NYU) por un importe aproximado de un millón de USD para la realización de una "prueba de concepto" de compilador de Ada 95. Con esos fondos, y aprovechando la existencia de GCC (el compilador de C de GNU, del que se aprovechó la mayor parte del dorsal), el equipo de la NYU construyó efectivamente el primer compilador de Ada 95, que liberó bajo la GNU GPL. El compilador tuvo tanto éxito que al acabar el proyecto parte de sus constructores fundaron una empresa (Ada Core Technologies), que desde entonces se ha convertido en líder en el mercado de compiladores y herramientas de ayuda a la construcción de programas en Ada.

En este proyecto es notable observar la combinación de elementos de investigación (de hecho, este proyecto avanzó en el conocimiento sobre la construcción de frontales y sistemas de tiempo de ejecución para compiladores de lenguajes de tipo Ada) y de promoción de estándares (que era el objetivo más claro de su financiador).

5.1.2. Financiación privada sin ánimo de lucro

Éste es un tipo de financiación, con muchas características similares a las del caso anterior, que realizan normalmente fundaciones u organizaciones no gubernamentales. La motivación directa en estos casos suele ser producir software libre para su uso en algún ámbito que la entidad financiadora considere

especialmente relevante, pero también puede encontrarse la motivación indirecta de contribuir a resolver un problema (por ejemplo, una fundación dedicada a promover la investigación sobre una enfermedad puede financiar la construcción de un programa estadístico que ayude al análisis de grupos de experimentación en los que se estudia esa enfermedad).

En general, tanto los motivos para realizar este tipo de financiación como sus mecanismos son muy similares a los de la financiación pública, aunque naturalmente, están siempre marcados por los objetivos de la entidad financiadora.

Nota

Probablemente, el caso paradigmático de fundación que promueve el desarrollo de software libre sea la Free Software Foundation (FSF). Desde mediados de la década de 1980 esta fundación se dedica a la promoción del proyecto GNU y a fomentar en general el desarrollo del software libre.

Otro caso interesante, aunque en otro ámbito bastante diferente, es la Open Bioinformatics Foundation. Entre los fines de esta fundación se encuentra el de promover el desarrollo de los programas informáticos básicos para la investigación en cualquiera de las ramas de la bioinformática. Y en general, realiza esta promoción financiando y ayudando a la construcción de programas libres.

5.1.3. Financiación por parte de quien necesita mejoras

Otro tipo de financiación para el desarrollo de software libre, ya no tan altruista, es el que tiene lugar cuando alguien necesita mejoras en un producto libre. Por ejemplo, para uso interno, una empresa puede necesitar, en un programa dado, cierta funcionalidad o que ciertos errores sean corregidos. En estos casos, lo habitual es que la empresa en cuestión contrate el desarrollo que necesita. Este desarrollo, muy habitualmente (bien porque lo impone la licencia del programa modificado, bien porque la empresa lo decide así), es software libre.

El caso de Corel y Wine

A finales de la década de 1990, Corel decidió portar sus productos a GNU/Linux. En este proceso, descubrió que un programa libre diseñado para facilitar la ejecución de binarios para Windows en entornos Linux podría permitirle muchos ahorros de desarrollo. Pero para ello era preciso mejorarlo, fundamentalmente añadiéndole la emulación de cierta funcionalidad de Windows que usaban los programas de Corel.

Para ello, Corel contrató a Macadamian, que contribuyó con sus mejoras al proyecto Wine. Así, tanto Corel como Wine salieron beneficiados.

5.1.4. Financiación con beneficios relacionados

En este tipo de financiación, lo que busca la entidad financiadora es conseguir beneficios en productos relacionados con el programa a cuyo desarrollo aporta recursos. Normalmente, en estos casos los beneficios que obtiene la empresa financiadora no son exclusivos, ya que otras pueden entrar también en el mercado de la venta de productos relacionados, pero o bien la cuota de mercado que tiene es suficiente como para que no le preocupe mucho repartir la tarta con otros, o bien tiene alguna ventaja competitiva clara.

Algunos ejemplos de productos relacionados con un software dado son los siguientes:

- Libros. La empresa en cuestión vende manuales, guías de uso, textos para cursos, etc., relacionados con el programa libre que ayuda a financiar. Por supuesto, otras empresas pueden vender también libros relacionados, pero normalmente financiar el proyecto le dará acceso antes que a sus competidores a desarrolladores clave, o simplemente le facilitará una buena imagen de cara a la comunidad de usuarios del programa en cuestión.

- Hardware. Si una empresa financia el desarrollo de sistemas libres para cierto tipo de hardware, puede dedicarse a vender con más facilidad ese tipo de hardware. De nuevo, como el software desarrollado es libre, pueden aparecer competidores que vendan aparatos del mismo tipo, usando esos desarrollos pero sin colaborar en su financiación. Pero incluso así, la empresa en cuestión tiene varias ventajas sobre sus competidores, y una de ellas puede ser que su posición como aportadora de recursos para el proyecto le permita influir para conseguir que los desarrollos que se realicen con prioridad sean los que más le interesen.

- CD con programas. Probablemente, el modelo más conocido de este tipo es el de las empresas que financian ciertos desarrollos que luego aplican a su distribución de software. Por ejemplo, tener un buen entorno de escritorio puede ayudar mucho a vender CD con una cierta distribución de GNU/Linux, y por tanto, financiar su desarrollo puede ser un buen negocio para quien los vende.

Hay que tener en cuenta que para estar en este apartado la financiación en cuestión ha de hacerse con ánimo de lucro, y para ello la entidad financiadora ha de percibir algún beneficio posible en esta financiación. En los casos reales, sin embargo, es habitual que haya siempre una combinación de ánimo de lucro y altruismo cuando una empresa aporta recursos para que se realice un programa libre del cual espera beneficiarse indirectamente.

Nota

Un caso muy conocido de aportación de recursos a un proyecto, si bien de forma relativamente indirecta, es la ayuda que la editorial O'Reilly presta al desarrollo de Perl. Naturalmente, no es casualidad que esa editorial sea también una de las principales editoras de temas relacionados con Perl. En cualquier caso, es obvio que O'Reilly no tiene la exclusiva de la edición de libros de ese tipo, y otras editoriales compiten en ese segmento de mercado, con diverso éxito.

VA Software (en sus comienzos VA Research y más tarde VA Linux) ha colaborado activamente en el desarrollo del núcleo de Linux. Con ello ha conseguido, entre otras cosas, asegurar su continuidad, lo que era especialmente crítico para ella, de cara a sus clientes cuando su principal negocio era vender equipos con GNU/Linux preinstalado.

Red Hat ha financiado el desarrollo de muchos componentes de GNOME, con lo que ha conseguido fundamentalmente tener un entorno de escritorio para su distribución, cosa que ha contribuido a aumentar sus ventas. Como en otros casos, otros fabricantes de distribuciones se han beneficiado de este desarrollo, aunque muchos de ellos no hayan colaborado con el proyecto GNOME en la misma medida que Red Hat (y no son pocos los que no han colaborado en absoluto). A pesar de ello, Red Hat se beneficia de su contribución a GNOME.

5.1.5. Financiación como inversión interna

Hay empresas que, como parte de su modelo de negocio, desarrollan directamente software libre. Por ejemplo, una empresa puede decidir iniciar un nuevo proyecto libre en un ámbito donde perciba que puede haber oportunidades de negocio con la idea de rentabilizar posteriormente esa inversión. Este modelo podría considerarse una variante del anterior (financiación indirecta), y los "beneficios relacionados" serían las ventajas que obtuviera la empresa de la producción del programa libre. Pero al ser en este caso el producto libre en sí mismo el que se espera que produzca los beneficios, parece conveniente abrir una clasificación específica.

Este tipo de financiación da lugar a varios modelos de negocio. Cuando se analicen éstos (apartado 5.2) se explicarán también las ventajas que normalmente obtiene una empresa de esta inversión en un proyecto y qué métodos suelen utilizarse para rentabilizarla. Pero en cualquier caso, hay que destacar que a veces puede que el software en cuestión se desarrolle simplemente para satisfacer las necesidades de la propia empresa, y que sólo después se decida liberar, y quizás abrir, una línea de negocio relacionada con él.

Nota

Digital Creations (hoy Zope Corporation) es uno de los casos más conocidos de empresa que se dedica al desarrollo de software libre con la esperanza de rentabilizar su inversión. El proyecto libre en que más está invirtiendo es Zope, un servidor de aplicaciones que está teniendo un cierto éxito. Su historia con el software libre comenzó cuando la entonces Digital Creations buscaba capital riesgo para desarrollar su servidor de aplicaciones propietario, hacia 1998. Uno de los grupos interesados en invertir en ellos (Opticality Ventures) les puso como condición que el producto resultante debía ser libre, porque en caso contrario no veían cómo iba a poder conseguir una cuota de mercado significativa. Digital Creations se decidió por ese camino, y pocos meses después anunciaba la primera versión de Zope (unos años después cambió su nombre). Hoy en día Zope Corporation está especializada en ofrecer servicios de consultoría, formación y soporte para sistemas de gestión de contenidos basados en Zope, y otros productos en los que sin duda Zope es la piedra angular.

Ximian (antes Helix Code) es un caso bien conocido de desarrollo de aplicaciones libres en el entorno empresarial. Muy ligada desde sus orígenes al proyecto GNOME, Ximian ha producido sistemas de software como Evolution (un gestor de información personal

que tiene una funcionalidad relativamente similar a la ofrecida por Microsoft Outlook), Red Carpet (un sistema fácil de usar para la gestión de paquetería en un sistema operativo) y MONO (una implementación de gran parte de .NET). La empresa fue fundada en octubre de 1999 y atrajo a muchos desarrolladores de GNOME, que pasaron a formar parte de su grupo de desarrolladores (en muchos casos continuando su colaboración con el proyecto GNOME). Ximian se posicionó como una empresa de ingeniería experta en adaptaciones de GNOME, en la construcción de aplicaciones basadas en GNOME, y en general, en proporcionar servicios de desarrollo basados en software libre, especialmente de herramientas relacionadas con el entorno de escritorio. En agosto de 2003, Ximian fue adquirida por Novell.

Cisco Enterprise Print System (CEPS) (http://ceps.sourceforge.net/) [17] es un sistema de gestión de impresión para organizaciones con gran cantidad de impresoras. Fue desarrollado internamente en Cisco para satisfacer sus propias necesidades y liberado en el año 2000 bajo la GNU GPL. Es difícil saber con seguridad los motivos que tuvo Cisco para hacer esto, pero es posible que tuvieran que ver con la búsqueda de contribuciones externas (informes de error, nuevos controladores, parches, etc.). En cualquier caso lo que está claro es que, como Cisco no tenía ningún plan para comercializar el producto y no se sabía muy bien cuál era su mercado potencial, no tenía mucho que perder con esta decisión.

5.1.6. Otros modos de financiación

Hay otros modos de financiación difíciles de clasificar entre los anteriores. A modo de ejemplo, pueden destacarse los siguientes:

- Utilización de mercados para poner en contacto a desarrolladores y clientes. La idea que sostiene este modo de financiación es que, sobre todo para pequeños desarrollos, es difícil que un cliente que desee un desarrollo concreto pueda entrar en contacto con un desarrollador capaz de acometerlo de forma eficiente. Para mejorar esta situación, se postulan los mercados de desarrollo de software libre, donde los desarrolladores publicarían sus habilidades y los clientes los desarrollos que precisan. Un desarrollador y un cliente se ponen de acuerdo; tenemos una situación similar a la ya descrita como "financiación por parte de quien necesita mejoras" (apartado 5.1.3).

  SourceXchange

  SourceXchange fue un ejemplo de mercado que ponía en contacto a desarrolladores con sus clientes potenciales. Para ofrecer un proyecto, un cliente escribía una RFP (*request for proposal*, 'petición de propuesta') en la que especificaba el desarrollo que necesitaba y los recursos que estaba dispuesto a proporcionar para ese desarrollo. Estas RFP se publicaban en el sitio. Cuando un desarrollador veía una que le interesaba, hacía una oferta para ella. Si un desarrollador y un cliente se ponían de acuerdo en los términos del desarrollo, comenzaba un proyecto. Normalmente cada proyecto estaba supervisado por un *peer reviewer*, un revisor que se encargaba de asegurarse de que el desarrollador cumplía las especificaciones y de que efectivamente éstas tenían sentido, que aconsejaba sobre cómo llevar adelante el proyecto, etc. SourceXchange (propiedad de la empresa CollabNet) se encargaba de ofrecer el sitio, de garantizar la competencia de los revisores, de asegurarse del pago en caso de que los proyectos se completasen y de ofrecer herramientas para su seguimiento (servicios que facturaba al cliente). El primer proyecto mediado por SourceXchange acabó en marzo de 2000, pero poco más de un año después, en abril de 2001, el sitio cerró.

- Venta de bonos para financiar un proyecto. Esta idea de financiación es similar a la de los mercados de bonos a los que acuden las empresas, pero orientada al desarrollo de software libre. Tiene unas cuantas variantes, pero una de las más conocidas funciona como sigue. Cuando un desarrollador (individual o empresa) tiene idea de un nuevo programa o de una mejora a uno existente, lo escribe en forma de especificación, estima el

coste que tendría su desarrollo y emite bonos para su construcción. Estos bonos tienen un valor que se ejecuta sólo si el proyecto se termina finalmente. Cuando el desarrollador ha vendido suficientes bonos, comienza el desarrollo, que va financiando con préstamos basados en ellos. Cuando acaba el desarrollo, y se certifica por alguna tercera parte independiente que efectivamente lo realizado cumple las especificaciones, el desarrollador "ejecuta" los bonos que había vendido, paga sus deudas, y lo que le queda son los beneficios que obtiene por el desarrollo.

¿Quién estaría interesado en adquirir los mencionados bonos? Obviamente, los usuarios que desearan que ese nuevo programa o esa mejora a uno ya existente se realizaran. De alguna manera, este sistema de bonos permitiría que las partes interesadas fijaran (siquiera parcialmente) las prioridades de los desarrolladores mediante la compra de bonos. Esto permitiría también que no hiciese falta que una sola entidad asumiese los costes de desarrollo, sino que éstos podrían repartirse entre muchas (incluyendo individuos), que además sólo tendrían que pagar si finalmente el proyecto terminara con éxito. Un mecanismo muy similar a éste se propone, con mucho más detalle, en "The Wall Street performer protocol. Using software completion bonds to fund open source software development", de Chris Rasch (1991) [191].

Bibliografia

El sistema de bonos descrito está basado en el *street performer protocol* ('protocolo del artista callejero') ("The street performer protocol", en: *Third USENIX Workshop on Electronic Commerce Proceedings*, 1998 [152], y "The street performer protocol and digital copyrights", 1999 [153]), un mecanismo basado en el comercio electrónico diseñado para facilitar la financiación privada de trabajos de creación libres. Resumiendo, quien esté interesado en que se realice un determinado trabajo prometería formalmente pagar una cierta cantidad si el trabajo se realiza y es publicado libremente. Sus intenciones son buscar una nueva manera de financiar trabajos relativamente pequeños que queden a disposición de todo el mundo, pero pueden extenderse de muchas formas (los bonos para la construcción de software libre son una de ellas). Puede verse un pequeño caso de puesta en práctica de un derivado de este protocolo, el *rational street performer protocol* ('protocolo racional del artista callejero', Paul Harrison, 2002, [137]) en http://www.csse.monash.edu.au/~pfh/circle/funding_results.html, donde se aplica a la consecución de fondos para la financiación de parte de The Circle, un proyecto de software libre.

- Cooperativas de desarrolladores. En este caso, los desarrolladores de software libre, en lugar de trabajar individualmente o para una empresa, se reúnen en algún tipo de asociación (normalmente similar a una cooperativa). Por lo demás, su funcionamiento es similar al de una empresa, matizado quizás por su compromiso ético con el software libre, que puede ser parte de sus estatutos (aunque ello también lo puede hacer una empresa). En este tipo de organizaciones pueden darse combinaciones variadas de trabajo voluntario con trabajo remunerado. Un ejemplo es Free Developers.

- Sistema de donaciones. Consiste en habilitar un mecanismo de pago al autor de un determinado software en la página web que alberga el proyecto. De esta forma, los usuarios interesados en que dicho proyecto continúe publicando nuevas versiones pueden apoyarlo económicamente, median-

te la realización de donaciones voluntarias a modo de financiación para el desarrollador.

5.2. Modelos de negocio basados en software libre

Además de los mecanismos de financiación de los proyectos, de los que ya hemos hablado, otro aspecto muy relacionado con la economía que merece la pena tratar es el de los modelos de negocio. Al hablar de estos mecanismos de financiación, ya se han mencionado de pasada algunos. Ahora, en este apartado, los describiremos de forma algo más metódica.

En general, puede decirse que son muchos los modelos de negocio que se están explorando en torno al software libre, algunos más clásicos y otros más innovadores. Hay que tener en cuenta que entre los modelos más habituales en la industria del software no es fácil usar aquéllos basados en la venta de licencias de uso de programas producidos, pues en el mundo del software libre ése es un mecanismo de financiación muy difícil de explotar. Sin embargo, sí que se pueden utilizar los basados en el servicio a terceros, con la ventaja de que sin ser necesariamente el productor de un programa se puede dar soporte completo sobre él.

Venta de software libre a tanto por copia

En el mundo del software libre es difícil cobrar licencias de uso, pero no imposible. En general, no hay nada en las definiciones de software libre que impida que una empresa cree un producto y sólo lo distribuya a quien pague una cierta cantidad. Por ejemplo, un determinado productor podría decidir distribuir su producto con una licencia libre, pero sólo a quien le pague 1.000 euros por copia (de manera similar a como se hace en el mundo clásico del software propietario).

Sin embargo, aunque esto es teóricamente posible, en la práctica es bastante difícil que suceda. Porque una vez que el productor ha *vendido* la primera copia, quien la recibe puede estar motivado a tratar de recuperar su inversión vendiendo más copias a un precio más bajo (algo que no puede prohibir la licencia del programa, si éste es libre). En el ejemplo anterior, podría tratar de vender diez copias a 100 euros cada una, con lo que el producto le saldría gratis (además, ello dificultaría mucho que el productor original vendiese otra copia a 1.000 euros, puesto que el producto se podría obtener legalmente por la décima parte). Es fácil deducir cómo este proceso continuaría en cascada hasta la venta de copias a un precio cercano al coste marginal de copia, que con las tecnologías actuales es prácticamente cero.

Aun así, y teniendo en cuenta que el mecanismo descrito hará que normalmente el productor no pueda poner un precio (especialmente un precio alto) al simple hecho de la redistribución del programa, hay modelos de negocio que implícitamente hacen justamente eso. Un ejemplo es el de las distribuciones de GNU/Linux, que se venden por un precio muy bajo comparado con el de sus competidores propietarios, pero superior (y normalmente claramente superior) al coste de copia (incluso cuando se pueden descargar libremente de Internet). Por supuesto, en estos casos entran en juego otros factores, como la imagen de marca o la comodidad para el consumidor. Pero no es éste el único caso. Por lo tanto, más que indicar que con software libre "no se puede vender a tanto por copia", hay que tener en cuenta que es más difícil hacerlo, y que probablemente se obtendrá menos beneficio, pero que puede haber modelos basados justamente en eso.

Dadas estas limitaciones (y estas ventajas), desde hace unos años se están probando variantes de los modelos de negocio habituales en la industria del software, a la vez que se buscan otros más innovadores que explotar las posibili-

dades que ofrece el software libre. Sin duda, en los próximos años veremos aún más experimentación en este campo, y también tendremos más información sobre qué modelos pueden funcionar bien y en qué circunstancias.

En este apartado vamos a ofrecer un panorama de los modelos que más habitualmente encontramos hoy en día, agrupados con la intención de mostrar al lector lo que tienen de común y lo que los diferencia, centrándonos en aquéllos basados en el desarrollo y los servicios en torno a un producto de software libre. Los ingresos, en este caso, provienen directamente de estas actividades de desarrollo y servicios para el producto, pero no necesariamente implican desarrollo de nuevos productos. Cuando sí que se lleva a cabo este desarrollo, estos modelos tienen como *subproducto* la financiación de productos de software libre, por lo que son modelos especialmente interesantes cuyo impacto puede ser grande en el mundo del software libre en general.

En cualquier caso, y aunque aquí se ofrece una clasificación relativamente clara, no hay que olvidar que casi todas las empresas usan en realidad combinaciones de los modelos que describimos, entre ellos y con otros más tradicionales.

5.2.1. Mejor conocimiento

La empresa que utiliza este modelo de negocio trata de rentabilizar su conocimiento de un producto (o un conjunto de productos) libre. Sus ingresos provendrán de clientes a los que venderá servicios relacionados con ese conocimiento: desarrollos basados en el producto, modificaciones, adaptaciones, instalaciones e integraciones con otros. La ventaja competitiva de la empresa estará en gran medida ligada al mejor conocimiento del producto: por ello la empresa estará especialmente bien situada si es la productora o si participa activamente en el proyecto que lo produce.

Ésta es una de las razones por las que las empresas que utilizan este modelo suelen participar activamente en los proyectos relacionados con el software sobre el que tratan de vender servicios: es una forma muy eficiente de obtener conocimiento sobre él, y lo que es más importante, de que ese conocimiento sea reconocido. Desde luego, explicarle a un cliente que entre los empleados hay varios desarrolladores del proyecto que produce el software que, por ejemplo, se quiere modificar, puede ser una buena garantía.

Relación con los proyectos de desarrollo

Por lo tanto, este tipo de empresas tienen un gran interés en dar la imagen de que poseen un buen conocimiento de determinados productos libres. Una interesante consecuencia de esto es que su apoyo a proyectos de software libre (por ejemplo, participando activamente en ellos o permitiendo que sus empleados lo hagan durante su jornada laboral) no es por lo tanto algo meramente filantrópico. Por el contrario, puede ser uno de los activos más rentables de la empresa, ya que sus clientes lo valorarán muy positivamente como una muestra clara de que ésta conoce el producto en cuestión. Además, de esta forma podrá seguir muy de cerca el desarrollo, tratando de asegurarse, por ejemplo, de qué mejoras demandadas por sus clientes pasan a formar parte del producto desarrollado por el proyecto.

Analizándolo desde un punto de vista más general, ésta es una situación en la que ambas partes, la empresa y el proyecto de desarrollo, ganan de la colaboración. El proyecto gana por el desarrollo realizado por la empresa, o porque algunos de sus desarrolladores pasan a estar remunerados (siquiera parcialmente) por su trabajo en el proyecto. La empresa gana en conocimiento del producto, en imagen hacia sus clientes y en una cierta influencia sobre el proyecto.

Los servicios que proporcionan este tipo de empresas pueden ser muy amplios, pero normalmente consisten en desarrollos a medida, adaptaciones o integraciones de los productos en los que son expertas, o bien servicios de consultoría en los que aconsejan a sus clientes cómo utilizar mejor el producto en cuestión (especialmente si es complejo o si su correcto funcionamiento es crítico para el propio cliente).

Ejemplos

Ejemplos de empresas que hasta cierto punto utilizan este modelo de negocio son las siguientes:

- LinuxCare (http://www.linuxcare.com) [45]. Fundada en 1996, proporcionaba en sus orígenes servicios de consultoría y soporte para GNU/Linux y software libre en EE.UU., y su plantilla estaba compuesta fundamentalmente por expertos en GNU/Linux. Sin embargo, en 1992 cambió sus objetivos, y desde entonces se ha especializado en proporcionar servicios casi exclusivamente a GNU/Linux ejecutando sobre máquinas virtuales z/VM en grandes ordenadores de IBM. Su modelo de negocio ha cambiado también al de "mejor conocimiento con limitaciones", puesto que ofrece como parte fundamental de sus servicios una aplicación no libre, Levanta.

- Alcôve (http://www.alcove.com) [3]. Fundada en 1997 en Francia, proporciona principalmente servicios de consultoría, consultaría estratégica, soporte y desarrollo para software libre. Desde su fundación, Alcôve ha mantenido en plantilla a desarrolladores de varios proyectos libres, lo cual ha tratado de rentabilizar en términos de imagen. También ha intentado ofrecer una imagen, en general, de empresa vinculada a la comunidad de software libre, por ejemplo, colaborando con asociaciones de usuarios y dando publicidad a sus colaboraciones con proyectos libres (por ejemplo, desde Alcôve-Labs –http://www.alcove-labs.org– [4]).

5.2.2. Mejor conocimiento con limitaciones

Estos modelos son similares a los expuestos en el apartado anterior, pero intentan limitar la competencia a la que pueden verse sometidos. Mientras que en los modelos *puros* basados en el mejor conocimiento cualquiera puede, en principio, entrar en competencia, ya que el software utilizado es el mismo (y libre), en este caso se trata de evitar esta situación poniendo barreras a esa competencia. Estas barreras suelen consistir en patentes o licencias propietarias, que normalmente afectan a una parte pequeña (pero fundamental) del

producto desarrollado. Por eso estos modelos pueden considerarse en realidad como mixtos, en el sentido que están a caballo entre el software libre y el software propietario.

En muchos casos, la comunidad del software libre desarrolla su propia versión para ese componente, con lo que la ventaja competitiva puede desaparecer, o incluso volverse en contra de la empresa en cuestión si su *competidor* libre se convierte en el estándar del mercado y pasa a ser demandado por sus propios clientes.

Ejemplos

Hay muchos casos en los que se usa este modelo de negocio, ya que es común considerarlo menos arriesgado que el de conocimiento *puro*. Sin embargo, las empresas que lo han utilizado han tenido evoluciones variadas. Algunas de ellas son las siguientes:

- Caldera (http://www.sco.com) [16]. La historia de Caldera es complicada. En sus inicios, creó su propia distribución de GNU/Linux, orientada a las empresas: Caldera OpenLinux. En 2001 compró la división de Unix de SCO, y en 2002 cambió su nombre a SCO Group. Su estrategia empresarial ha dado tantos vuelcos como su nombre, desde su total apoyo a GNU/Linux hasta sus demandas contra IBM y Red Hat en 2003 y el abandono de su propia distribución. Pero en lo que se refiere a este apartado, el negocio de Caldera, al menos hasta 2002, es un claro exponente del mejor conocimiento con limitaciones. Caldera trataba de explotar su conocimiento de la plataforma GNU/Linux, pero limitando la competencia a la que podía verse sometida mediante la inclusión de software propietario en su distribución. Esto hacía difícil a sus clientes cambiar de distribución una vez que la habían adoptado, pues aunque las otras distribuciones de GNU/Linux incluían la parte libre de Caldera OpenLinux, no se encontraba en ellas la parte propietaria.

- Ximian (http://ximian.com/) [74]. Fundada en 1999 con el nombre de Helix Code por desarrolladores muy vinculados al proyecto GNOME, fue adquirida en agosto de 2003 por Novell. La mayor parte del software que ha desarrollado ha sido libre (en general, parte de GNOME). Sin embargo, en un ámbito muy concreto Ximian decidió licenciar un componente como software propietario: el Connector for Exchange. Este módulo permite a uno de sus productos estrella, Evolution (un gestor de información personal que incluye correo electrónico, agenda, calendario, etc.), interactuar con servidores Microsoft Exchange, muy utilizados en grandes organizaciones. De esta manera trataba de competir ventajosamente con otras empresas que proporcionaban servicios basados en GNOME, quizás con los productos desarrollados por la propia Ximian que no podían interaccionar tan fácilmente con Exchange. Salvo por este producto, el modelo de Ximian ha sido de "mejor conocimiento", y también se ha basado en ser la fuente de un programa (como veremos más adelante). En cualquier caso, este componente fue liberado en 2005.

5.2.3. Fuente de un producto libre

Este modelo es similar al basado en el mejor conocimiento, pero lo especializa, con lo que la empresa que lo utiliza es productora, de forma prácticamente íntegra, de un producto libre. Naturalmente, la ventaja competitiva aumenta al ser los desarrolladores del producto en cuestión, controlar su evolución y tenerlo antes que la competencia. Todo esto posiciona a la empresa desarrolladora en un lugar muy bueno de cara a los clientes que quieran servicios sobre ese programa. Además, es un modelo muy interesante en términos de imagen, ya que la empresa ha demostrado su potencial desarrollador con la creación y el mantenimiento de la aplicación en cuestión, lo que puede ser muy interesante a la hora de convencer a posibles clientes de sus capacidades.

Igualmente, proporciona muy buena imagen de cara a la comunidad del software libre en general, ya que ésta recibe de la empresa un nuevo producto libre que pasa a formar parte del acervo común.

Ejemplos

Son muchos los productos libres que comenzaron su desarrollo dentro de una empresa, y muy habitualmente ha continuado siendo esa empresa la que ha guiado su desarrollo posterior. A modo de ejemplo, podemos citar los casos siguientes:

- Ximian. Ya se ha mencionado cómo en parte ha usado el modelo de mejor conocimiento con limitaciones. Pero en general, Ximian ha seguido un claro modelo basado en ser la fuente de programas libres. Sus productos principales, como Evolution o Red Carpet, se han distribuido bajo licencias GPL. Sin embargo, otros también importantes, como MONO, se distribuyen en gran parte bajo licencia MIT X11 o LGPL. En todos estos casos, Ximian ha desarrollado los productos casi en exclusiva desde el comienzo. La empresa ha tratado de rentabilizar este desarrollo consiguiendo contratos para hacerlos evolucionar en ciertos sentidos, para adaptarlos a las necesidades de sus clientes, y ofreciendo personalización y mantenimiento.

- Zope Corporation (http://www.zope.com/) [75]. En 1995 se funda Digital Creations, que desarrolla un producto propietario para la gestión de anuncios clasificados vía web. En 1997 recibió una inyección de capital por parte, entre otros, de una empresa de capital riesgo, Opticality Ventures. Lo extraño (en aquella época) de esta inversión fue que como condición le pusieron que distribuyera como software libre la evolución de su producto, lo que más adelante fue Zope, uno de los gestores de contenidos más populares en Internet. Desde entonces, el modelo de negocio de la empresa fue producir Zope y productos relacionados con éste, y ofrecer servicios de adaptación y mantenimiento para todos ellos. Zope Corporation ha sabido, además, crear una dinámica comunidad de desarrolladores de software libre en torno a sus productos y colaborar activamente con ellos.

5.2.4. Fuente de un producto con limitaciones

Este modelo es similar al anterior, pero toma medidas para limitar la competencia o maximizar los ingresos. Entre las limitaciones más habituales, podemos considerar las siguientes:

- Distribución propietaria durante un tiempo, luego libre. Con o sin promesa de distribución libre posterior, cada nueva versión del producto se vende como software propietario. Pasado un tiempo (normalmente, cuando se empieza a comercializar una nueva versión, también como software propietario), esa versión pasa a distribuirse con una licencia libre. De esta manera, la empresa productora obtiene ingresos de los clientes interesados en disponer lo antes posible de nuevas versiones, y a la vez minimiza la competencia, ya que cualquier empresa que quiera competir usando ese producto sólo podrá hacerlo con la versión libre (disponible sólo cuando ya hay una nueva versión propietaria, supuestamente mejor y más completa).

- Distribución limitada durante un tiempo. En este caso, el software es libre desde que se comienza a distribuir. Pero como no hay nada en una licencia libre que obligue a distribuir el programa a todo el que lo quiera (esto es algo que quien tiene el software puede hacer o no), el productor lo distribuye durante un tiempo sólo a sus clientes, que le pagan por ello (normalmente en forma de contrato de mantenimiento). Al cabo de un

tiempo, lo distribuye a cualquiera, por ejemplo poniéndolo en un archivo de acceso público. De esta manera, el productor obtiene ingresos de sus clientes, que perciben esta disposición preferente del software como un valor añadido. Naturalmente, el modelo sólo funciona si los clientes a su vez no hacen público el programa cuando lo reciben. Para cierto tipo de clientes, esto puede no ser habitual.

En general, en estos casos las empresas desarrolladoras obtienen los beneficios mencionados, pero no a coste cero. Debido al retraso con el que el producto está disponible para la comunidad del software libre, es prácticamente imposible que ésta pueda colaborar en su desarrollo, por lo que el productor se beneficiará muy poco de contribuciones externas.

Ejemplos

Algunas empresas que utilizan este modelo de negocio son las siguientes:

- artofcode LLC (http://artofcode.com/) [9]. Desde el año 2000, artofcode comercializa Ghostscript en tres versiones (anteriormente lo había hecho Alladin Enterprises con un modelo similar). La versión más actual la distribuye como AFPL Ghostscript, bajo una licencia propietaria (que permite el uso y la distribución no comercial). La siguiente (con un retraso de un año, más o menos) la distribuye como GNU Ghostscript, bajo la GNU GPL. Por ejemplo, en el verano de 2003, la versión AFPL es la 8.11 (liberada el 16 de agosto), mientras que la versión GNU es la 7.07 (distribuida como tal el 17 de mayo, pero cuya versión AFPL equivalente es de 2002). Además, artofcode ofrece una tercera versión, con una licencia propietaria que permite la integración en productos no compatibles con la GNU GPL (en este caso usa un modelo dual, que será descrito más adelante).

- Ada Core Technologies (http://www.gnat.com/) [2]. Fue fundada en 1994 por los autores del primer compilador de Ada 95, cuyo desarrollo fue financiado en parte por el Gobierno de EE.UU., que estaba basado en GCC, el compilador de GNU. Desde el principio sus productos han sido software libre. Pero la mayoría de ellos los ofrecen primero a sus clientes, como parte de un contrato de mantenimiento. Por ejemplo, su compilador, que continúa basándose en GCC y se distribuye bajo la GNU GPL, se ofrece a sus clientes como GNAT Pro. Ada Core Technologies no ofrece este compilador al público en general de ninguna manera, y normalmente no se encuentran versiones de él en la Red. Sin embargo, con un retraso variable (en torno a un año), Ada Core Technologies ofrece las versiones *públicas* de su compilador, muy similares pero sin ningún tipo de soporte, en un archivo de FTP anónimo.

5.2.5. Licencias especiales

En estos modelos, la empresa produce un producto que distribuye bajo dos o más licencias. Al menos una de ellas es de software libre, pero las otras típicamente son propietarias y le permiten vender el producto de una forma más o menos tradicional. Normalmente, estas ventas se complementan con la oferta de consultoría y desarrollos relacionados con el producto. Por ejemplo, una empresa puede distribuir un producto como software libre bajo la GNU GPL, pero ofrecer también una versión propietaria (simultáneamente, y sin retraso para ninguna de las dos) para quien no quiera las condiciones de la GPL, por ejemplo, porque quiera integrar el producto con uno propietario (algo que la GPL no permite).

Ejemplo

Sleepycat Software (http://www.sleepycat.com/download/oslicense.html) [60]. Esta empresa fue fundada en 1996 y anuncia que desde ese momento ha tenido beneficios (lo que desde luego es notable en una empresa relacionada con el software). Sus productos, incluyendo Berkeley DB (un gestor de datos muy popular que puede empotrarse fácilmente en otras aplicaciones), se distribuyen bajo una licencia libre que especifica que en caso de empotrarse en otro producto, ha de ofrecerse el código fuente de ambos. Sleepycat ofrece servicios de consultoría y desarrollo para sus productos, pero además los ofrece bajo licencias que permiten empotrarlos sin tener que distribuir el código fuente. Naturalmente, esto lo hace bajo contrato específico, y en general, en régimen de venta como software propietario. En 2005, Sleepycat Software fue adquirida por Oracle.

5.2.6. Venta de marca

Aunque puedan conseguirse productos muy similares por menos dinero, muchos clientes están dispuestos a pagar el extra por comprar una *marca*. Este principio es utilizado por empresas que invierten en establecer una marca con buena imagen y bien reconocida que les permita luego vender con margen suficiente productos libres. En muchos casos no sólo venden esos productos, sino que los acompañan de servicios que los clientes aceptarán también como valor añadido.

Los casos más conocidos de este modelo de negocio son las empresas que comercializan distribuciones GNU/Linux. Estas empresas tratan de vender algo que en general se puede obtener a un coste bastante más bajo en la Red (o en otras fuentes con menos imagen de marca). Por ello han de conseguir que el consumidor reconozca su marca y esté dispuesto a pagar el sobreprecio. Para ello no sólo invierten en publicidad, sino que también ofrecen ventajas objetivas (por ejemplo, una distribución bien conjuntada o un canal de distribución que llegue hasta las cercanías del cliente). Además, suelen ofrecer a su alrededor una gran cantidad de servicios (desde formación hasta programas de certificación para terceras partes), tratando de rentabilizar al máximo esa imagen de marca.

Ejemplo

Red Hat (http://www.redhat.com) [56]. Red Hat Linux comenzó a distribuirse en 1994 (la empresa empezó a conocerse con el nombre actual en 1995). Durante mucho tiempo Red Hat consiguió colocar su nombre como el de la distribución de GNU/Linux por excelencia (aunque a mediados de la década de 2000 comparte esa posición con otras empresas, como OpenSUSE, Ubuntu, y quizás Debian). Varios años después Red Hat comercializa todo tipo de servicios relacionados con la distribución, con GNU/Linux y con software libre en general.

5.3. Otras clasificaciones de modelos de negocio

En la literatura sobre software libre, hay otras clasificaciones de modelos de negocio clásicas. A modo de ejemplo ofrecemos a continuación una de ellas.

5.3.1. Clasificación de Hecker

La clasificación que se ofrece en "Setting up shop: the business of open-source software" (Frank Hecker, 1998) [141] fue la más usada por la publicidad de la Open Source Initiative, y también una de las primeras en tratar de categorizar los negocios que estaban surgiendo por aquella época. Sin embargo, incluye varios modelos poco centrados en software libre (en los que éste es poco más que un acompañante al modelo principal). En cualquier caso, los modelos que describe son los siguientes:

- *Support seller* (venta de servicios relacionados con el producto). La empresa promueve un producto libre (que ha desarrollado o en el desarrollo del cual participa activamente) y vende servicios, como consultoría o adaptación a necesidades concretas para éste.

- *Loss leader* (venta de otros productos propietarios). En este caso, el programa libre se utiliza para promover de alguna forma la venta de otros productos propietarios relacionados con él.

- *Widget frosting* (venta de hardware). El negocio fundamental es la venta de hardware y el software libre se considera un complemento de éste que puede ayudar a la empresa a obtener una ventaja competitiva.

- *Accessorizing* (venta de accesorios). Se comercializan productos relacionados con el software libre, como libros, dispositivos informáticos, etc.

- *Service enabler* (venta de servicios). El software libre sirve para crear un servicio (normalmente accesible en línea) del que la empresa obtiene algún beneficio.

- *Brand licensing* (venta de marca). Una empresa registra marcas que consigue asociar con programas libres, probablemente desarrollados por ella. Luego obtiene ingresos cuando vende derechos del uso de esas marcas.

- *Sell it, free it* (vende, libera). Es un modelo similar al de *loss leader*, pero realizado de forma cíclica. Primero se comercializa un producto como software libre. Si se consigue que tenga cierto éxito, la siguiente versión se distribuye como software propietario durante un tiempo, al cabo del cual se libera también. Para entonces, se empieza a distribuir una nueva versión propietaria, y así sucesivamente.

- *Software franchising* (franquicia de software). Una empresa franquicia el uso de sus marcas relacionadas con un programa libre determinado.

Nota

El lector habrá podido observar que esta clasificación es bastante diferente de la que hemos ofrecido nosotros, pero aun así algunas de sus categorías coinciden casi exactamente con algunas de las nuestras.

5.4. Impacto sobre las situaciones de monopolio

El mercado informático tiende a la dominación de un producto en cada uno de sus segmentos. Los usuarios quieren rentabilizar el esfuerzo realizado en aprender cómo funciona un programa, las empresas quieren encontrar a gente formada en el uso de su software, y todos quieren que los datos que gestionan puedan ser entendidos por los programas de las empresas y las personas con las que se relacionan. Por eso, cualquier iniciativa dedicada a romper una situación *de facto* en la que un producto domina claramente el mercado está destinada a producir más de lo mismo: si tiene éxito, vendrá otro a ocupar ese hueco, y en breve tendremos un nuevo producto dominante. Sólo los cambios tecnológicos producen, durante un tiempo, la inestabilidad suficiente como para que nadie domine claramente.

Pero el hecho de que haya un producto dominante no ha de llevar necesariamente a la constitución de un monopolio empresarial. Por ejemplo, la gasolina es un producto que casi domina el mercado de combustibles para turismos, pero (en un mercado de la gasolina libre) hay muchas empresas productoras y distribuidoras de este producto. En realidad, al hablar de software, lo preocupante es lo que ocurre cuando un producto llega a dominar el mercado porque ese producto tiene una sola empresa proveedora posible. El software libre ofrece una alternativa a esta situación: los productos libres pueden estar promovidos por una empresa en concreto, pero esta empresa no los controla, o al menos no hasta los extremos a los que nos tiene acostumbrados el software propietario. En el mundo del software libre, un producto dominante no conlleva necesariamente un monopolio de empresa. Por el contrario, sea el que sea el producto que domine el mercado, muchas empresas pueden competir en proporcionarlo, mejorarlo, adaptarlo a las necesidades de sus clientes y ofrecer servicios en torno a él.

5.4.1. Elementos que favorecen los productos dominantes

En informática es muy común que haya un producto claramente dominante en cada segmento de mercado. Y eso es normal por varios motivos, entre los que cabe destacar los siguientes:

- Formatos de datos. En muchos casos el formato de datos está fuertemente ligado a una aplicación. Cuando un número suficientemente alto de gente la utiliza, su formato de datos se convierte en estándar *de facto*, y las presiones para usarlo (y por tanto, la aplicación) son formidables.

- Cadenas de distribución. Normalmente uno de los problemas para empezar a usar un programa es obtener una copia de él. Y suele ser difícil encontrar los programas que no son líderes en su mercado. Las cadenas de distribución son costosas de mantener, de forma que los competidores minoritarios lo tienen difícil para llegar a la tienda de informática, donde

el usuario final los pueda comprar. El producto dominante, sin embargo, lo tiene fácil: el primer interesado en tenerlo va a ser la propia tienda de informática.

- Marketing. El marketing "gratuito" que obtiene un producto una vez que lo usa una fracción significativa de una población determinada es enorme. El "boca a boca" funciona mucho, también el preguntar e intercambiar información con los conocidos. Pero sobre todo el impacto en los medios es muy grande: las revistas de informática hablarán una y otra vez de un producto si parece ser el que más se usa; habrá cursos de formación en torno a él, libros que lo describan, entrevistas a sus usuarios, etc.

- Inversión en formación. Una vez que se ha invertido tiempo y recursos en aprender cómo funciona una herramienta, se está muy motivado para no cambiarla. Además, usualmente esa herramienta es la que ya domina el mercado, porque es más fácil encontrar personal y material que ayuden a aprender a usarla.

- Software preinstalado. Recibir una máquina con software ya instalado es, desde luego, un gran incentivo para usarlo, incluso si hay que pagar por él aparte. Y normalmente, el tipo de software que el vendedor de la máquina estará dispuesto a preinstalar será solamente el más utilizado.

5.4.2. El mundo del software propietario

En el mundo del software propietario la aparición de un producto dominante en un segmento cualquiera equivale a un monopolio por parte de la empresa que lo produce. Por ejemplo, tenemos estas situaciones monopolísticas *de facto* (o casi) de producto y empresa en los mercados de sistemas operativos, autoedición, bases de datos, diseño gráfico, procesadores de textos, hojas de cálculo, etc.

Y esto es así porque la empresa en cuestión tiene un gran control sobre el producto líder, tan grande que sólo ellos pueden marcar su evolución, las líneas fundamentales en las que se va a desarrollar, su calidad, etc. Los usuarios tienen muy poco control, dado que estarán muy poco motivados para probar con otros productos (por las razones que se han comentado en el apartado anterior). Ante esto, poco podrán hacer, salvo intentar desafiar la posición dominante del producto mejorando excepcionalmente los suyos (para tratar de contrarrestar esos mismos motivos), normalmente con poco éxito.

Esta situación pone a todo el sector en manos de la estrategia de la empresa dominante. Todos los actores dependen de ella, e incluso el desarrollo de la tecnología software en ese campo estará mediatizado por las mejoras que le haga a su producto. En el caso general, ésta es una situación en la que aparecen

los peores efectos económicos del monopolio, y en particular, la falta de motivación de la empresa líder para acercar el producto a las necesidades (siempre en evolución) de sus clientes. éstos se han convertido en un mercado cautivo.

5.4.3. La situación con software libre

Sin embargo, en el caso del software libre un producto dominante no se traduce automáticamente en un monopolio de empresa. Si el producto es libre, cualquier empresa puede trabajar con él, mejorarlo, adaptarlo a las necesidades de un cliente, y en general, ayudar en su evolución. Además, precisamente por su posición dominante, habrá muchas empresas interesadas en trabajar con él. Si el productor "original" (la empresa que desarrolló originalmente el producto) quiere permanecer en el negocio, ha de competir con todas ellas, y por eso estará muy motivada para hacer evolucionar el producto precisamente en la línea que sus usuarios quieran. Naturalmente, tendrá la ventaja de un mejor conocimiento del programa, pero eso es todo. Tienen que competir por cada cliente.

Por lo tanto, la aparición de productos dominantes se traduce, en el mundo del software libre, en una mayor competencia entre empresas. Y con ello los usuarios retoman el control: las empresas en competencia no pueden hacer otra cosa que hacerles caso si quieren sobrevivir. Y precisamente esto es lo que asegurará que el producto mejore.

Productos libres que son dominantes en su sector

Apache es desde hace tiempo líder en el mercado de servidores web. Pero hay muchas empresas detrás de Apache, desde algunas muy grandes (como IBM) hasta otras muy pequeñas. Y todas ellas no tienen más remedio que competir mejorándolo y normalmente contribuyendo al proyecto con sus mejoras. A pesar de que Apache es casi un monopolio en muchos ámbitos (por ejemplo, es casi el único servidor web que se considera sobre la plataforma GNU/Linux o *BSD), no depende de una sola empresa, sino de literalmente decenas de ellas.

Las distribuciones de GNU/Linux son también un caso interesante. GNU/Linux no es, desde luego, un monopolio, pero es posiblemente la segunda opción en el mercado de sistemas operativos. Y eso no ha forzado una situación en la que una empresa tenga su control. Al contrario, hay decenas de distribuciones, realizadas por empresas diferentes, que compiten libremente en el mercado. Cada una de ellas trata de ofrecer mejoras que sus competidores tienen que adoptar a riesgo de quedar fuera de él. Pero además, no pueden separarse demasiado de lo que es "GNU/Linux estándar", pues eso sería rechazado por los usuarios como una "salida de la norma". La situación después de varios años de crecimiento de la cuota de mercado de GNU/Linux nos muestra decenas de empresas que compiten y hacen evolucionar el sistema. Y de nuevo, todas ellas van detrás de la satisfacción de las necesidades de sus usuarios. Sólo así pueden mantenerse en el mercado.

GCC es un producto dominante en el mundo de compiladores C y C++ para el mercado GNU/Linux. Y sin embargo, eso no ha llevado a ninguna situación de monopolio de empresa, incluso cuando Cygnus (hoy Red Hat) se encargó durante mucho tiempo de coordinar su desarrollo. Hay muchas empresas que hacen mejoras en el sistema y todas ellas compiten, cada una en su nicho específico, por satisfacer las demandas de sus usuarios. De hecho, cuando alguna empresa u organización específica ha fallado en el trabajo de coordinación (o así lo han percibido una parte de los usuarios) ha habido espacio para un *fork* (bifurcación) del proyecto, con dos productos en paralelo durante un tiempo, hasta que eventualmente han vuelto a unirse (como está ocurriendo ahora con GCC 3.x).

5.4.4. Estrategias para constituirse en monopolio con software libre

A pesar de que el mundo del software libre es mucho más hostil a los monopolios de empresa que el mundo del software propietario, hay estrategias que una empresa puede utilizar para tratar de aproximarse a una situación de dominación monopolística de un mercado. éstas son prácticas comunes en muchos otros sectores económicos, y para evitarlas trabajan las entidades de regulación de la competencia, por lo que no hablaremos de ellas en detalle. Sin embargo, sí que mencionaremos una que es hasta cierto punto específica del mercado del software, y que ya se está experimentando en algunas situaciones: la aceptación de productos certificados por terceros.

Cuando una empresa quiere distribuir un producto software (libre o propietario) que funcione en combinación con otros, es común "certificar" ese producto para una cierta combinación. El fabricante se compromete a ofrecer servicios (actualización, soporte, resolución de problemas, etc.) sólo si el cliente le asegura que está usando el producto en un entorno certificado por él. Por ejemplo, un fabricante de gestores de bases de datos puede certificar su producto para una determinada distribución de GNU/Linux, y para ninguna otra. Eso implicará que sus clientes tendrán que usar esa distribución de GNU/Linux u olvidarse del soporte por parte del fabricante (cosa que, si el producto es propietario, puede ser imposible en la práctica). Si un fabricante dado consigue una posición claramente dominante como producto certificado de terceras partes, los usuarios no van a tener muchas más posibilidades que utilizar ese producto. Si en ese segmento la certificación es importante, estaremos de nuevo ante una situación de empresa monopolística.

Nota

Hasta cierto punto, en el mercado de distribuciones de GNU/Linux se empiezan a observar ciertos casos de situaciones tendentes al monopolio *de facto* en la certificación. Por ejemplo, hay muchos fabricantes de productos propietarios que sólo certifican sus productos sobre una distribución de GNU/Linux dada (muy habitualmente, Red Hat Linux). Por el momento esto no parece redundar en una situación monopolística por parte de ninguna empresa, lo que podría ser debido a que en el mercado de distribuciones de GNU/Linux, la certificación debe de tener poca importancia para los usuarios. Pero sólo el futuro dirá si en algún momento esta situación se acerca a una de monopolio *de facto*.

Sin embargo, es importante tener en cuenta dos comentarios con respecto a lo expuesto. El primero es que estas posiciones monopolísticas no serán fáciles de conseguir, y en cualquier caso se conseguirán por mecanismos en general "no informáticos" (a diferencia de la situación de producto dominante, que como ya hemos visto es relativamente normal, a la cual se llega por mecanismos puramente relacionados con la informática y sus patrones de uso). El segundo es que si todo el software utilizado es libre, esta estrategia tiene pocas posibilidades de éxito (si es que tiene alguna). Un fabricante podrá conseguir

que muchas empresas certifiquen para sus productos, pero los clientes siempre podrán buscar fuentes de servicio y soporte diferentes de esas empresas que han certificado para él si así lo consideran conveniente.

6. Software libre y administraciones públicas

> "[...] el software, para ser aceptable para el Estado, no basta con que sea técnicamente suficiente para llevar a cabo una tarea, sino que además las condiciones de contratación deben satisfacer una serie de requisitos en materia de licencia, sin los cuales el Estado no puede garantizar al ciudadano el procesamiento adecuado de sus datos, velando por su integridad, confidencialidad y accesibilidad a lo largo del tiempo, porque son aspectos muy críticos para su normal desempeño."
>
> Edgar David Villanueva Núñez (carta de respuesta al gerente general de Microsoft del Perú, 2001)

Las instituciones públicas, tanto las que tienen capacidad legislativa como las que se dedican a administrar el Estado (las "administraciones públicas"), tienen un papel muy relevante en lo que se refiere a adopción y promoción de tecnologías. Aunque prácticamente en el año 2000 no hubo (salvo casos anecdóticos) interés de estas instituciones por el fenómeno del software libre, la situación comenzó a cambiar a partir de esa fecha. Por un lado, muchas administraciones públicas comenzaron a emplear software libre como parte de su infraestructura informática. Por otro, en su papel de promotoras de la sociedad de la información, algunas empezaron a promover directa o indirectamente el desarrollo y el uso de software libre. Además, algunos cuerpos legislativos se han ido ocupando (poco a poco) del software libre, a veces favoreciendo su desarrollo, a veces dificultándolo, y a veces simplemente teniéndolo en cuenta.

Antes de entrar en detalle, es conveniente recordar que el software libre se desarrolló durante mucho tiempo al margen del apoyo explícito (e incluso del interés) de las instituciones públicas. Por ello, la reciente atención que está despertando en muchas de ellas no está exenta de polémicas, confusiones y problemas. Además, durante los últimos años las iniciativas relacionadas con los estándares abiertos van también cobrando fuerza, y se traducen en muchos casos en medidas (más o menos directas) relacionadas con el software libre.

En este capítulo trataremos de exponer cuál es la situación actual y qué peculiaridades tiene el software libre cuando se relaciona con "lo público".

6.1. Impacto en las administraciones públicas

Ha habido varios estudios que se han centrado en las implicaciones del uso del software libre en las administraciones públicas (por ejemplo, "Open source software for the public administration", 2004 [159]; "Open-source software in e-Government, analysis and recommendations drawn up by a working group under the danish board of technology", 2002 [180]; "Free software / open source: information society opportunities for Europe?", 1999 [132], y "The case for government promotion of open source software", 1999 [213]). A continuación comentamos algunas de las más destacables (tanto positivas como negativas).

6.1.1. Ventajas e implicaciones positivas

Algunas de las ventajas del uso de software libre en las administraciones públicas y de las principales nuevas perspectivas que permite son las siguientes:

1) Fomento de la industria local

Una de las mayores ventajas del software libre es la posibilidad de desarrollar industria local de software. Cuando se usa software propietario, todo lo gastado en licencias va directamente al fabricante del producto, y además esa compra redunda en el fortalecimiento de su posición, lo cual no es necesariamente perjudicial, pero sí que es poco eficiente para la región vinculada a la Administración si analizamos la alternativa de usar un programa libre.

En este caso, las empresas locales podrán competir proporcionando servicios (y el propio programa) a la Administración, en condiciones muy similares a cualquier otra empresa. Digamos que, de alguna manera, la Administración está allanando el campo de juego y haciendo más fácil que cualquiera pueda competir en él. Y naturalmente, entre esos "cualquiera" estarán las empresas locales, que tendrán la posibilidad de aprovechar sus ventajas competitivas (mejor conocimiento de las necesidades del cliente, cercanía geográfica, etc.).

2) Independencia de proveedor y competencia en el mercado

Es obvio que cualquier organización preferirá depender de un mercado en régimen de competencia que de un solo proveedor, que puede imponer las condiciones en las que proporciona su producto. Sin embargo, en el mundo de la Administración, esta preferencia se convierte en requisito fundamental, e incluso en obligación legal en algunos casos. La Administración no puede, en general, elegir contratar a un suministrador dado, sino que debe especificar sus necesidades de forma que cualquier empresa interesada que cumpla unas ciertas características técnicas y que proporcione el servicio o el producto demandado con una cierta calidad, pueda optar a un contrato.

De nuevo, en el caso del software propietario, para cada producto no hay más que un proveedor (aunque use una variedad de intermediarios). Si se especifica un producto dado, se está decidiendo también a qué proveedor contratará la Administración. Y en muchos casos es prácticamente imposible evitar especificar un cierto producto, cuando estamos hablando de programas de ordenador. Razones de compatibilidad dentro de la organización o de ahorros en formación y administración, u otras muchas, hacen habitual que una administración decida usar un cierto producto.

La única salida a esta situación es que el producto especificado sea libre. En ese caso, cualquier empresa interesada podrá proporcionarlo, y también podrá proporcionar cualquier tipo de servicio sobre él (sujeto únicamente a sus capacidades y conocimientos del producto). Además, en caso de contratar de

esta manera, la Administración podrá en el futuro cambiar de proveedor si así lo desea, inmediatamente y sin ningún problema técnico, puesto que aunque cambie de empresa, el producto que usará será el mismo.

3) Flexibilidad y adaptación a las necesidades exactas

Aunque la adaptación a sus necesidades exactas es algo que necesita cualquier organización que precisa la informática, las peculiaridades de la Administración hacen que éste sea un factor muy importante para el éxito de la implantación de un sistema informático. En el caso de usar software libre, la adaptación puede hacerse con mucha mayor facilidad, y lo que es más importante, sirviéndose de un mercado con competencia si hace falta contratarla.

Cuando la Administración compra un producto propietario, modificarlo pasa normalmente por alcanzar un acuerdo con su productor, que es el único que legalmente (y muchas veces técnicamente) puede hacerlo. En esas condiciones, es difícil realizar buenas negociaciones, sobre todo si el productor no está excesivamente interesado en el mercado que le ofrece la administración en cuestión. Sin embargo, usando un producto libre, la Administración puede modificarlo a su antojo si dispone de personal para ello, o contratar externamente la modificación. Esta contratación la puede realizar en principio cualquier empresa que tenga los conocimientos y las capacidades para ello, por lo que es de esperar la concurrencia de varias. Eso, necesariamente, tiende a abaratar los costes y a mejorar la calidad.

El caso de las distribuciones de GNU/Linux

En España se ha hecho común durante los últimos años la creación de distribuciones de GNU/Linux propias por parte de ciertas comunidades autónomas. Esta tendencia comenzó con GNU/Linux, pero hoy en día son muchas más. Aunque la existencia de todas estas distribuciones ha sido criticada por algunos expertos, supone un claro exponente de la flexibilidad que permite el software libre. Cada administración pública puede, gastando relativamente pocos recursos, contratar la adaptación de GNU/Linux adaptándola a sus necesidades y preferencias, prácticamente sin límite. Por ejemplo, puede cambiar el aspecto del escritorio, elegir el conjunto de aplicaciones y el idioma por defecto, mejorar la localización de las aplicaciones, etc. En otras palabras: si se quiere, el escritorio (y cualquier otro elemento del software que funcione en el ordenador) puede adaptarse a unas necesidades exactas.

Por supuesto, esta adaptación supondrá unos costes, pero la experiencia muestra que puede conseguirse mucho con relativamente pocos recursos, y la tendencia parece indicar que cada vez será más fácil (y más barato) realizar distribuciones personalizadas.

4) Adopción más fácil de estándares abiertos

Por su propia naturaleza, es habitual que los programas libres utilicen estándares abiertos o estándares no privativos. De hecho, casi por definición, cualquier aspecto de un programa libre que se quiera considerar puede reproducirse con facilidad, y por lo tanto, no es privativo. Por ejemplo, los protocolos que un programa libre utiliza para interactuar con otros programas pueden estudiarse y reproducirse, por lo que no son privativos. Pero además, muy habitualmente y por el propio interés de los proyectos, se intenta usar estándares abiertos.

En cualquier caso, y sea cual sea la razón, es un hecho que los programas libres utilizan habitualmente estándares no privativos para el intercambio de datos. Las ventajas que ello supone para las administraciones públicas van mucho más allá de las que puede encontrar cualquier organización, dado que la promoción de estándares privativos (incluso de forma indirecta, al usarlos) en ese caso sería mucho más preocupante. Y en un aspecto al menos el uso de estándares no privativos es fundamental, el de la interacción con el ciudadano, que no ha de verse obligado a comprar ningún producto a una empresa en particular para poder relacionarse con su administración.

5) Escrutinio público de seguridad

Para una administración pública, poder garantizar que sus sistemas informáticos hacen sólo lo que está previsto que hagan es un requisito fundamental, y en muchos países, un requisito legal. No son pocas las veces que esos sistemas manejan datos privados, en los que pueden estar interesados terceros (pensemos en datos fiscales, penales, censales, electorales, etc.). Difícilmente si se usa una aplicación propietaria, sin código fuente disponible, puede asegurarse que efectivamente esa aplicación tratará esos datos como debe. Pero incluso si se ofrece su código fuente, las posibilidades que tendrá una institución pública para asegurar que no contiene código *extraño* serán muy limitadas. Sólo si se puede encargar ese trabajo de forma habitual y rutinaria a terceros, y además cualquier parte interesada puede escrutarlos, la Administración podrá estar razonablemente segura de cumplir con ese deber fundamental, o al menos, de tomar todas las medidas en su mano para hacerlo.

6) Disponibilidad a largo plazo

Muchos datos que manejan las administraciones, y los programas que sirven para calcularlos, han de estar disponibles dentro de decenas de años. Es muy difícil asegurar que un programa propietario cualquiera estará disponible cuando hayan pasado esos períodos de tiempo, y más si lo que se quiere es que funcione en la plataforma habitual en ese momento futuro. Por el contrario, es muy posible que su productor haya perdido interés en el producto y que no lo haya portado a nuevas plataformas, o que sólo esté dispuesto a hacerlo ante grandes contraprestaciones económicas. De nuevo, hay que recordar que sólo él puede hacer ese porte, y por lo tanto, será difícil negociar con él. En el caso del software libre, en cambio, la aplicación está disponible, con seguridad, para que cualquiera la porte y la deje funcionando según las necesidades de la Administración. Si eso no sucede de forma espontánea, la Administración siempre puede dirigirse a varias empresas buscando la mejor oferta para hacer el trabajo. Así se puede garantizar que la aplicación, y los datos que maneja, estarán disponibles cuando haga falta.

7) Impacto más allá del uso por parte de la Administración

Muchas aplicaciones utilizadas o promovidas por las administraciones públicas son también utilizadas por muchos otros sectores de la sociedad. Por ello, cualquier inversión pública en el desarrollo de un producto libre que le interese beneficia no sólo a la propia administración, sino también a todos los ciudadanos, que podrán usar ese producto para sus tareas informáticas, quizás con las mejoras aportadas por la Administración.

Bibliografía

El lector interesado en un informe sobre las ventajas del software libre para la administración, escrito en el contexto estadounidense de 1999, puede consultar "The case for government promotion of open source software" (Mitch Stoltz, 1999) [213].

Nota

Un caso muy especial, pero de gran impacto, que muestra este mejor aprovechamiento de los recursos públicos es la localización (adaptación a los usos y costumbres de una comunidad) de un programa. Aunque el aspecto más visible de la localización es la traducción del programa y su documentación, hay otros que se ven también afectados por ella (desde la utilización de símbolos de la moneda local o la presentación de la fecha y la hora en formatos habituales en la comunidad en cuestión, hasta el uso de ejemplos y formas de expresión adecuados a las costumbres locales en la documentación).

En cualquier caso, está claro que si una administración pública dedica recursos a que una determinada aplicación sea localizada para sus necesidades, es más que probable que esas necesidades coincidan con las de sus ciudadanos, de manera que no sólo consiguirá tener un programa informático que satisfaga sus necesidades, sino que también podrá, sin gasto extra, ponerlo a disposición de cualquier ciudadano que pueda utilizarlo con provecho. Por ejemplo, cuando una administración pública financia la adaptación de un programa ofimático a una lengua que se habla en su ámbito de actuación, no sólo podrá usarlo en sus propias dependencias, sino que también podrá ofrecerlo a sus ciudadanos, con lo que eso puede suponer de fomento de la sociedad de la información.

6.1.2. Dificultades de adopción y otros problemas

Sin embargo, aunque las ventajas de uso del software libre en las administraciones sean muchas, también son muchas las dificultades que se han de afrontar a la hora de ponerlo en práctica. Entre ellas pueden destacarse las siguientes:

1) Desconocimiento y falta de decisión política

El primer problema con el que se encuentra el software libre para su introducción en las administraciones es uno que, sin duda, es compartido por otras organizaciones: el software libre es aún muy desconocido entre quienes toman las decisiones.

Afortunadamente, éste es un problema que se va solucionando paulatinamente, pero aún en muchos ámbitos de las administraciones el software libre es percibido como algo *extraño*, de manera que tomar decisiones en la línea de usarlo implica asumir ciertos riesgos.

Unido a esto, suele encontrarse un problema de decisión política. La principal ventaja del software libre en la Administración no es el coste (siendo éste, de todas formas, importante, sobre todo cuando se habla de despliegues para gran cantidad de puestos), sino como ya hemos visto, sobre todo ventajas *de fondo*, estratégicas. Y por tanto, quedan en gran medida en el ámbito de las decisiones políticas, no técnicas. Sin voluntad política de cambiar los sistemas informáticos y la filosofía con la que se contratan, es difícil avanzar en la implantación de software libre en la Administración.

2) Poca adecuación de los mecanismos de contratación

Los mecanismos de contratación que se usan hoy en día en la Administración, desde los modelos de concurso público habituales hasta la división del gasto en partidas, están diseñados fundamentalmente para la compra de productos informáticos, y no tanto para la adquisición de servicios relacionados con los programas. Sin embargo, cuando se utiliza software libre, habitualmente no hay producto que comprar o su precio es casi despreciable. Por el contrario, para aprovechar las posibilidades que ofrece el software libre, es conveniente poder contratar servicios a su alrededor. Esto hace preciso que, antes de utilizar seriamente software libre, se hayan diseñado mecanismos burocráticos adecuados que faciliten la contratación en estos casos.

3) Falta de estrategia de implantación

Muchas veces el software libre comienza a usarse en una administración simplemente porque el coste de adquisición es más bajo. Es muy habitual en estos casos que el producto en cuestión se incorpore al sistema informático sin mayor planificación, y en general, sin una estrategia global de uso y aprovechamiento de software libre. Esto causa que la mayor parte de sus ventajas se pierdan por el camino, ya que todo queda en el uso de un *producto más barato*, cuando ya hemos visto que, en general, las mayores ventajas son de otro tipo.

Si a esto unimos que si no se hace un diseño adecuado del cambio, el uso de software libre puede suponer unos costes de transición no despreciables, veremos que ciertas experiencias aisladas, y fuera de un marco claro, de uso de software libre en la Administración pueden resultar fallidas y frustrantes.

4) Escasez o ausencia de productos libres en ciertos segmentos

La implantación de software libre en cualquier organización puede chocar con la falta de alternativas libres de la calidad adecuada para cierto tipo de aplicaciones. En esos casos, la solución es complicada: lo único que se puede hacer es tratar de promover la aparición del producto libre que se necesita.

Afortunadamente, las administraciones públicas están en una buena posición para estudiar seriamente si les conviene fomentar, o incluso financiar o cofinanciar, el desarrollo de ese producto. Hay que recordar que entre sus fines suele estar, por ejemplo, que sus administrados puedan acceder mejor a la sociedad de la información o el fomento del tejido industrial local. Sin duda, la creación de muchos programas libres incidirá positivamente en ambos objetivos, por lo que al mero cálculo de coste/beneficio directo habría que sumar los beneficios indirectos que esta decisión causaría.

5) Interoperabilidad con sistemas existentes

No es habitual que se acometa una migración a software libre con todo el sistema informático completo, a la vez. Por ello, es importante que la parte que se quiere migrar siga funcionando correctamente en el marco del resto del software con el que tendrá que interoperar. éste es un problema

conocido en cualquier migración (aunque sea a un producto privativo), pero puede tener una especial incidencia en el caso del software libre. En cualquier caso, será algo que habrá que tener en cuenta al estudiar la transición. Afortunadamente, muchas veces se pueden hacer adaptaciones del software libre que se ha de instalar para que interopere adecuadamente con otros sistemas, pero si es el caso, habrá que considerar este punto al presupuestar los gastos de la migración.

6) Migración de los datos
Éste es un problema genérico de cualquier migración a nuevas aplicaciones que utilicen un formato de datos diferente, incluso si son privativas. De hecho, en el caso del software libre este problema a veces está muy mitigado, pues es habitual hacer un esfuerzo especial para prever cuantos formatos y estándares de intercambio de datos sea posible. Pero habitualmente es preciso migrar los datos. Y el coste de hacerlo es alto. Por ello, al hacer el cálculo de lo que puede suponer una migración a software libre, éste es un factor que habrá que tener muy en cuenta.

6.2. Actuaciones de las administraciones públicas en el mundo del software

Las administraciones públicas actúan sobre el mundo del software al menos de tres formas:

- Comprando programas y servicios relacionados con ellos. Las administraciones, como grandes usuarios de informática, son un actor fundamental en el mercado del software.

- Promoviendo de diversas formas el uso (y la adquisición) de ciertos programas en la sociedad. Esta promoción se hace a veces ofreciendo incentivos económicos (desgravaciones fiscales, incentivos directos, etc.), a veces con información y recomendaciones, a veces por "efecto ejemplo"...

- Financiando (directa o indirectamente) proyectos de investigación y desarrollo que diseñen el futuro de la informática.

En cada uno de estos ámbitos el software libre puede presentar ventajas específicas (además de las ya descritas en el apartado anterior) interesantes tanto para la Administración como para la sociedad en general.

6.2.1. ¿Cómo satisfacer mejor las necesidades de las administraciones públicas?

Las administraciones públicas son grandes consumidoras de informática. En lo que al software se refiere, compran habitualmente tanto productos de consumo masivo (*off-the-shelf*) como sistemas a medida. Desde este punto de vis-

ta, son fundamentalmente grandes centros de compras, similares a las grandes empresas, aunque con sus propias peculiaridades. Por ejemplo, en muchos ámbitos se supone que las decisiones de compra de las administraciones públicas no han de tener en cuenta simplemente parámetros de coste frente a funcionalidad, sino otros, como impacto de la compra en el tejido industrial y social o consideraciones estratégicas a largo plazo, que pueden también ser importantes.

En cualquier caso, lo habitual hoy en día en lo que a software de consumo masivo se refiere es utilizar los productos propietarios líderes del mercado. La cantidad de recursos públicos que se gastan los ayuntamientos, las comunidades autónomas, la Administración central y las administraciones europeas en comprar licencias de Windows, Office u otros productos similares es ciertamente considerable. Pero progresivamente las soluciones libres van penetrando en este mercado. Cada vez más, se están considerando soluciones basadas en software libre para los servidores, y productos como OpenOffice, y GNU/Linux con GNOME o KDE son cada vez más considerados en el escritorio.

¿Qué se puede ganar con esta migración a software libre? Para ilustrarlo, consideremos el escenario siguiente. Supongamos que con una fracción de lo gastado en dos o tres productos propietarios "estrella" por todas las administraciones europeas (o probablemente las de cualquier país desarrollado de tamaño medio), se podría promover un concurso público para que una empresa (o dos, o tres, o cuatro) mejorase y adaptase los programas libres ahora disponibles para que en el plazo de uno o dos años estuvieran listos para su uso masivo al menos para ciertas tareas típicas (si no lo estuvieran ya). Imaginemos por ejemplo un esfuerzo coordinado, a escala nacional o europea, para que todas las administraciones participasen en un consorcio que se encargase de la gestión de estos concursos. En poco tiempo habría una industria "local" especializada en realizar estas mejoras y estas adaptaciones. Y las administraciones podrían elegir entre las tres o cuatro distribuciones libres producidas por esta industria. Para fomentar la competencia, se podría recompensar económicamente a cada empresa según la cantidad de administraciones que eligiesen usar su distribución. Y todo el resultado de esta operación, al ser software libre, estaría también a disposición de empresas y usuarios individuales, que en muchos casos tendrían necesidades similares a las de las administraciones.

En el caso del software hecho a medida, el proceso habitual por el momento pasa normalmente por contratar con una empresa los programas necesarios bajo un modelo propietario. Todo el desarrollo realizado a petición de la Administración es propiedad de la empresa que lo desarrolla. Y normalmente la administración contratante queda atada a su proveedor para todo lo que tenga que ver con mejoras, actualizaciones y soporte, en un círculo vicioso que dificulta mucho la competencia y ralentiza el proceso de modernización de

las administraciones públicas. Lo que es peor, muchas veces el mismo programa es vendido una y otra vez a administraciones similares, aplicando en cada nuevo caso los costes que habría supuesto hacer el desarrollo desde cero.

Consideremos de nuevo un escenario que muestre cómo podrían ser las cosas de otra forma. Un consorcio de administraciones públicas con necesidades de un cierto software a medida podría exigir que el resultado obtenido fuera software libre. Esto permitiría que otras administraciones se beneficiasen también del trabajo y que a medio plazo estuviesen interesadas en colaborar en el consorcio para que se tuvieran en cuenta sus necesidades peculiares. Al ser libre el software resultante, no habría obligación de contratar las mejoras y las adaptaciones al mismo proveedor, de manera que se introduciría de esta forma competencia en ese mercado (que hoy por hoy es casi cautivo). Y en cualquier caso, el coste final para cualquiera de las administraciones implicadas no sería nunca mayor que si se hubiera utilizado un modelo propietario.

¿Son estos escenarios ciencia ficción? Como se verá más adelante, ya hay tímidas iniciativas en direcciones similares a las que se exponen en ellos. Además de ayudar a crear y mantener una industria en el ámbito de la Administración pública que realiza la compra, el software libre tiene más ventajas específicas en el entorno público. Por ejemplo, es la forma más efectiva de tener software desarrollado en lenguas minoritarias (preocupación esencial de muchas administraciones públicas). También puede ayudar mucho en el mantenimiento de la independencia estratégica a largo plazo y en asegurar el acceso a los datos que custodian las administraciones públicas dentro de mucho tiempo. Por todo ello, las entidades públicas están cada vez más interesadas en el software libre como usuarias.

Algunos casos relacionados con administraciones alemanas

En julio de 2003 se liberó la primera versión estable de Kolab, un producto del proyecto Kroupware. Kolab es un sistema libre de ayuda informática al trabajo en grupo (*groupware*) basado en KDE. El motivo por el que mencionamos este proyecto en este punto es porque su origen fue un concurso del Bundesamt für Sicherheit in der Informationstechnik (BSI) del Gobierno alemán (que podría traducirse como 'Agencia Federal de Seguridad en Tecnologías de la Información'). Este concurso pedía la provisión de una solución que interoperase con Windows y Outlook por un lado y GNU/Linux y KDE por otro. Entre las ofertas presentadas, ganó la propuesta conjuntamente por tres empresas, Erfrakon, Intevation y Klarälvdalens Datakonsult, que proponían una solución libre basada parcialmente en software ya desarrollado por el proyecto KDE, completado con desarrollos propios libres, que dieron lugar a Kolab.

En mayo de 2003 el Ayuntamiento de Múnich (Alemania) aprobó la migración a GNU/Linux y aplicaciones de ofimática libres de todos los ordenadores usados como escritorios, unos catorce mil. La decisión de hacerlo no fue sólo económica: también se tuvieron en cuenta aspectos estratégicos y cualitativos, según declararon sus responsables. En el exhaustivo estudio que se realizó previamente a la decisión, la solución finalmente elegida (GNU/Linux más OpenOffice, fundamentalmente) obtuvo 6.218 puntos (de un máximo de diez mil) frente a los poco más de cinco mil que obtuvo la solución "tradicional" basada en software de Microsoft.

En julio de 2003 se hizo público por parte del Koordinierungs-und Beratungsstelle der Bundesregierung für Informationstechnik in der Bundesverwaltung (KBSt), del Ministerio del Interior alemán, el documento *Leitfaden für die Migration von Basissoftwarekomponenten auf Server- und Arbeitsplatzsystemen* [107] ('Guía de migración para componentes software básicos en servidores y estaciones de trabajo'), que ofrece un conjunto de orientaciones sobre cómo pueden migrar a soluciones basadas en software libre las entidades

públicas alemanas. Estas orientaciones están diseñadas para que quien esté a cargo de la toma de decisiones pueda evaluar si conviene una migración a software libre y en qué condiciones debe hacerla si toma esa decisión.

6.2.2. Promoción de la sociedad de la información

Las entidades públicas dedican muchos recursos a la promoción de planes que incentivan el gasto en informática. ésta es una herramienta formidable, que puede ayudar mucho a la extensión de nuevas tecnologías por la sociedad. Pero también es una herramienta peligrosa. Por ejemplo, puede no ser muy buena idea promover el uso de Internet en la sociedad recomendando un determinado navegador que fomente la posición de monopolio *de facto* de una empresa, lo que a la larga podría ser perjudicial para la sociedad a la que se está tratando de beneficiar.

De nuevo, el software libre puede ayudar en estas situaciones. En primer lugar, es neutro frente a fabricantes, ya que ninguno tiene la exclusiva de ningún programa libre. Si una administración quiere promover el uso de una familia de programas libres, puede convocar concursos, a los que se puede presentar cualquier empresa del sector, para gestionar su entrega a los ciudadanos, su mejora o su ampliación con las funcionalidades que se desee, etc. En segundo lugar, puede ayudar mucho en los aspectos económicos. Por ejemplo, en muchos casos puede utilizarse la misma cantidad de recursos en adquirir una cierta cantidad de licencias de un programa propietario para su uso por los ciudadanos, que en adquirir una copia de uno libre y contratar soporte o adaptaciones para él; o incluso en negociar con un productor de software propietario la compra de los derechos de su producto para convertirlo en software libre.

En otro ámbito, podemos imaginar que parte de la cantidad destinada a un programa de informatización de escuelas se dedica a crear una distribución de GNU/Linux adaptada a las necesidades de la docencia en enseñanza primaria. Y con el resto de los recursos se contrata soporte para que el software sea mantenido en esas escuelas, de forma que no sea un simple "software florero", sino que realmente haya gente encargada de su correcto funcionamiento. De esta manera no sólo se cubren las necesidades del sistema educativo, sino que además se genera un mercado para empresas, habitualmente de ámbito local, capaces de ofrecer servicios de mantenimiento. Y por supuesto, se deja completamente abierto el camino al futuro: el software no quedará obsoleto en pocos años obligando a recomenzar desde cero, sino que se podrá ir actualizando incrementalmente, año tras año, manteniendo los beneficios del programa con una inversión similar.

Nota

El lector conocedor de las iniciativas públicas con respecto al software libre reconocerá en este ejemplo el caso de LinEx. A finales de 2001 la Junta de Extremadura (España) decidió utilizar una distribución de GNU/Linux para informatizar todos los colegios públicos de la región. Para ello, financió la construcción de LinEx, una distribución basada en Debian GNU/Linux que fue anunciada en primavera de 2002, y se encargó de que fuera un requisito en todos los concursos de adquisición de equipamiento informático para centros educativos. Además, inició programas de formación y capacitación de profesores, de creación de materiales docentes y de extensión de la experiencia a otros ámbitos. A mediados de 2003, la experiencia parece ser un éxito, que ya se está extendiendo institucionalmente a otras regiones (por ejemplo, a Andalucía, también en España, mediante el proyecto Guadalinex).

6.2.3. Fomento de la investigación

También en política de I+D el software libre tiene interesantes ventajas que merece la pena analizar. Con dinero público se están financiando los desarrollos de gran cantidad de software del que la sociedad no acaba beneficiándose ni siquiera indirectamente. Habitualmente, los programas públicos de fomento de la investigación y el desarrollo financian total o parcialmente proyectos que crean programas sin atender a qué derechos va a tener el público sobre ellos. En muchos casos los resultados, sin un plan adecuado de comercialización, simplemente quedan en algún cajón, cubiertos de polvo. En otros, las mismas personas que financiaron un programa mediante impuestos acaban pagándolo de nuevo si lo quieren usar (ya que tienen que adquirir licencias de uso).

El software libre ofrece una opción interesante, que poco a poco las autoridades encargadas de la política de innovación en muchas administraciones están considerando con atención. Especialmente cuando la investigación es precompetitiva (lo más habitual en los casos de financiación pública), el hecho de que los programas resultantes sean libres permite que la industria en su conjunto (y por ende la sociedad) se beneficie grandemente del dinero público gastado en I+D en el campo del software. Donde una empresa ve un resultado de imposible comercialización, otra puede ver una oportunidad de negocio. Así, por un lado, se maximizan los resultados de los programas de investigación, y por otro, se favorece la competencia entre las empresas que quieran utilizar los resultados de un proyecto, ya que todas ellas competirán a partir de los mismos programas resultado del proyecto.

Este modelo no es nuevo. En gran medida es el que ha permitido el desarrollo de Internet. Si las administraciones públicas exigen que los resultados de la investigación realizada con sus fondos sean distribuidos en forma de programas libres, no sería extraño que apareciesen, a diversas escalas, casos similares. O bien los resultados de esas investigaciones son malos o inútiles (y en ese caso debería replantearse la forma de selección de los proyectos), o bien la dinamización que supondría dejarlos listos para que cualquier empresa pudiera convertirlos en producto permitiría desarrollos sencillamente impredecibles.

6.3. Ejemplos de iniciativas legislativas

En los apartados siguientes se repasan algunas iniciativas legislativas concretas relacionadas con el uso y la promoción del software libre por parte de las administraciones públicas. Desde luego, la lista ofrecida no pretende ser exhaustiva, y se ha centrado en las iniciativas que han sido pioneras desde algún punto de vista (incluso si no han sido finalmente aprobadas). El lector interesado en completarla puede consultar "GrULIC. Legislación sobre el uso de software libre en el Estado" [133], donde se citan muchos más casos similares. Además, en un apéndice (apéndice D) se incluye con fines ilustrativos el texto íntegro, o las partes más relevantes, de varias de estas iniciativas.

6.3.1. Proyectos de ley en Francia

En 1999 y 2000 se presentaron en Francia dos proyectos de ley relacionados con el software libre que fueron los pioneros de una larga serie de debates legislativos sobre la materia:

- El Proyecto de Ley 1999-495, propuesto por Laffitte, Trégouet y Cabanel, fue expuesto en el servidor web del Senado de la República Francesa a partir de octubre de 1999. Tras un proceso de discusión pública por Internet (http://www.senat.fr/consult/loglibre/index.htm) [102] que se prolongó durante dos meses, el Proyecto sufrió algunas modificaciones. El resultado es el Proyecto de Ley número 2000-117 (Laffitte, Trégouet y Cabanel, Proposition de Loi numéro 117, Senado de la República Francesa, 2000) [162], que abogaba por el uso obligatorio de software libre en la Administración, previendo excepciones y medidas transitorias para aquellos casos en los que ello no fuera aún técnicamente posible, en un marco más general que intentaba generalizar en la Administración francesa el uso de Internet y del software libre.

- En abril de 2000 los diputados Jean-Yves Le Déaut, Christian Paul y Pierre Cohen propusieron una nueva ley cuyo objetivo era similar al proyecto de Laffitte, Trégouet y Cabanel: reforzar las libertades y la seguridad del consumidor, así como mejorar la igualdad de derechos en la sociedad de la información.

  Sin embargo, a diferencia del proyecto de ley de Laffitte, Trégouet y Cabanel, éste otro no forzaba a utilizar software libre en la Administración. Este proyecto de ley se centraba en que el software utilizado en la Administración tuviera el código fuente disponible, si bien no obligaba a que éste se distribuyera con licencias de software libre.

  Para conseguir sus objetivos los legisladores pretendían garantizar el "derecho a la compatibilidad" del software, proporcionando mecanismos que llevaran a la práctica el principio de interoperabilidad plasmado en la Directiva del Consejo Europeo relativa a la protección jurídica de programas

de ordenador (Directiva 91/250/CEE del Consejo, de 14 de mayo de 1991, relativa a la protección jurídica de programas de ordenador, 1991) [111].

Ninguno de los dos proyectos franceses se convirtió en ley, pero ambos han servido de inspiración a la mayoría de las iniciativas posteriores en todo el mundo, y por ello es especialmente interesante su estudio. El segundo (el propuesto por Le Déaut, Paul y Cohen) perseguía la compatibilidad y la interoperabilidad del software, haciendo hincapié en la disponibilidad del código fuente del software utilizado en la Administración. Sin embargo, no requería que las aplicaciones desarrolladas fueran software libre, entendido como aquél que se distribuye con licencias que garantizan la libertad de modificación, uso y redistribución del programa.

Más adelante (apartado D.1 y apartado D.2 del apéndice D) reproduciremos de manera casi íntegra el articulado y la exposición de motivos de ambos proyectos de ley. En particular, son de especial interés las exposiciones de motivos, que resaltan cuáles son los problemas que acechan hoy en día a las administraciones públicas en relación con el uso de software en general.

6.3.2. Proyecto de ley en Brasil

El diputado Walter Pinheiro presentó en diciembre de 1999 un proyecto de ley sobre el software libre en la Cámara Federal de Brasil (Proposição pl-2269/1999. Dispõe sobre a utilização de programas abertos pelos entes de direito público e de direito privado sob controle acionário da administração pública, Cámara de los Diputados de Brasil, diciembre de 1999) [185]. Este proyecto afectaba a la utilización de software libre en la Administración pública y en las empresas privadas controladas accionarialmente por el Estado.

En él se recomienda el uso de software libre en estas entidades que no tenga restricciones en cuanto a su préstamo, modificación o distribución. El articulado de la ley describe de manera pormenorizada qué se entiende por software libre y cómo deben ser las licencias que lo acompañen. Las definiciones coinciden con la definición clásica de software libre del proyecto GNU. En la exposición de motivos se repasa la historia del proyecto GNU, analizando sus ventajas y logros. Asimismo se hace referencia a la situación actual del software libre, utilizando como ejemplo el sistema operativo GNU/Linux.

Una de las partes más interesantes, el artículo primero, deja bien claro el ámbito en el que se propone el uso de software libre (teniendo en cuenta que la definición que se ofrece en los artículos posteriores para "programa abierto" es, como ya se ha dicho, software libre):

> "La Administración pública en todos los niveles, los poderes de la República, las empresas estatales y de economía mixta, las empresas públicas y todos los demás organismos públicos o privados sujetos al control de la sociedad brasileña están obligados a utilizar preferentemente, en sus sistemas y equipamientos de informática, programas abiertos, libres de restricciones propietarias en cuanto a su cesión, modificación y distribución."

6.3.3. Proyectos de ley en Perú

Son varios los proyectos de ley relacionados con el uso del software libre en las administraciones públicas que se han propuesto en Perú ("GNU Perú. Proyectos de ley sobre software libre en la Administración pública del gobierno peruano", Congreso de la República) [184]. El primero y el más conocido fue propuesto por el congresista Edgar Villanueva Núñez en diciembre de 2001 (Proyecto de Ley sobre software libre número 1609, diciembre de 2001) [222]. En él se define *software libre* según la definición clásica de las cuatro libertades (a la que se le da quizás una mayor precisión legal, con una definición que especifica seis características que ha de tener un programa libre) y se propone su utilización exclusiva en la Administración peruana:

> "Artículo 2. Los poderes ejecutivo, legislativo y judicial, los organismos descentralizados y las empresas donde el Estado posea mayoría accionaria, emplearán en sus sistemas y equipamientos de informática exclusivamente programas o software libres."

No obstante, un poco más adelante, los artículos 4 y 5 incluyen algunas excepciones a esta regla.

En su día esta propuesta de ley tuvo gran repercusión mundial. Por un lado, fue la primera vez que se propuso el uso exclusivo del software libre en una administración. Pero incluso más importante para su repercusión que esa novedad fue el intercambio epistolar entre el congresista Villanueva y la representación de Microsoft en Perú, que realizó alegaciones a esta propuesta. Esta propuesta de ley también es interesante por la posición que tomó al respecto la embajada de EE.UU., que incluso llegó a enviar por conducto oficial una notificación (adjuntando un informe elaborado por Microsoft) al Congreso peruano en la que expresaba su "preocupación sobre las recientes propuestas del Congreso de la República para restringir las compras por parte del Gobierno peruano de software de código abierto o software libre" ("Carta al presidente del Congreso de la República", 2002) [147]. Entre otros motivos, tanto las alegaciones de Microsoft como las de la embajada de EE.UU. trataban de mostrar cómo la propuesta de ley discriminaría a unas empresas frente a otras y haría imposible las inversiones necesarias para la creación de una industria nacional de creación de software. Ante esto, Villanueva argumentó que su proposición de ley no discriminaba ni favorecía de ninguna manera a ninguna empresa en particular, pues no especificaba quién podría ser proveedor de la Administración, sino cómo (en qué condiciones) tendría que realizarse la provisión de software. Esta argumentación es muy clara para entender cómo la promoción del software libre en la Administración no perjudica en ningún caso la libre competencia entre las empresas suministradoras.

Más adelante, los congresistas peruanos Edgar Villanueva Núñez y Jacques Rodrich Ackerman presentaron un nuevo proyecto de ley, el número 2485, de 8 de abril de 2002 (Proyecto de Ley de Uso de Software Libre en la Administración pública número 2485, 2002) [223], que en agosto de 2003 sigue su trámite parlamentario. Este proyecto es una evolución del Proyecto de Ley 1609 [222],

del cual recoge varios comentarios y mejoras, y puede considerarse como un buen ejemplo de proyecto de ley que propone el uso exclusivo de software libre en las administraciones públicas, salvo en ciertos casos excepcionales. Por su interés, incluimos su texto íntegro (apartado D.3 del apéndice D). En particular, su exposición de motivos es un buen resumen de las características que debería tener el software utilizado por las administraciones públicas y de cómo éstas las cumple mejor el software libre que el software propietario.

6.3.4. Proyectos de ley en España

En España ha habido varias iniciativas legislativas relacionadas con el software libre. Citamos a continuación algunas de ellas:

- Decreto de Medidas de Impulso de la Sociedad del Conocimiento en Andalucía
  Una de las iniciativas legislativas más importantes (por haber entrado en vigor) que han tenido lugar en España ha sido sin duda la adoptada por Andalucía. En el Decreto de Medidas de Impulso de la Sociedad del Conocimiento en Andalucía (Decreto 72/2003, de 18 de marzo, de Medidas de Impulso de la Sociedad del Conocimiento en Andalucía, 2003) [99] se trata del uso de software libre, fundamentalmente (pero no sólo) en el entorno educativo.
  Entre otros detalles, fomenta el uso de software libre de forma preferente en los centros docentes públicos, obliga a que todo el equipamiento adquirido para estos centros sea compatible con sistemas operativos libres, y lo mismo para los centros de la Junta que ofrezcan acceso público a Internet.

- Proposición de Ley de Software Libre en el marco de la Administración pública de Cataluña
  En otras comunidades se han discutido proposiciones más ambiciosas, pero sin que hayan logrado la mayoría necesaria. Entre ellas, la más conocida es probablemente la debatida en el Parlament de Catalunya (Proposició de llei de programari lliure en el marc de l'Administració pública de Catalunya, 2002) [221], muy similar a la que el mismo partido (Esquerra Republicana de Catalunya) presentó en el Congreso de los Diputados, de la que hablaremos a continuación. Esta propuesta no prosperó cuando fue sometida a votación.

- Proyecto de Ley de Puigcercós Boixassa en el Congreso de los Diputados
  También hubo una iniciativa en el Congreso de los Diputados propuesta por Joan Puigcercós Boixassa (Esquerra Republicana de Catalunya) (Proposición de Ley de Medidas para la Implantación del Software Libre en la Administración del Estado, 2002) [188]. Esta iniciativa propone la preferencia del uso de software libre en la Administración del Estado, y en ese sentido es similar a otras iniciativas con ese fin. Sin embargo, tiene la peculiaridad interesante de hacer énfasis en la disposición de los programas libres localizados para los idiomas cooficiales (en las comunidades

autónomas que los tienen). La iniciativa no consiguió ser aprobada en el trámite parlamentario.

7. Ingeniería del software libre

"The best way to have a good idea is to have many of them."

"La mejor forma de tener una buena idea es tener muchas."

Linus Pauling

En los capítulos anteriores se ha mostrado por qué el desafío del software libre no es el de un nuevo competidor que bajo las mismas reglas genera software de manera más rápida, más barata y de mejor calidad: el software libre se diferencia del software "tradicional" en aspectos más fundamentales, empezando por razones y motivaciones filosóficas, siguiendo por nuevas pautas económicas y de mercado, y finalizando con una forma diferente de generar software. La ingeniería del software no puede ser ajena a esto y desde hace poco más de un lustro ha ido profundizando en la investigación de todos estos aspectos. Este capítulo pretende mostrar los estudios más significativos y las evidencias que aportan con el objetivo de ofrecer al lector una visión del estado del arte y de las perspectivas de futuro de lo que hemos dado en llamar *la ingeniería del software libre*.

7.1. Introducción

Aunque hace ya varias décadas que se desarrolla software libre, sólo desde hace unos pocos años se ha empezado a prestar atención a sus modelos y procesos de desarrollo desde el punto de vista de la ingeniería del software. Igual que no hay un único modelo de desarrollo de software propietario, tampoco hay un único modelo de software libre[5], pero aun así pueden encontrarse interesantes características que comparten gran parte de los proyectos estudiados y que podrían estar enraizadas en las propiedades de los programas libres.

[5]En el artículo "The ecology of open source software development" (Kieran Healy y Alan Schussman, 2003) [140] se muestra la gran variedad de proyectos, así como su diversidad en cuanto al número de desarrolladores, al uso de herramientas y a las descargas.

En 1997 Eric S. Raymond publicó el primer artículo ampliamente difundido, "La catedral y el bazar" (*The cathedral and the bazaar. Musings on Linux and open source by an accidental revolutionary*, O'Reilly & Associates –http://www.ora.com–, 2001) [192], en el que se describían algunas características de los modelos de desarrollo de software libre, haciendo especial énfasis en lo que diferenciaba a estos modelos de los de desarrollo propietario. Este artículo se ha convertido desde entonces en uno de los más conocidos (y criticados) del mundo del software libre, y hasta cierto punto, en la señal de comienzo para el desarrollo de la ingeniería del software libre.

7.2. La catedral y el bazar

Raymond establece una analogía entre la forma de construir las catedrales medievales y la manera clásica de producir software. Argumenta que en ambos casos existe una clara distribución de tareas y funciones, en la que destaca la existencia de un diseñador que está por encima de todo y que ha de controlar en todo momento el desarrollo de la actividad. Asimismo, la planificación está estrictamente controlada, lo que da lugar a unos procesos claramente detallados en los que idealmente cada participante en la actividad tiene un papel específico muy delimitado.

Dentro de lo que Raymond toma como el modelo de creación de catedrales no sólo tienen cabida los procesos pesados que podemos encontrar en la industria del software (el modelo en cascada clásico, las diferentes vertientes del Rational Unified Process, etc.), sino también proyectos de software libre, como es el caso de GNU y NetBSD. Para Raymond, estos proyectos se encuentran fuertemente centralizados, ya que unas pocas personas son las que realizan el diseño y la implementación del software. Las tareas que desempeñan estas personas, así como sus funciones, están perfectamente definidas, y alguien que quisiera entrar a formar parte del equipo de desarrollo necesitaría que se le asignara una tarea y una función según las necesidades del proyecto. Por otra parte, las entregas de este tipo de programas se encuentran espaciadas en el tiempo a partir de una planificación bastante estricta. Esto supone tener pocas entregas del software y ciclos largos, que constan de varias etapas, entre las entregas.

El modelo antagónico al de la catedral es el bazar. Según Raymond, algunos de los programas de software libre, en especial el núcleo Linux, se han desarrollado siguiendo un esquema similar al de un bazar oriental. En un bazar no existe una máxima autoridad que controle los procesos que se van desarrollando ni que planifique estrictamente lo que ha de suceder. Por otro lado, los roles de los participantes pueden cambiar de manera continua (los vendedores pueden pasar a ser clientes) y sin indicación externa.

Pero lo más novedoso de "La catedral y el bazar" es la descripción del proceso que ha hecho de Linux un éxito dentro del mundo del software libre; es una sucesión de "buenas maneras" para aprovechar al máximo las posibilidades que ofrece la disponibilidad de código fuente y la interactividad mediante el uso de sistemas y herramientas telemáticos.

Un proyecto de software libre suele surgir a raíz de una acción puramente personal; la causa se ha de buscar en un desarrollador que vea limitada su capacidad de resolver un problema. El desarrollador ha de tener los conocimientos necesarios para, por lo menos, empezar a resolverlo. Una vez que haya conseguido tener algo utilizable, con algo de funcionalidad, sencillo y, a ser posible, bien diseñado o escrito, lo mejor que puede hacer es compartir esa solución con la comunidad del software libre. Es lo que se denomina *publicación tem-*

prana (*release early*, en inglés), que permite llamar la atención de otras personas (generalmente desarrolladores) que tengan el mismo problema y que puedan estar interesados en la solución.

Uno de los principios básicos de este modelo de desarrollo es considerar a los usuarios como codesarrolladores. Se los ha de tratar con mimo, no sólo porque pueden proporcionar publicidad mediante el "boca a boca", sino también por el hecho de que realizarán una de las tareas más costosas que existen en la generación de software: las pruebas. Al contrario que el codesarrollo, que es difícilmente escalable, la depuración y las pruebas tienen la propiedad de ser altamente paralelizables. El usuario será el que tome el software y lo pruebe en su máquina bajo unas condiciones específicas (una arquitectura, unas herramientas, etc.), una tarea que multiplicada por un gran número de arquitecturas y entornos supondría un gran esfuerzo para el equipo de desarrollo.

Si se trata a los usuarios como codesarrolladores puede darse el caso de que alguno de ellos encuentre un error y lo resuelva enviando un parche al desarrollador del proyecto para que el problema esté solucionado en la siguiente versión. También puede suceder, por ejemplo, que una persona diferente de la que descubre un error sea la que finalmente lo entienda y lo corrija. En cualquier caso, todas estas circunstancias son muy provechosas para el desarrollo del software, por lo que es muy beneficioso entrar en una dinámica de esta índole.

Esta situación se torna más efectiva con entregas frecuentes, ya que la motivación para encontrar, notificar y corregir errores es alta debido a que se supone que se va a ser atendido inmediatamente. Además se consiguen ventajas secundarias, como el hecho de que al integrar frecuentemente –idealmente una o más veces al día– no haga falta una última fase de integración de los módulos que componen el programa. Esto se ha denominado *publicación frecuente* (que en la terminología anglosajona se conoce como *release often*) y posibilita una gran modularidad (Alessandro Narduzzo y Alessandro Rossi, "Modularity in action: GNU/Linux and free/open source software development model unleashed", mayo 2003) [176], a la vez que maximiza el efecto propagandístico que tiene la publicación de una nueva versión del software.

Nota

La gestión de nuevas versiones depende, lógicamente, del tamaño del proyecto, ya que los problemas a los que hay que enfrentarse no son iguales cuando el equipo de desarrollo tiene dos componentes que cuando tiene cientos. Mientras que, en general, en los proyectos pequeños este proceso es más bien informal, la gestión de entregas en los proyectos grandes suele seguir un proceso definido no exento de cierta complejidad. Hay un artículo titulado "Release management within open source projects" (Justin R. Ehrenkrantz, 2003) [110] que describe detalladamente la secuencia que se sigue en el servidor web Apache, el núcleo Linux y el sistema de versiones Subversion.

Para evitar que la publicación frecuente asuste a usuarios que prioricen la estabilidad sobre la rapidez con la que el software evoluciona, algunos proyectos de software libre cuentan con varias ramas de desarrollo en paralelo. El caso

más conocido es el del núcleo Linux, que tiene una rama estable –dirigida a aquellas personas que estiman su fiabilidad– y otra inestable –pensada para desarrolladores– con las últimas innovaciones y novedades.

7.3. Liderazgo y toma de decisiones en el bazar

Raymond supone que todo proyecto de software libre ha de contar con un *dictador benevolente*, una especie de líder –que generalmente coincide con el fundador del proyecto– que ha de guiar el proyecto y que se reserva siempre la última palabra en la toma de decisiones. Las habilidades que ha de tener esta persona son principalmente las de saber motivar y coordinar un proyecto, entender a los usuarios y codesarrolladores, buscar consensos e integrar a todo aquél que pueda aportar algo. Como se puede observar, entre los requisitos más importantes no se ha mencionado el que sea técnicamente competente, aunque nunca esté de más.

Con el crecimiento en tamaño y en número de desarrolladores de algunos proyectos de software libre, han ido apareciendo nuevas formas de organizar la toma de decisiones. Linux, por ejemplo, dispone de una estructura jerárquica basada en la delegación de responsabilidades por parte de Linus Torvalds, el "dictador benevolente". Así, vemos cómo hay partes de Linux que cuentan con sus propios "dictadores benevolentes", aunque éstos vean limitado su "poder" por el hecho de que Linus Torvalds sea el que decida en último término. Este caso es un claro ejemplo de cómo la alta modularidad existente en un proyecto de software libre ha propiciado una forma de organización y de toma de decisiones específica (Alessandro Narduzzo y Alessandro Rossi, "Modularity in action: GNU/Linux and free/open source software development model unleashed", 2003) [176].

Nota

Hay quien dice que la forma de organizarse dentro de los proyectos de software libre se asemeja a la de un equipo quirúrgico, como propuso Harlan Mills (de IBM) a principios de la década de los setenta y popularizó Brooks en su famoso libro *The mythical man-month* (Frederick P. Brooks Jr., 1975) [150]. Aunque se pueden dar casos en los que el equipo de desarrollo de alguna aplicación de software libre esté formado por un diseñador/desarrollador principal (el cirujano) y por muchos codesarrolladores que realizan tareas auxiliares (administración de sistemas, mantenimiento, tareas especializadas, documentación...), no existe en ningún caso una separación tan estricta y definida como la propuesta por Mills y Brooks. De todas formas, como bien apunta Brooks en el caso del equipo quirúrgico, en el software libre el número de desarrolladores que tienen que comunicarse para crear un sistema grande y complejo es más reducido que el número total de desarrolladores.

En el caso de la Fundación Apache nos encontramos con una *meritocracia*, ya que dicha institución cuenta con un comité de directores formado por personas que han contribuido de manera notable al proyecto. En realidad, no se trata de una meritocracia rigurosa en la que los que más contribuyen son los que gobiernan, ya que el comité de directores es elegido democrática y periódicamente por los miembros de la Fundación (que se encarga de gestionar varios proyectos de software libre, como Apache, Jakarta, etc.). Para llegar a

ser miembro de la Fundación Apache, se ha de haber aportado de forma continuada e importante a uno o varios proyectos de la Fundación. Este sistema también es utilizado en otros proyectos grandes, como FreeBSD o GNOME.

Otro caso interesante de organización formal es el GCC Steering Committee. Fue creado en 1998 para evitar que nadie se hiciera con el control del proyecto GCC (GNU Compiler Collection, el sistema de compilación de GNU) y refrendado por la FSF (promotora del proyecto GNU) pocos meses después. En cierto sentido es un comité continuador de la tradición de uno similar que había habido en el proyecto EGCS (que durante un tiempo fue paralelo al proyecto GCC, pero que más tarde se unió a éste). Su misión fundamental es asegurarse de que el proyecto GCC respeta la declaración de objetivos (*mission statement*) del proyecto. Los miembros del comité lo son a título personal, y son elegidos por el propio proyecto de forma que representen, con cierta fidelidad, a las diferentes comunidades que colaboran en el desarrollo de GCC (desarrolladores de soporte para diversos lenguajes, desarrolladores relacionados con el núcleo, grupos interesados en programación empotrada, etc.).

El liderazgo de un proyecto de software libre por parte de una misma persona no tiene por qué ser eterno. Se pueden dar básicamente dos circunstancias por las que un líder de proyecto deje de serlo. La primera de ellas es la falta de interés, tiempo o motivación para seguir adelante. En ese caso, se ha de pasar el testigo a otro desarrollador que asuma la función de líder del proyecto. Estudios recientes (Jesús M. González Barahona y Gregorio Robles, 2003) [87] demuestran que, en general, los proyectos suelen tener un liderazgo que cambia de manos con frecuencia, de manera que se pueden observar varias *generaciones* de desarrolladores en el tiempo. El segundo caso es más problemático: se trata de una bifurcación. Las licencias de software libre permiten tomar el código, modificarlo y redistribuirlo por cualquier persona sin que haga falta el visto bueno del líder del proyecto. Esto no se suele dar generalmente, salvo en los casos en los que se haga con el fin de evitar de manera deliberada precisamente al líder del proyecto (y un posible veto suyo a la aportación). Estamos ciertamente ante una especie de "golpe de estado", por otro lado totalmente lícito y legítimo. Por ello uno de los objetivos que busca un líder de proyecto al mantener satisfechos a sus codesarrolladores es minimizar la posibilidad de que exista una bifurcación.

7.4. Procesos en el software libre

Aunque el software libre no está necesariamente asociado con un proceso de desarrollo software específico, existe un amplio consenso sobre los procesos más comunes que se utilizan. Esto no quiere decir que no existan proyectos de software libre que hayan sido creados utilizando procesos clásicos, como el modelo en cascada. Generalmente, el modelo de desarrollo en proyectos de

software libre suele ser más informal, debido a que gran parte del equipo de desarrollo realiza esas tareas de manera voluntaria y sin recompensa económica, al menos directa, a cambio.

La forma en la que se capturan requisitos en el mundo del software libre depende tanto de la "edad" como del tamaño del proyecto. En las primeras etapas, el fundador del proyecto y el usuario suelen coincidir en una misma persona. Más adelante, y si el proyecto crece, la captura de requisitos suele tener lugar por medio de las listas de correo electrónico y se suele llegar a una clara distinción entre el equipo de desarrollo –o al menos, los desarrolladores más activos– y los usuarios. Para proyectos grandes, con muchos usuarios y muchos desarrolladores, la captura de requisitos se hace mediante la misma herramienta que se utiliza para gestionar los errores del proyecto. En este caso, en vez de hablar de errores, se refieren a actividades, aunque el mecanismo utilizado para su gestión es idéntico al de la corrección de errores (se las clasificará según importancia, dependencia, etc., y se podrá monitorizar si han sido resueltas o no). El uso de esta herramienta para la planificación es más bien reciente, por lo que se puede observar que en el mundo del software libre existe una cierta evolución desde la carencia absoluta hasta un sistema centralizado de gestión ingenieril de actividades, aunque indudablemente éste sea bastante limitado. En cualquier caso, no suele ser común un documento que recoja los requisitos, tal como es normal en el modelo en cascada.

En cuanto al diseño global del sistema, sólo los grandes proyectos suelen tenerlo documentado de manera exhaustiva. En el resto, lo más probable es que el o los desarrolladores principales sean los únicos que lo posean –a veces sólo en su mente– o que vaya fraguándose en versiones posteriores del software. La carencia de un diseño detallado no sólo implica limitaciones en cuanto a la posible reutilización de módulos, sino que también es un gran obstáculo a la hora de permitir el acceso a nuevos desarrolladores, ya que éstos se tendrán que enfrentar a un proceso de aprendizaje lento y costoso. El diseño detallado, por su parte, tampoco está muy generalizado. Su ausencia implica que se pierdan muchas posibilidades de reutilización de código.

La implementación es la fase en la que se concentra el mayor esfuerzo por parte de los desarrolladores de software libre, entre otras razones porque a sus ojos es manifiestamente la más divertida. Para ello se suele seguir el paradigma de programación clásico de prueba y error hasta que se consiguen los resultados deseados desde el punto de vista subjetivo del programador. Históricamente, raro es el caso en el que se han incluido pruebas unitarias conjuntamente con el código, aun cuando facilitarían la modificación o la inclusión de código posterior por parte de otros desarrolladores. En el caso de algunos proyectos grandes, como por ejemplo Mozilla, existen máquinas dedicadas exclusivamente a descargarse los repositorios que contienen el código más reciente

para compilarlo para diferentes arquitecturas ("An overview of the software engineering process and tools in the Mozilla project", 2002) [193]. Los errores detectados se notifican a una lista de correo de desarrolladores.

Sin embargo, las pruebas automáticas no están muy arraigadas. Por lo general, serán los propios usuarios, dentro de la gran variedad de usos, arquitecturas y combinaciones, los que las realizarán. Esto tiene la ventaja de que se paralelizan a un coste mínimo para el equipo de desarrollo. El problema que plantea este modelo es cómo organizarse para que la realimentación por parte de los usuarios exista y sea lo más eficiente posible.

En cuanto al mantenimiento del software en el mundo del software libre –entendiéndolo como el mantenimiento de versiones antiguas–, ésta es una tarea cuya existencia depende del proyecto. En proyectos donde se requiere una gran estabilidad, como por ejemplo núcleos del sistema operativo, se mantienen versiones antiguas, ya que un cambio a una nueva versión puede resultar traumático. Pero por lo general, en la mayoría de los proyectos de software libre, si se tiene una versión antigua y se encuentra un error, los desarrolladores no suelen ponerse a corregirlo, sino que aconsejan utilizar la versión más moderna con la esperanza de que ese error haya desaparecido por el hecho de que el software ha evolucionado.

7.5. Crítica a "La catedral y el bazar"

"La catedral y el bazar" adolece de una falta de sistematicidad y rigor acorde con su naturaleza más bien ensayística y ciertamente poco científica. Las críticas más frecuentes se refieren a que explica básicamente una experiencia puntual –el caso Linux– y que se pretenden generalizar las conclusiones para todos los proyectos de software libre. En este sentido, en "Cave or community? An empirical examination of 100 mature open source projects" [160] se puede ver que la existencia de una comunidad tan amplia como la comunidad con la que cuenta el núcleo Linux es más bien una excepción.

Todavía más críticos se muestran aquéllos que piensan que Linux es un ejemplo de desarrollo que sigue el modelo de desarrollo de las catedrales. Argumentan que parece evidente que existe una cabeza pensante, o al menos una persona que tiene la máxima potestad, y un sistema jerárquico mediante delegación de responsabilidades hasta llegar a los obreros/programadores. Asimismo, existe reparto de tareas, aunque sea de manera implícita. En "A second look at the cathedral and the bazaar" [91] se va más allá y se sostiene –no sin cierta acritud y arrogancia en la argumentación– que la metáfora del bazar es internamente contradictoria.

Otro de los puntos más criticados de "La catedral y el bazar" es su afirmación de que la ley de Brooks, que dice que "agregar desarrolladores a un proyecto de software retrasado lo retrasa aún más" (*The mythical man-month. Essays on software engineering*, 1975) [150], no es válida en el mundo del software libre.

En [148] se puede leer cómo en realidad lo que pasa es que las condiciones de entorno son diferentes y lo que en un principio aparenta ser un incongruencia con la ley de Brooks, tras un estudio más exhaustivo, no deja de ser un espejismo.

7.6. Estudios cuantitativos

El software libre permite ahondar en el estudio cuantitativo del código y del resto de parámetros que intervienen en su generación, dada la accesibilidad a los mismos. Esto permite que áreas de la ingeniería del software tradicional –como la ingeniería del software empírica– se puedan ver potenciadas debido a la existencia de una ingente cantidad de información a la que se puede acceder sin necesidad de una fuerte intromisión en el desarrollo de software libre. Los autores estamos convencidos de que esta visión puede ayudar enormemente en el análisis y en la comprensión de los fenómenos ligados a la generación de software libre (y de software en general), y que se puede llegar incluso, entre otras posibilidades, a tener modelos predictivos del software que se realimenten en tiempo real.

La idea que hay detrás es muy simple: "dado que tenemos la posibilidad de estudiar la evolución de un número inmenso de proyectos de software libre, hagámoslo." Y es que además del estado actual de un proyecto, toda su evolución en el pasado es pública, por lo que toda esta información, debidamente extraída, analizada y empaquetada, puede servir como una base de conocimiento que permita evaluar la *salud* de un proyecto, facilitar la toma de decisiones y pronosticar complicaciones actuales y futuras.

El primer estudio cuantitativo de cierta importancia en el mundo del software libre data de 1998, aunque fue publicado a principios de 2000 ("The orbiten free software survey") [127]. Su finalidad era conocer de manera empírica la participación de los desarrolladores en el mundo del software libre. Para ello se valían del tratamiento estadístico de la asignación de autoría que los autores suelen situar en la cabeza de los ficheros de código fuente. Se demostró que la participación sigue la ley de Pareto ("Course of Political Economy", Lausana, 1896) [182]: el 80% del código corresponde al 20% más activo de los desarrolladores, mientras que el 80% restante de desarrolladores tiene una aportación de un 20% del total. Muchos estudios posteriores han confirmado y ampliado a otras formas de participación diferentes de la aportación de código fuente (mensajes a lista de correo, notificación de errores o incluso el número de descargas, como se pueden ver en http://www-mmd.eng.cam.ac.uk/people/fhh10/Sourceforge/Sourceforge%20paper.pdf [145]) la validez de este resultado.

Nota

El hecho de que aparezcan muchos términos de las ciencias económicas en el estudio ingenieril del software libre es consecuencia del interés que algunos economistas han mostrado en los últimos tiempos en conocer y entender los procesos que llevan a voluntarios a producir bienes de gran valor sin obtener generalmente beneficio directo a cambio. El

artículo más conocido es "Cooking pot markets: an economic model for the trade in free goods and services on the Internet" [125], en el que se introduce la *economía del regalo* en Internet. En http://www.wikipedia.org/wiki/Pareto [232] se puede obtener más información sobre el principio de Pareto y su generalización en la distribución de Pareto. Asimismo, son interesantes la curva de Lorenz (http://www.wikipedia.org/wiki/Lorenz_curve) [231], que representa de manera gráfica la participación en el proyecto de los desarrolladores, y el coeficiente de Gini (http://www.wikipedia.org/wiki/Gini_coefficient) [230], que se calcula a partir de la curva de Lorenz y del que resulta un número a partir del cual se puede ver la desigualdad en el sistema.

La herramienta que se utilizó para la realización de este estudio fue publicada con una licencia libre por los autores, por lo que se ofrece tanto la posibilidad de reproducir sus resultados como de efectuar nuevos estudios.

En un estudio posterior, Koch ("Results from software engineering research into open source development projects using public data", 2000) [158] fue más allá y también analizó las interacciones que se llevan a cabo en un proyecto de software libre. La fuente de información eran las listas de correo y el repositorio de versiones del proyecto GNOME. Pero el aspecto más interesante del estudio de Koch es el análisis económico. Koch se centra en comprobar la validez de predicción de costes clásicos (puntos de función, modelo de COCOMO...) y muestra los problemas que su aplicación conlleva, aunque ciertamente admite que los resultados que obtiene –tomados con ciertas precauciones– se ajustan parcialmente a la realidad. Concluye que el software libre necesita métodos de estudio y modelos propios, ya que los conocidos no se adaptan a su naturaleza. Sin embargo, resulta evidente que la posibilidad de obtener públicamente muchos de los datos relacionados con el desarrollo del software libre permite ser optimistas de cara a la consecución de éstos en un futuro no muy lejano. El de Koch se puede considerar el primer estudio cuantitativo completo, aun cuando ciertamente adolece de una metodología clara y, sobre todo, de unas herramientas *ad hoc* que permitieran comprobar sus resultados y el estudio de otros proyectos.

Mockus *et al*. presentaron en el año 2000 el primer estudio de proyectos de software libre que englobaba la descripción completa del proceso de desarrollo y de las estructuras organizativas, incluyendo evidencias tanto cualitativas como cuantitativas ("What is the context of charityware?") [172]. Utilizan para tal fin la historia de cambios del software, así como de los informes de error, para cuantificar aspectos de la participación de desarrolladores, el tamaño del núcleo, la autoría de código, la productividad, la densidad de defectos y los intervalos de resolución de errores. En cierto sentido, este estudio no deja de ser un estudio de ingeniería de software clásico, con la salvedad de que la obtención de los datos ha sido realizada íntegramente mediante la inspección semiautomática de los datos que los proyectos ofrecen públicamente en la Red. Al igual que en "Results from software engineering research into open source development projects using public data", 2000 [158], con este artículo no se proporcionó ninguna herramienta ni proceso automático que permitiera ser reutilizada en el futuro por otros equipos de investigación.

En "Estimating Linux's size", 2000 [227], y "More than a gigabuck: estimating GNU/Linux's" [228] encontramos el análisis cuantitativo de las líneas de código y los lenguajes de programación utilizados en la distribución Red Hat. González Barahona *et al.* han seguido sus pasos en una serie de artículos dedicados a la distribución Debian (*vid.* por ejemplo "Anatomy of two GNU/Linux distributions" [88]). Todos ellos ofrecen una especie de *radiografía* de estas dos populares distribuciones de GNU/Linux realizadas a partir de los datos que aporta una herramienta que cuenta el número de líneas físicas (las líneas de código que no son ni líneas en blanco ni comentarios) de un programa. Aparte del espectacular resultado en líneas de código totales (la versión de Debian 3.0 –conocida como Woody–, supera los cien millones de líneas de código), se puede ver la distribución del número de líneas en cada lenguaje de programación. La posibilidad de estudiar la evolución de las diferentes versiones de Debian en el tiempo aporta algunas evidencias interesantes [88]. Cabe mencionar que el tamaño medio de los paquetes permanece prácticamente constante a lo largo de los últimos cinco años, por lo que su tendencia natural a ir creciendo se ve neutralizada por la inclusión de paquetes más pequeños. Por otro lado, se puede percibir cómo la importancia del lenguaje de programación C, todavía predominante, decrece con el tiempo, mientras que lenguajes de guión (Python, PHP y Perl) y Java contabilizan un crecimiento explosivo. Los lenguajes compilados "clásicos" (Pascal, Ada, Modula...) están en clara recesión. Finalmente, estos artículos incluyen un apartado dedicado a mostrar los resultados obtenidos si se aplica el modelo COCOMO –un modelo de estimación de esfuerzos clásico que data de principios de la década de los ochenta (*Software Engineering Economics*, 1981) [93] y que se utiliza en proyectos de software propietario– para realizar una estimación de esfuerzo, duración del proyecto y costes.

Aunque precursores, la mayoría de los estudios presentados en este apartado están bastante limitados a los proyectos que analizan. La metodología que se ha usado se ajusta al proyecto en estudio, es parcialmente manual y en contadas ocasiones la parte automatizada puede ser utilizada de forma general con el resto de proyectos de software libre. Esto supone que el esfuerzo que hay que realizar para estudiar un nuevo proyecto es muy grande, ya que se ha de volver a adaptar el método utilizado y repetir las acciones manuales que se han llevado a cabo.

Por eso, los últimos esfuerzos ("Studying the evolution of libre software projects using publicly available data", en: *Proceedings* del 3rd Workshop on Open Source Software Engineering, 25th International Conference on Software Engineering, Portland, EE.UU. [196] o "Automating the measurement of open source projects", 2003 [124]) se centran en crear una infraestructura de análisis que integre varias herramientas de manera que se automatice el proceso al máximo. Existen dos motivaciones bastante evidentes para hacer esto: la primera es que una vez que se ha invertido mucho tiempo y esfuerzo en crear una herramienta para analizar un proyecto –haciendo especial hincapié en que sea genérica–, utilizarla para otros proyectos de software libre implica un esfuer-

zo mínimo. Por otro lado, el análisis mediante una serie de herramientas que analizan los programas desde diferentes puntos de vista –a veces complementarios, otras veces no– permite obtener una mayor perspectiva del proyecto. En Libre Software Engineering Web Site [86] se pueden seguir con mayor detenimiento estas iniciativas.

7.7. Trabajo futuro

Después de presentar la breve pero intensa historia de la ingeniería del software aplicada al software libre, podemos afirmar que ésta se encuentra todavía dando sus primeros pasos. Quedan varios aspectos de gran importancia pendientes de estudio y examen minuciosos para dar con un modelo que explique, al menos en parte, cómo se genera software libre. Las cuestiones que se han de afrontar en un futuro próximo son la clasificación de los proyectos de software libre, la creación de una metodología basada en lo posible en elementos de análisis automáticos y la utilización de los conocimientos adquiridos para la realización de modelos que permitan entender el desarrollo de software libre a la vez que faciliten la toma de decisiones a partir de la experiencia adquirida.

Un aspecto que tampoco se debe dejar de lado y que ya se está abordando en la actualidad es la evaluación de la validez de los métodos de ingeniería clásica en el campo del software libre en todas las intensificaciones de la ingeniería del software. Así, por ejemplo, las leyes de la evolución del software postuladas por Lehman ("Metrics and laws of software evolution - the nineties view" [165]) a principios de la década de los setenta y actualizadas y aumentadas en los ochenta y los noventa parece que no se cumplen incondicionalmente en el desarrollo de algunos proyectos de software libre ("Understanding open source software evolution: applying, breaking and rethinking the laws of software evolution", 2003 [199]).

En la actualidad, una de las deficiencias más importantes es la falta de una clasificación estricta del software libre que permita catalogar los proyectos en diferentes categorías. A día de hoy, los criterios de clasificación son demasiado burdos, de manera que se meten en el mismo saco proyectos cuyas características organizativas, técnicas o de cualquier otra índole son muy dispares. El argumento de que Linux, con una amplia comunidad y un gran número de desarrolladores, tiene una naturaleza diferente y no se comporta de idéntica manera que un proyecto mucho más limitado en número de desarrolladores y usuarios, es muy acertado. En cualquier caso, una clasificación más exhaustiva permitiría reutilizar la experiencia adquirida en proyectos similares (es decir, con características parecidas), facilitaría las predicciones, posibilitaría pronosticar riesgos, etc.

El segundo aspecto importante al que debe enfrentarse la ingeniería del software libre, muy ligado al punto anterior y a las tendencias actuales, es la creación de una metodología y de herramientas que la soporten. Una metodología clara y concisa permitirá estudiar todos los proyectos con el mismo rasero,

averiguar su estado actual, conocer su evolución y, por supuesto, clasificarlos. Las herramientas son esenciales a la hora de abordar este problema, ya que, una vez creadas, permiten tener al alcance de la mano el análisis de miles de proyectos con un mínimo esfuerzo adicional. Uno de los objetivos de la ingeniería del software libre es que a partir de un número limitado de parámetros que indiquen dónde encontrar en la Red información sobre el proyecto (la dirección del repositorio de versiones del software, el lugar donde se almacenan los archivos de las listas de correo electrónico, la localización del sistema de gestión de errores y una mínima encuesta) se pueda realizar un estudio exhaustivo del mismo. A los gestores del proyecto sólo les separaría un botón de un análisis completo, una especie de *análisis clínico* que permitiera juzgar la *salud* del proyecto y que a la vez incluyese una indicación sobre los aspectos que necesitan ser mejorados.

Una vez que se consigan métodos, clasificación y modelos, las posibilidades que ofrece la simulación y, para ser más concretos, los agentes inteligentes, pueden ser enormes. Habida cuenta de que partimos de un sistema cuya complejidad es notoria, resulta interesante la creación de modelos dinámicos en los que los diferentes entes que participan en la generación de software puedan ser modelados. Evidentemente cuanto más sepamos de los diferentes elementos, más ajustado a la realidad será el modelo. Aunque se conocen varias propuestas de simulaciones para el software libre, éstas son bastante simples e incompletas. En cierta medida, esto se debe a que todavía existe una gran laguna de conocimiento con respecto a las interacciones que tienen lugar en la generación de software libre. Si se consigue empaquetar y procesar convenientemente la información de los proyectos a lo largo de toda su historia, los agentes pueden ser cruciales para conocer cómo será la evolución en el futuro. Aunque existen muchas propuestas sobre cómo se ha de enfocar este problema, una de las más avanzadas por ahora se puede encontrar en http://wwwai.wu-wien.ac.at/~koch/oss-book/ [82].

7.8. Resumen

En resumen, en este capítulo hemos podido ver cómo la ingeniería del software libre es un campo joven y todavía por explorar. Sus primeros pasos se deben a escritos ensayísticos en los que se proponía –no sin cierta falta de rigor científico– un modelo de desarrollo más eficiente, pero paulatinamente se ha ido progresando en el estudio sistemático del software libre desde una óptica ingenieril. En la actualidad, tras varios años de informes y estudios de proyectos libres cuantitativos y cualitativos completos, se está llevando a cabo un enorme esfuerzo en la consecución de una infraestructura global que permita la clasificación, el análisis y la modelización de los proyectos en un espacio de tiempo limitado y de manera parcialmente automática. Cuando el análisis de los proyectos de software libre deje de ser tan costoso en tiempo y en esfuerzo como lo es hoy en día, es muy probable que se abra una nueva etapa en la

ingeniería del software libre en la que entren en escena otro tipo de técnicas cuyo propósito principal se sitúe en torno a la predicción de la evolución del software y al pronóstico de posibles complicaciones.

8. Entornos y tecnologías de desarrollo

> "The tools we use have a profound (and devious!) influence on our thinking habits, and, therefore, on our thinking abilities."
>
> "Las herramientas que usamos tienen una influencia profunda (¡y tortuosa!) sobre nuestros hábitos de pensamiento y, por tanto, sobre nuestras habilidades de pensamiento."
>
> Edsger W. Dijkstra, "How do we tell truths that might hurt?"

Los proyectos de software libre han ido creando a lo largo de los años sus propias herramientas y sistemas (también libres) para la ayuda en el proceso de desarrollo. Aunque cada proyecto sigue sus propias reglas y usa su propio conjunto de herramientas, hay ciertas prácticas, entornos y tecnologías que pueden considerarse como habituales en el mundo del desarrollo de software libre. En este capítulo trataremos los más comunes y comentaremos su impacto en la gestión y la evolución de los proyectos.

8.1. Caracterización de entornos, herramientas y sistemas

Antes de explicar herramientas concretas, vamos a definir las características y las propiedades generales que éstas tienen en función del trabajo que se ha de llevar a cabo y del modo de organización de los desarrolladores.

En primer lugar, aunque no necesariamente determinante, es habitual que se trate de que el entorno, las herramientas de desarrollo (e incluso la máquina virtual objetivo, cuando la hay), sea también *libre*. Esto no siempre ha sido así. Por ejemplo, el proyecto GNU, con el objetivo de reemplazar a Unix, tuvo que desarrollarse sobre y para sistemas Unix propietarios hasta la aparición de los Linux y los BSD libres. Hoy en día, especialmente cuando el software libre se desarrolla como parte de un modelo de negocio, se tiende a que la máquina objetivo pueda también ser un sistema propietario, a menudo mediante máquinas virtuales interpuestas (Java, Python, PHP, etc.). En cualquier caso, el entorno y la máquina virtual han de estar lo bastante difundidos y ser lo bastante económicos como para poder reunir a suficientes codesarrolladores que dispongan de esas herramientas.

En segundo lugar, también para atraer al mayor número posible de codesarrolladores, las herramientas deben ser *sencillas*, ampliamente *conocidas* y capaces de funcionar en máquinas *económicas*. Posiblemente por estas razones el mundo del software libre es relativamente conservador en lenguajes, herramientas y entornos.

En tercer lugar, el modelo de desarrollo de software libre suele ser eminentemente distribuido, con muchos colaboradores potenciales repartidos por todo el mundo. Por ello son precisas herramientas de colaboración, generalmente

asíncronas, que permitan además que el desarrollo avance con facilidad, independientemente de la cantidad y el ritmo de trabajo de cada colaborador, sin retrasar a nadie.

Finalmente, es conveniente proporcionar a los desarrolladores recursos de los que carecen, como máquinas de arquitecturas diversas en las que puedan compilar y probar sus programas.

8.2. Lenguajes y herramientas asociadas

La mayoría del software libre está realizado en lenguaje C, no sólo porque C es el lenguaje natural de toda variante de Unix (plataforma habitual del software libre), sino también por su amplia difusión, tanto en las mentes como en las máquinas (GCC es un compilador estándar instalado por defecto en casi todas las distribuciones). Precisamente por estas razones y por su eficiencia, lo recomienda Stallman en los proyectos de GNU ("GNU coding standards") [203]. Otros lenguajes que se le acercan bastante son C++, también soportado por GCC por defecto, y Java, que presenta cierta semejanza y que es popular porque permite desarrollar para máquinas virtuales disponibles en gran variedad de plataformas. Generalmente no se tienen en cuenta razones de ingeniería software: en SourceForge (*vid.* apartado 8.9.1), en el año 2004, por cada ciento sesenta proyectos en C había uno en Ada, lenguaje supuestamente más apropiado para desarrollar programas de calidad. Del mismo modo, el inglés es la *lingua franca* entre los desarrolladores de software libre, a pesar de que el esperanto es un idioma mucho más fácil de aprender y con una estructura más lógica. También son populares lenguajes interpretados diseñados para prototipado rápido de aplicaciones normales y servicios web, como Perl, Python y PHP.

Así como C es el lenguaje estándar, *make* es la herramienta estándar de construcción de programas, dados sus códigos fuente. El programador libre usará normalmente la versión de GNU (GNU make) [36] mucho más que la incompatible de BSD (Adam de Boor, "PMake - a tutorial") [100]. Con ellas puede especificar árboles de dependencias entre ficheros y reglas para generar los ficheros dependientes a partir de aquéllos de los que dependen. Así, se puede especificar que un fichero objeto `x.o` depende de los ficheros fuente `x.c` y `x.h` y que para construirlo hay que ejecutar **`gcc -c x.c`**. O que el ejecutable de nuestro programa depende de una colección de objetos y se monta de una determinada manera. Cuando se modifica un código fuente y luego se ejecuta *make*, se recompilarán solamente los módulos afectados y se volverá a montar el objeto final. Es una herramienta de muy bajo nivel, ya que, por ejemplo, es incapaz de averiguar por sí misma cuándo se debe recompilar un módulo en C, a pesar de que podría averiguarlo examinando las cadenas de *includes*. También es muy potente, porque puede combinar todas las herramientas disponibles de transformación de ficheros para construir objetivos muy complejos de un proyecto multilenguaje. Pero es muy complicada y muy dependiente de entornos de tipo Unix. Otras alternativas supuestamente mejores, como *jam*

(Jam Product Information) [41], *aap* (Aap Project) [1] o *ant* (The Apache Ant Project) [7] son poco usadas (esta última está ganado popularidad, sobre todo en el mundo de Java).

Dada la heterogeneidad de sistemas existentes incluso en el mundo de Unix, también se usan herramientas dedicadas a ayudarnos a que nuestros programas sean portátiles. Las herramientas de GNU *autoconf* (http://www.gnu.org/software/autoconf) [10], *automake* (http://www.gnu.org/software/automake) [32] y *libtool* (http://www.gnu.org/software/libtool) [35] facilitan estas tareas en entornos C y Unix. Dada la heterogeneidad de idiomas, juegos de caracteres y entornos culturales, el programador de C (y de muchos otros lenguajes), recurre a *gettext* (http://www.gnu.org/software/gettext) [31] y a las facilidades de internacionalización de la biblioteca estándar de C (http://www.gnu.org/software/libc) [34] para realizar programas que se puedan localizar para un entorno cultural fácilmente y en tiempo de ejecución.

Así, cuando recibimos un paquete fuente, lo más probable es que esté escrito en C, empaquetado con *tar*, comprimido con *gzip*, hecho portátil con *autoconf* y herramientas asociadas, y construible e instalable con *make*. Su instalación se llevará a cabo por medio de un proceso muy similar al siguiente:

```
tar xzvf paquete-1.3.5.tar.gz
cd paquete-1.3.5
./configure
make
make install
```

8.3. Entornos integrados de desarrollo

Un entorno integrado de desarrollo (IDE, *integrated development environment*) es un sistema que facilita el trabajo del desarrollador de software, integrando sólidamente la edición orientada al lenguaje, la compilación o interpretación, la depuración, las medidas de rendimiento, la incorporación de los códigos fuente a un sistema de control de fuentes, etc., normalmente de forma modular.

No todos los desarrolladores de software libre gustan de estas herramientas, aunque su uso se ha ido extendiendo progresivamente. En el mundo del software libre, la primera que se utilizó ampliamente fue GNU Emacs (http://www.gnu.org/software/emacs/) [33], obra estrella de Richard Stallman, escrita y extensible en Emacs Lisp, para el que existen multitud de contribuciones.

Eclipse (Eclipse - An Open Development Platform) [23] puede considerarse actualmente como el IDE de referencia en el mundo del software libre, con el inconveniente de que funciona mejor (hacia mayo de 2007) sobre una máquina virtual Java que no es libre (la de Sun, que se espera que lo

sea pronto, de todas formas). Otros entornos populares son Kdevelop (http://www.kdevelop.org) [42] para KDE, Anjuta (http://www.anjuta.org) [6] para GNOME, Netbeans (http://www.netbeans.org) [51] de Sun para Java y Code::Blocks (http://www.codeblocks.org) [18] para aplicaciones C++.

8.4. Mecanismos básicos de colaboración

El software libre es un fenómeno que es posible debido a la colaboración de comunidades distribuidas y que, por tanto, necesitan herramientas que hagan efectiva esa colaboración. Aunque durante mucho tiempo se utilizó correo postal de cintas magnéticas, el desarrollo ágil del software libre empezó cuando fue posible comunicarse rápidamente con muchas personas y distribuirles los códigos fuente de los programas o responder con comentarios y parches. Por conveniencia, mejor que enviar código, en los mensajes podría difundirse información sobre un sitio donde recogerlo. De hecho, muy al principio de los años setenta, el correo electrónico fue una extensión del protocolo de transferencia de ficheros de ARPANET.

En el mundo de Unix, a mediados de los setenta, se desarrolla *uucp*, el protocolo de transferencia de ficheros de Unix, apto para comunicar máquinas por líneas conmutadas, además de dedicadas, sobre el que se montó correo electrónico, y en 1979, el primer enlace USENET sobre UUCP. Las noticias (*news*) de USENET, un sistema de foros temático estructurado jerárquicamente y distribuido por inundación a los sitios suscritos a una jerarquía, tuvieron un papel fundamental en el desarrollo libre de software, al enviarse programas completos en fuente a los grupos de la jerarquía `comp.sources.`

En paralelo, se desarrollaron las listas de correo, entre las que cabe mencionar los gestores de listas de BITNET (1981). Hoy en día existe la tendencia a preferir las listas de correo a los grupos de noticias del tipo USENET. La razón principal ha sido el abuso con fines comerciales y la intrusión de gente "despistada", que introduce ruido en las discusiones. Además, las listas de correo ofrecen más control y pueden llegar a más gente. Los destinatarios han de suscribirse y cualquier dirección de correo es válida, aunque no haya acceso directo a Internet. El administrador de la lista puede elegir conocer quién se suscribe o dar de baja a alguien. Puede restringir las contribuciones a los miembros o puede elegir moderar los artículos antes de que aparezcan[6].

(6)También existen grupos de noticias moderados

Tradicionalmente la administración de las listas se ha hecho por correo electrónico, por medio de mensajes especiales con contraseña, lo que permite que el administrador no tenga acceso permanente a Internet, aunque cada vez eso es un fenómeno más raro, de modo que el gestor de listas de correo más popular hoy en día (Mailman, the GNU Mailing List Manager) [46] no puede ser

administrado por correo electrónico, sino necesariamente vía web. Las listas de correo juegan un papel crucial en el software libre, y llegan en muchos casos[7] a ser el único método de contribución.

Hoy en día, con la popularidad del web, muchos foros son puros foros web o *weblogs*, sin otra interfaz que la que se ofrece por medio del navegador. Pueden ser genéricos, como los populares SlashDot (Slashdot: News for Nerds") [58] o BarraPunto (http://barrapunto.com) [11], donde se anuncia nuevo software libre o se discuten noticias relacionadas, o especializados en un programa concreto, que normalmente están integrados en sitios de colaboración con diversas herramientas adicionales (*vid.* apartado 8.6.2). También hay interfaces web a grupos de noticias y listas tradicionales.

(7)Por ejemplo, las contribuciones a Linux deben hacerse como parches de texto a la lista `linux-kernel@vger.kernel.org`.

Asimismo se ha hecho popular un mecanismo de colaboración basado en *wikis*, sobre todo cuando se trata de elaborar un documento conjunto, que puede ser la especificación de un programa, un módulo o un sistema. Hablaremos de él en el apartado 8.6.2.

Finalmente, hay que mencionar los mecanismos de interacción en los que los desarrolladores conversan en tiempo real. Para software libre no suele ser un mecanismo práctico, porque con desarrolladores distribuidos por todo el mundo no es fácil encontrar una hora apropiada para todos. No obstante, hay varios proyectos que hacen uso de herramientas de charla textual, bien regularmente o bien en congresos virtuales con fechas acotadas. La herramienta más usada es el IRC (Internet Relay Chat, http://www.ietf.org/rfc/rfc2810.txt) [151], que normalmente comunica a gente por medio de "canales" temáticos establecidos a partir de una serie de servidores colaboradores. No es común que se utilicen herramientas multimedia (sonido, imagen...), probablemente porque pueden requerir conexiones de calidad no disponibles para todos y presentar problemas de disponibilidad de software libre y la dificultad de registrar y editar los resultados de las conversaciones, con el propósito de documentar.

8.5. Gestión de fuentes

A todo proyecto de desarrollo de programas le conviene tener archivada su historia, por ejemplo, porque alguna modificación puede producir un error oculto que se descubra tardíamente y haya que recuperar el original, al menos para analizar el problema. Si el proyecto lo desarrollan varias personas, es necesario también registrar al autor de cada cambio, con las razones del mismo explicadas. Si del proyecto van haciéndose entregas versionadas, es necesario saber exactamente qué versiones de cada módulo forman dichas entregas. Muchas veces, un proyecto mantiene una versión estable y otra experimental; ambas hay que mantenerlas, corregir sus errores, y transferir errores corregidos de una versión a la otra. Todo esto puede hacerse guardando y etiquetando convenientemente todas y cada una de las versiones de los ficheros, lo que generalmente se ha considerado como un coste excesivo, aunque con los discos actuales empiece a no ser tan cierto. Lo que normalmente hace un

sistema de control de fuentes, también llamado *sistema de gestión de versiones*, es registrar la historia de los ficheros como un conjunto de diferencias sobre un patrón, normalmente el más reciente, por eficiencia, etiquetando además cada diferencia con los metadatos necesarios.

Pero también queremos que un sistema de estas características sirva para que muchos programadores colaboren efectivamente, sin pisarse el trabajo, pero sin detener el avance de cada uno. Debe permitirnos, pues, que haya varios programadores trabajando de manera concurrente, pero con un cierto control. Este control puede ser optimista o pesimista. Con un control pesimista, un programador puede reservarse unos ficheros para una mejora durante un tiempo, durante el cual nadie puede tocar estos ficheros. Eso es muy seguro, pero bloqueará a otros programadores y el proyecto puede retrasarse, sobre todo si el que ha bloqueado los ficheros está ocupado en otras cosas o incluso se ha olvidado de ellos. Permitir a otros avanzar es más dinámico, pero más peligroso, ya que puede haber modificaciones incompatibles. Un sistema optimista deja avanzar, pero nos avisa cuando ha habido conflictos y nos proporciona herramientas para resolverlos.

8.5.1. CVS

CVS (Concurrent Version System) es un sistema de gestión de fuentes optimista diseñado a finales de los ochenta y que es usado por abrumadora mayoría en los proyectos libres (Concurrent Version System [20], *Open source code development with CVS*, 2ª edición) [113], "The Internet standards process", 3ª revisión [95]). Utiliza un repositorio central al que se accede según un sistema cliente/servidor. El administrador del sitio decide quiénes tienen acceso al repositorio o a qué partes de él, aunque normalmente, una vez que un desarrollador ha sido admitido en su círculo de confianza, tiene acceso a todos los ficheros. Además, se puede permitir un acceso anónimo, sólo en lectura, a todo el mundo.

El colaborador anónimo

El *CVS anónimo* es una herramienta fundamental para realizar el concepto de "entregar pronto y frecuentemente" defendido por Eric Raymond. Cualquier usuario ansioso de probar la última versión de un programa la puede extraer del CVS, descubrir errores y comunicarlos, incluso en forma de parches con la corrección. Y puede examinar la historia de todo el desarrollo.

Veamos un poco de mecánica. Un usuario avanzado quiere obtener la última versión del módulo `mod` de un repositorio accesible anónimamente en `progs.org`, directorio `/var/lib/cvs` y protocolo `pserver`. La primera vez declarará su intención de acceder:

```
cvs -d:pserver:anonymous@progs.org:/var/lib/cvs login
```

Se pide una contraseña, que será la del usuario anónimo (normalmente retorno de carro), que se registrará en un fichero local (realmente esta operación no es necesaria para acceso anónimo, pero el programa se queja si no existe el fichero con la contraseña). Seguidamente, lo importante es obtener la primera copia del módulo:

```
cvs -d:pserver:anonymous@progs.org:/var/lib/cvs co mod
```

Esto creará un directorio `mod` con todos los ficheros y los directorios del módulo y ciertos metadatos (contenidos en subdirectorios llamados `CVS`), que permitirán, entre otras cosas, no tener que repetir la información ya dada. Nuestro usuario avanzado se introduce en el directorio creado, genera el paquete y prueba:

```
cd mod
./configure
make
make install
    ...
```

Cuando quiere una nueva versión, simplemente actualiza su copia dentro de `mod`.

```
cd mod
cvs update
./configure
make
make install
...
```

Si descubre un error, puede corregirlo en el sitio y luego mandar un parche por correo electrónico al mantenedor del programa (individual o lista de correo):

```
cvs diff -ubB | mail -s "Mis parches" mod-maint@progs.org
```

El desarrollador normal

El desarrollador normal tendrá una cuenta en el servidor. Puede usar el mismo mecanismo y el mismo protocolo que el usuario anónimo, cambiando `anonymous` por el nombre de su cuenta.

Nota

Por seguridad, para cuentas con derecho a escritura, se suele usar `ssh`, que proporciona un canal autenticado y cifrado.

Una vez que tiene una copia de trabajo del módulo, puede hacer las modificaciones necesarias, y cuando considere que se han estabilizado, *comprometer* los cambios en el repositorio. Por ejemplo, si modifica los ficheros `parte.h` y `parte.c`, los comprometerá así:

```
cvs ci parte.h parte.c
```

Antes de terminar la operación, el CVS le pedirá una explicación de lo que ha hecho, que se adjuntará al historial de ambos ficheros. Asimismo incrementará el *número de revisión* de cada fichero en una unidad. Este número identifica cada momento importante en la historia de un fichero y puede usarse para recuperar cada uno de esos momentos.

¿Cuándo debe un desarrollador realizar una operación de compromiso? Ésta es una cuestión metodológica que deben acordar los miembros de un proyecto, pero parece obvio que no deben comprometerse cambios que no se compilen. Pero es preferible además que pasen una batería de pruebas mínima. En muchos proyectos es además necesario contar con la aprobación de un supervisor de proyecto o subproyecto que examine la modificación.

Durante el desarrollo de la modificación, alguien puede haber alterado otros ficheros, o incluso los mismos. Por ello conviene que el desarrollador haga, con relativa frecuencia, una operación de actualización de su copia (*cvs update*). Si se han modificado otros ficheros, puede haber cambiado el entorno y pueden fallar pruebas que antes pasaban. Si se han modificado los mismos ficheros, puede ser que estos cambios se hayan producido en lugares o rutinas que no hemos tocado o en código que sí que hemos modificado. En el primer caso no hay conflicto (al menos aparente) y la operación de modificación "mezcla" nuestra versión con la del repositorio, de manera que se generan ficheros combinados, con todas las modificaciones. En caso contrario hay conflicto, en cuyo caso hay que ponerse de acuerdo con el desarrollador que ha hecho los otros cambios y acordar una versión final.

Para identificar mejor cada componente de un proyecto, es conveniente que lleve asociada directamente información de revisión. CVS puede marcar los códigos fuente y los objetos automáticamente, siempre y cuando se observe cierta disciplina. Por ejemplo, si en un comentario del código fuente escribimos la palabra clave `$Id$`, cada vez que el fichero se comprometa en el repo-

[8] En CVS los números de revisión normalmente tienen dos componentes (mayor y menor), pero pueden tener cuatro, seis, etc.

sitorio, dicha palabra se sustituirá por una cadena de identificación en la que podremos ver el nombre del fichero, el número de revisión[8] , la fecha y la hora del compromiso y el autor del mismo:

```
$Id: parte.c,v 1.7 2003/07/11 08:20:47 joaquin Exp $
```

Si esa palabra clave la incluimos dentro de una cadena de caracteres del programa, al compilarlo la cadena aparecerá en el objeto y el ejecutable, con lo que será posible identificarlo con una herramienta (*ident*).

El administrador

Obviamente, los administradores son los que tienen y deben llevar la parte más complicada del mantenimiento del repositorio. Por ejemplo, tienen que dar de alta el proyecto, dar permisos a los desarrolladores y coordinarlos, etiquetar versiones entregadas, etc.

Es práctica común que todo proyecto tenga una versión estable y otra experimental. Para ello se crean ramas. Mientras que los que se dedican al mantenimiento corrigen errores de la rama estable, los nuevos desarrollos se hacen sobre la rama experimental. Cuando ésta se estabiliza, hay que pasarla a estable, no sin antes aplicar las correcciones hechas sobre la rama estable anterior. Esta operación se llama *mezclar*, es delicada y está soportada en CVS, aunque de forma algo primitiva. Esta idea puede extenderse al concepto de ramas experimentales que evolucionan en distintas direcciones, que llegarán a buen puerto o no, y que en todo caso, a menos que sean vías muertas, habrá que integrar total o parcialmente en el producto estable, con mezclas apropiadas.

Un derecho que nos da el software libre es la modificación de un programa para uso privado. Aunque es deseable contribuir con toda mejora al acervo comunitario, muchas veces las modificaciones que queremos hacer son demasiado específicas y no interesan al público en general. Pero sí que nos interesa incorporar la evolución del programa original. Esto puede realizarse con un tipo especial de ramificación y mezcla (*ramas de vendedor*).

El administrador también puede facilitar la coordinación del equipo por medio de mecanismos automáticos, como hacer que se produzcan mensajes de correo electrónico cuando ocurran ciertas cosas, como los compromisos, o forzar a que se realicen ciertas acciones automáticas antes de realizar un compromiso, como comprobaciones automáticas de estilo, compilaciones o pruebas.

8.5.2. Otros sistemas de gestión de fuentes

A pesar de ser el sistema de control de versiones más ampliamente usado, CVS tiene notables inconvenientes:

Nota

En 2007 `Subversion` es ya claramente el sucesor de CVS, y muchos desarrollos de software libre han migrado a él.

1) CVS no soporta renombrados o cambios de directorio de ficheros, ni tampoco metadatos (propietarios, permisos, etc.) ni enlaces simbólicos.

2) Al ser una evolución de un sistema de control de versiones de ficheros individuales, no soporta, naturalmente, el control de versiones de grupos completos.

3) CVS no soporta conjuntos de cambios coherentes. En efecto, para añadir una característica o corregir un error, puede ser preciso cambiar varios ficheros. Estos cambios deberían ser atómicos.

4) En CVS es bastante complicado el uso de ramas y mezclas. En efecto, si creamos una rama experimental de un proyecto y deseamos incluir las correcciones hechas en la rama estable, es preciso conocer en detalle qué correcciones se han hecho ya, para no hacerlas varias veces.

5) CVS depende de un servidor centralizado, y aunque se puede trabajar sin conexión, sí que la necesitamos para generar versiones, compararlas y mezclarlas.

6) CVS no genera, sin la ayuda de herramientas aparte, el fichero `changelog`, que muestra la historia global de cambios de un proyecto.

7) CVS no soporta bien proyectos con un número muy grande de ficheros, como es el caso del núcleo Linux.

Y sin embargo, hay otros sistemas libres que solucionan varios de estos problemas. Podemos destacar el ya designado sucesor de CVS, Subversion (http://subversion.tigris.org) [62], (http://svnbook.red-bean.com/) [96], que resuelve estrictamente los problemas básicos de CVS y puede utilizar extensiones de HTTP (WebDAV) para soslayar políticas de seguridad agresivas.

El modelo de desarrollo basado en un repositorio centralizado, aunque apto para el trabajo cooperativo, no satisface todas las expectativas, ya que por un lado depende de la accesibilidad y el buen funcionamiento del servidor, y por otro de los administradores de ese servidor, el hecho de que podamos crear nuestras propias ramas de desarrollo. A veces se desean repositorios distribuidos que permitan a cualquiera tener un repositorio con una rama privada o pública que luego pueda mezclar o no con la oficial. Así funcionan *GNU arch* (Arch Revision Control System) [8] o *bazaar* (Bazaar GPL Distributed Version Control Software) [12], así como el sistema propietario BitKeeper (Bitkeeper Source Management) [14], elegido por Linus Torvalds para mantener Linux

desde febrero de 2002, ya que según él no había ninguna herramienta libre apropiada. Se dice que el uso de Bitkeeper dobló el ritmo de desarrollo de Linux. No obstante, la decisión fue muy criticada porque era propietario, con una licencia que permitía a los proyectos libres obtenerlo gratuitamente siempre que todas las operaciones de compromiso de cambios con sus metadatos se registrasen en un servidor público designado por los propietarios y accesible a todo el mundo, y siempre que el licenciatario no desarrollara otro sistema de control de fuentes que pudiera competir con él. Fue precisamente el intento de desarrollar un producto libre compatible por parte de un empleado de la misma empresa donde trabajaba Linus Torvalds el detonante del cambio de sistema de gestión de fuentes. Linus desarrolla rápidamente un sustituto provisional, *git* ("Git manual page") [218], que pronto se convierte en definitivo, donde se condensa toda la experiencia en el desarrollo cooperativo y descentralizado de Linux: soporta proyectos de gran tamaño de forma descentralizada, facilitando en gran medida el desarrollo de ramas tentativas y su mezcla con otras o con la principal, con mecanismos de seguridad criptográficos que impiden alterar el historial. A partir de abril de 2005 Linux se mantiene con *git* o envoltorios del mismo (por ejemplo, *cogito* –"Cogito manual page"– [90].

8.6. Documentación

En el mundo del software libre, apenas se usan procesadores de texto WYSIWYG y otras herramientas ofimáticas que tanto éxito tienen en otros entornos, a pesar de que hay ya herramientas libres, como OpenOffice.org. Ello es debido a varios factores importantes:

- Es conveniente mantener la documentación bajo control de fuentes, y los sistemas de control de fuentes, como CVS, aunque admiten formatos binarios, prefieren formatos textuales transparentes, editables con un editor de texto normal y procesables con herramientas desarrolladas para programas, que nos permitan ver fácilmente las diferencias entre versiones, generar y aplicar parches basados en esas diferencias, y realizar operaciones de mezcla.

Nota

En Unix las herramientas que hacen estas operaciones más comunes son *diff*, *diff3*, *patch* y *merge*.

- Ciertas licencias de documentación libre, especialmente la GFDL (*vid.* apartado 10.2.1), exigen formatos transparentes, sobre todo porque facilitan el trabajo a quienes realizan documentos derivados.

- Las herramientas WYSIWYG ("what you see is what you get") generalmente no contienen más información que la estricta de visualización, por lo que hacen muy difícil, si no imposible, el procesamiento automático, como identificar autores o título, y la conversión a otros formatos. Incluso aunque permitan conversión de formatos, ésta suele hacerse de forma interactiva, y es muchas veces imposible automatizarla (con *make*, por ejemplo).

- Generalmente las herramientas ofimáticas generan unos formatos de ficheros muy voluminosos, asunto muy desagradable para los desarrolladores o los repositorios.

Por todo ello, el programador libre, acostumbrado también a programar y compilar, prefiere formatos de documentación transparentes, en muchos casos simple texto puro y en muchos otros formatos de documentación procesables.

Los formatos procesables utilizados no son muchos. Tradicionalmente, en el mundo de Unix los programas se han documentado en los formatos esperados por la familia de procesadores *roff*, con versión libre (*GNU troff*) [37] de Norman Walsh. No obstante, esta práctica se ha ido abandonando, excepto para las páginas de manual tradicionales, ya que es casi obligado preparar páginas de manual para las herramientas más básicas del sistema. Debido a que muchas páginas de manual han crecido excesivamente de modo que difícilmente se las puede llamar *páginas*, fue necesario elaborar un formato alternativo hipertextual que permitiera seguir documentos estructurados con índices y referencias cruzadas. El proyecto GNU diseñó el formato *texinfo* (Texinfo - The GNU Documentation System) [63] y lo convirtió en su estándar. Este formato permite obtener documentos navegables con la herramienta *info* o dentro del editor *emacs*, y a su vez, obtener documentos impresos de calidad con ayuda del procesador de textos TeX, de Donald Knuth (The TeXbook) [156].

El formato *texinfo* puede traducirse a HTML multipágina si se desea, y mucha gente prefiere ver la información con un navegador web, pero se pierde la capacidad de búsqueda de palabras en un documento. ésta es una de las consecuencias indeseables de la popularidad de HTML, ya que los navegadores no implementan el concepto de *documento multipágina*, a pesar de que existen elementos `link` que permiten enlazar las partes.

Existe una imperiosa demanda de que los documentos complejos se puedan ver como páginas web multipágina fácilmente navegables. Hay gente que escribe documentación en LaTeX (*LaTeX user's guide and reference manual*) [163], también aplicación de TeX, muy popular entre los científicos, más expresiva que Texinfo y convertible a HTML multipágina con ciertas herramientas (The LaTeX Web Companion) [130], siempre que se guarde cierta disciplina. En efecto, las aplicaciones de TeX son conjuntos de macros que combinan operadores tipográficos de muy bajo nivel para convertirlos en lenguajes abstractos que trabajan con conceptos de alto nivel (autor, título, resumen, capítulo, apartado, etc.). Si sólo se utilizan las macros básicas, la conversión es sencilla. Pero como nadie impide usar operadores de bajo nivel y, además, existen cantidades enormes de paquetes de macros fuera de la capacidad de mantenimiento de los mantenedores de los conversores, es difícil conseguir que las conversiones salgan bien.

8.6.1. DocBook

El problema radica en que no existe separación entre contenido y presentación, ni en las aplicaciones de TeX ni en las de *nroff*, ya que la abstracción se construye por capas. Esta separación la tienen las aplicaciones de SGML (*standard generalized markup language*) [81] y XML (*extensible markup language*) [224], en las que la presentación se especifica con *hojas de estilo* completamente separadas. Muy pronto empezaron a utilizarse aplicaciones muy sencillas de SGML, como `linuxdoc` y `debiandoc`, pero debido a su escasa capacidad expresiva, se optó por `DocBook` (*DocBook: the definitive guide*) [225].

`DocBook` es una aplicación de SGML originalmente desarrollada para documentación técnica informática y hoy con una variante XML. Actualmente DocBook es el estándar de formato de documentación libre para muchos proyectos (Linux Documentation Project, KDE, GNOME, Mandriva Linux, etc.) y un objetivo que se debe alcanzar en otros (Linux, *BSD, Debian, etc).

Sin embargo, DocBook es un lenguaje complejo, plagado de etiquetas, por lo que es conveniente disponer de herramientas de ayuda a la edición, aún escasas y elementales, entre las cuales la más popular es el modo `psgml` de *emacs*. También es pesado de procesar y los procesadores libres aún generan una calidad de documentos poco atractiva.

8.6.2. *Wikis*

Mucha gente encuentra demasiado complicado escribir documentación con lenguajes tan complejos como DocBook y mecanismos de colaboración como CVS. Por ello se ha hecho muy popular un nuevo mecanismo de colaboración para la elaboración de documentos en línea vía web llamado *wiki*, inventado por Ward Cunningham ("Wiki design principles") [97], puesto por primera vez en servicio en 1995 y hoy ampliamente utilizado en muy diversas variantes para elaborar documentos muy dinámicos, no destinados a ser impresos y muchas veces con una vida corta (por ejemplo, la organización de una conferencia).

Al contrario que DocBook, un *wiki* tiene un lenguaje de marcas muy simple y conciso que recuerda la presentación final, sin ser exactamente como ella. Por ejemplo, los párrafos se separan por una línea en blanco, los elementos de listas empiezan por un guión si no se numeran y por un cero si se numeran, y las celdas de las tablas se separan por barras verticales y horizontales.

Tampoco existe el concepto de "documento completo", sino que un *wiki* es más bien un conjunto de pequeños documentos enlazados que se van creando a medida que es necesario explicar un nuevo concepto o tema. La creación de los documentos es casi automática, ya que la herramienta de edición muestra

muy claramente que hemos introducido un concepto (por medio de un `NombreWiki`, casi siempre dos palabras juntas con la primera letra en mayúscula). Casi ningún *wiki* admite hiperenlaces dentro de la misma página.

A diferencia de CVS, cualquiera puede escribir en un *wiki*, aunque se aconseja que el autor se identifique con un registro previo. Cuando se visita un *wiki* se ve que todas las páginas tienen un botón que permite su edición. Si se pulsa, el navegador nos mostrará en un formulario el código fuente del documento, que podremos cambiar. No es una edición WYSIWYG, lo que disuade al que quiera fastidiar, pero es tan sencillo que cualquier interesado puede modificar documentos con muy pequeño esfuerzo.

Los *wikis* llevan su propio control de versiones de documentos, de modo que generalmente están accesibles todas sus versiones, con indicación de quién las hizo y cuándo. También se pueden comparar con facilidad. Además, suelen llevar mecanismos de búsqueda, al menos por nombre de página y por palabra contenida.

Normalmente el autor original de una página querrá enterarse de las modificaciones que se le hagan. Para ello puede suscribirse a los cambios y recibir notificaciones de los mismos por correo electrónico. A veces, quien ve un documento no se atreve a cambiar nada, pero puede hacer un comentario. Normalmente toda página *wiki* tiene asociado un foro de comentarios que se pegan al final del documento y que bien el autor original o bien alguien que asuma la función de editor puede emplear para reformar el texto original, posiblemente moviendo frases de los comentarios a los sitios oportunos.

Sugerencia

La mejor forma de entender el concepto de *wiki* es accediendo a uno y experimentando en una página destinada a ello, denominada habitualmente `SandBox`.

8.7. Gestión de errores y otros temas

Uno de los puntos fuertes del modelo de desarrollo libre es que la comunidad contribuya con informes de errores y que sienta que esos informes o soluciones son tenidos en cuenta. Por ello es necesario un mecanismo sencillo de informe de errores, de modo que los desarrolladores reciban información suficiente, de forma sistemática y con todos los detalles necesarios, bien aportados por el colaborador, como la explicación de lo que pasa, el nivel de importancia y la posible solución, o bien por algún mecanismo automático que determine, por ejemplo, la versión del programa y el entorno en el que funciona. Asimismo es necesario que los errores se guarden en una base de datos que pueda ser consultada, para ver si un error ya ha sido comunicado, si ha sido corregido, su nivel de importancia, etc.

Existen varios de estos sistemas, de diferente filosofía. Unos son vía web, otros vía correo electrónico, a partir de algún programa intermediario. Todos tienen interfaz web para consulta. Unos permiten informes anónimos, otros requieren identificación (una dirección de correo válida) para evitar el ruido. Aunque los procedimientos vía web parecen los más sencillos, con ellos no es posible obtener fácilmente información automática del entorno del error. Por ejemplo, el sistema de Debian proporciona programas como reportbug, que tras preguntar el nombre del paquete sobre el que queremos informar, consulta al servidor de errores qué errores ya hay comunicados sobre éste. Si ninguno de ellos se refiere a nuestro problema, nos pide una descripción del mismo, un nivel de importancia ("crítico", "grave", "serio", "importante", "no se puede regenerar a partir de los códigos fuente", "normal", "menor" o "sugerencia") y etiquetas sobre su categoría (por ejemplo, "seguridad"). Tras ello, si confirmamos la petición, automáticamente averigua la versión del paquete y de aquéllos de los que depende, así como la versión y la arquitectura del núcleo. Obviamente, la dirección de correo electrónico la sabe, por lo que se envía al sitio correcto un informe similar al siguiente:

```
Package: w3m-ssl
Version: 0.2.1-4
Severity: important

After reloading a page containing complex tables several dozen times, w3m
had used all physical memory and thrashing commenced. This is an Alpha machine.

--System Information
Debian Release: testing/unstable
Kernel Version: Linux romana 2.2.19 #1 Fri Jun 1 18:20:08 PDT 2001 alpha Unknown

Versions of the packages w3m-ssl depends on:
ii  libc6.1     2.2.3-7        GNU C Library: Shared libraries and Timezone data
ii  libgc5      5.0.alpha4-8   Conservative garbage collector for C
ii  libgpmg1    1.19.3-6       General Purpose Mouse Library [libc6]
ii  libncurses5 5.2.20010318-3 Shared libraries for terminal handling
ii  libssl0.9.6 0.9.6a-3       SSL shared libraries
ii  w3m         0.2.1-2        WWW browsable pager with tables/frames support
```

Este mensaje genera un número de error que nos es devuelto, enviado al mantenedor y guardado en la base de datos. Cuando se resuelva el error, recibiremos también una notificación. Cada error tiene asignada una dirección de correo electrónico que se puede utilizar para proporcionar información adicional, por ejemplo. La base de datos de errores la podremos consultar en http://bugs.debian.org en cualquier momento.

A veces los sistemas de seguimiento de errores tienen mecanismos para asignar la corrección a alguien y ponerle un plazo. También existen otros temas, como trabajos pendientes, mejoras solicitadas, traducciones, etc., que requieren asimismo mecanismos de gestión similares. En software libre generalmente no se pueden emplear mecanismos muy rígidos de gestión de los trabajos que tiene que hacer cada desarrollador. Después de todo, muchos colaboradores son voluntarios y no se los puede obligar a nada. No obstante, sí que se pueden definir tareas y esperar a que alguien se dé de alta en el sistema y las asuma, declarando un plazo para ello. Tanto si se tiene control sobre lo que pueden hacer determinadas personas como si no, siempre es conveniente tener controladas las tareas que hay que hacer, las dependencias que tienen, su nivel de importancia y quiénes están trabajando en ellas. Muchos de los proyectos importantes gestionan estos temas con Bugzilla (*The Bugzilla guide*) [89] o derivados del mismo.

A veces una persona, trabajando en un proyecto, puede descubrir errores en otro del que depende su trabajo, pero que tiene un sistema de gestión de errores distinto al que está acostumbrada. Esto es especialmente cierto para usuarios de distribuciones que quieren utilizar una única herramienta para informar y seguir la corrección de sus errores. Para facilitar el informe y el seguimiento de esos errores, puede ser conveniente *federar* distintos sistemas, como hace *Malone* (The Malone Bug Tracker) [47].

8.8. Soporte para otras arquitecturas

El soporte mínimo para trabajar con un programa portable es el acceso a *granjas de compilación*, que permiten compilarlo en varias arquitecturas y sistemas operativos. Por ejemplo, SourceForge (*vid.* apartado 8.9.1) ofreció durante un tiempo entornos Debian GNU/Linux para Intel x86, DEC Alpha, PowerPC y SPARC, además de entornos Solaris y Mac OS/X.

También es útil poder probar (no sólo compilar) el programa en esos entornos. Pero este servicio requiere más recursos y más tiempo de administrador. Ya el servicio de compilación puede ser problemático, porque habitualmente hay que proporcionar entornos de compilación para varios lenguajes, con una gran cantidad de bibliotecas. Si lo que se quiere es probar un programa cualquiera, las dificultades aumentan exponencialmente, no sólo porque es muy difícil disponer de los recursos requeridos, sino por razones de seguridad, que pueden hacer muy complicado administrar estos sistemas. No obstante, existen unos pocos servicios de *granja de servidores*, con instalaciones estándar de arquitecturas diversas, que pueden permitirnos probar algunas cosas.

Las granjas públicas mencionadas anteriormente suelen ser un servicio de uso manual. El desarrollador invitado copia sus ficheros en una de esas máquinas, los compila y prueba el resultado. Probablemente deberá hacerlo de vez en cuando, en el momento previo a la liberación de una versión importante del programa. Puede ser mucho más interesante que las compilaciones y la eje-

cución de las pruebas de regresión se hagan sistemáticamente, de forma automática, por ejemplo todas las noches, si ha habido cambio en los códigos fuente. Así funcionan algunos proyectos importantes, que proporcionan una infraestructura propia para desarrolladores externos, que suele llamarse *tinderbox* ('yesquero'). Es el caso de Mozilla, financiado por Netscape, cuyo *tinderbox* (http://www.mozilla.org/tinderbox.html) [50] presenta una interfaz web a los resultados de la compilación y pruebas de componentes de su navegador en todas las arquitecturas donde funciona. Esta interfaz está íntimamente relacionada con el CVS y muestra esos resultados para distintos estados (entre compromisos), identificando al responsable de los fallos y facilitando el avance, soslayando el problema hasta que se solucione. También usan yesqueros los proyectos OpenOffice y FreeBSD, al menos.

8.9. Sitios de soporte al desarrollo

Los sitios de soporte al desarrollo proporcionan, de manera más o menos integrada, todos los servicios descritos anteriormente, junto con algunos adicionales que permiten la búsqueda de proyectos por categorías y su clasificación según parámetros sencillos de actividad. Ello libera al desarrollador de montarse y administrarse toda una infraestructura de colaboración para poder concentrarse en su proyecto.

8.9.1. SourceForge

Por lo que respecta a este tipo de servicios, uno de los primeros en establecerse, y el más popular, fue SourceForge (http://sourceforge.net) [61], gestionado por la OSDN (Open Software Development Network), subsidiaria de VA Software, que albergaba en marzo de 2007 más de 144.000 proyectos. Está estructurado en torno a un conjunto de programas del mismo nombre, que hasta la versión 2 fueron software libre.

SourceForge, como prototipo de estos sitios, ofrece una interfaz web o portal global de entrada (http://sourceforge.net/) y un subportal por proyecto (http://proyecto.sourceforge.net). En la interfaz global podemos ver noticias, anuncios, enlaces y una invitación a hacerse miembro o a entrar si ya se es. Para colaborar en el sitio, conviene hacerse miembro, lo cual es imprescindible si se quiere crear un proyecto nuevo o participar en uno ya existente. Para ser espectador no es necesario, y como tales, podemos ver cuáles son los proyectos que experimentan un desarrollo más activo o los que son descargados con más frecuencia, y podemos buscar proyectos por categorías o dando una palabra de su descripción, y se nos mostrarán ordenados por su nivel de actividad. Dentro de cada proyecto, podemos ver su descripción, su estado (alfa, beta, producción), sus descriptores (lenguaje, sistema operativo, temática, tipo de usuarios, idioma, licencia...), errores y temas pendientes o subsanados, los de-

talles de su actividad en el tiempo..., o descargarlo. También podemos participar en foros o informar de errores, incluso de forma anónima, lo cual no es muy conveniente (porque, por ejemplo, no nos pueden responder).

Cualquier usuario autenticado puede solicitar dar de alta un proyecto, que será admitido por los administradores del sitio si cumple las políticas del mismo, que en el caso de SourceForge son bastante liberales. Una vez autorizado, el creador puede dar de alta a otros usuarios registrados como administradores adicionales o como desarrolladores, con acceso a la modificación de fuentes. Tras la autorización, no hay muchos más controles sobre el proyecto, lo que ocasiona la existencia de muchos proyectos muertos. Esto no confunde demasiado a los usuarios, porque como en las búsquedas los proyectos se muestran ordenados por el grado actividad, los que tienen una actividad baja o nula apenas son visibles. Estos proyectos corren el riesgo de ser eliminados por los propietarios del sitio. Los servicios que SourceForge proporciona a un proyecto, y que podríamos esperar de cualquier servicio similar, son los siguientes:

- Albergue para las páginas web del portal del proyecto, en la dirección *proyecto.sourceforge.net*, donde se muestra el mismo al público. Estas páginas pueden ser estáticas o dinámicas (con CGI o PHP), en cuyo caso pueden hacer uso de una base de datos (MySQL). Se introducen directamente mediante órdenes de copia remota y pueden manipularse por medio de sesiones interactivas de terminal remoto (SSH).

- Opcionalmente un servidor virtual que responda a direcciones de un dominio obtenido aparte, como *www.proyecto.org* o *cvs.proyecto.org*.

- Tantos foros web y/o listas de correo como sean necesarios, según el criterio de un administrador.

- Un servicio de noticias donde los administradores anuncien las novedades sobre el proyecto.

- Rastreadores (*trackers*) para informe y seguimiento de errores, peticiones de soporte, peticiones de mejoras o integración de parches. Los administradores dan una prioridad al asunto y asignan su solución a un desarrollador.

- Gestores de tareas, similares a los rastreadores, que permiten definir subproyectos con una serie de tareas. Estas tareas, además de una prioridad, tienen un plazo. Los desarrolladores a los que se les asignan pueden manifestar de vez en cuando un porcentaje de realización de las tareas.

- Un CVS o un Subversion con derechos iniciales de acceso para todos los desarrolladores.

- Servicio de subida y bajada de paquetes de software. Utilizándolo se tiene un registro de las versiones introducidas y se posibilita que los interesados

reciban un aviso cuando esto suceda. Además, la subida implica la creación de varias réplicas en todo el mundo, lo que facilita la distribución.

- Servicio de publicación de documentos en HTML. Cualquiera puede registrarlos, pero sólo después de la aprobación por parte de un administrador serán visibles.

- Copia de seguridad para recuperación de desastres, como rotura de disco, no de errores de usuario, como borrar un fichero accidentalmente.

- Mecanismo integrado de donaciones a usuarios, a proyectos y al propio SourceForge.

Un usuario autenticado tiene una página personal donde se reúne toda la información de su interés, como proyectos a los que está asociado, temas o tareas que tiene pendientes, así como foros y ficheros que ha declarado que quiere vigilar. Además, para que no tenga que estar pendiente de su página personal, el usuario recibirá por correo electrónico notificaciones de lo que quiere vigilar.

8.9.2. Herederos de SourceForge

En 2001 VA Software estuvo a punto de quebrar, en plena crisis de las puntocom. Entonces anunció una nueva versión de su software SourceForge con licencia no libre, en un intento de procurarse una fuente de ingresos vendiéndolo a empresas para sus desarrollos internos. Asimismo eliminó mecanismos que permitían volcar un proyecto para llevarlo a otro sitio. Ambos hechos fueron vistos como una amenaza por la que los miles de proyectos alojados en SourceForge quedarían atrapados en manos de una sola empresa, que utilizaría la plataforma para mostrar software no libre. Ante eso y la posibilidad de que el sitio cerrara se desarrollaron hijos de la versión libre y se abrieron portales basados en ella, entre los que destacan Savannah (http://savannah.gnu.org) [57], dedicado al proyecto GNU y a otros programas con licencia de tipo *copyleft*, o BerliOS (BerliOS: The Open Source Mediator) [13], concebido como punto de encuentro entre desarrolladores de software libre y empresas. Sin embargo, esto es sólo un paso con vistas a desarrollar una plataforma distribuida y replicada, donde nadie tenga control absoluto de los proyectos (Savannah The Next Generation, 2001) [98].

Otro ejemplo de sistema de gestión de proyectos de software libre es Launchpad (https://launchpad.net) [43], utilizado por Ubuntu para el desarrollo de cada versión de la distribución. Launchpad no es un repositorio de código fuente, sino que tiene como propósito ofrecer soporte para seguimiento de código, incidencias y traducciones. Para ello utiliza la ya mencionada herramienta Malone, que permite redirigir las incidencias a cada repositorio de código de los módulos afectados.

8.9.3. Otros sitios y programas

Naturalmente, se han desarrollado y siguen desarrollándose sistemas de colaboración, y algunas empresas hacen negocio del mantenimiento y del servicio de esos sitios. Por ejemplo, el proyecto Tigris (Tigris.org: Open Source Software Engineering Tools) [64], que sólo mantiene proyectos de ingeniería de software libre, usa un portal de colaboración (SourceCast) mantenido por una empresa de servicios (CollabNet), que también mantiene sitios de proyectos individuales, como OpenOffice.org. Nuevos sitios vienen adoptando nuevo software libre, como GForce (http://gforge.org) [30], utilizado por el proyecto Debian (http://alioth.debian.org) [5]. Puede verse una comparación exhaustiva de muchos sitios en "Comparison of free/open source hosting (FOSPhost) sites available for hosting projects externally from project owners" [202].

9. Estudio de casos

> "GNU, which stands for 'Gnu's Not Unix', is the name for the complete Unix-compatible software system which I am writing so that I can give it away free to everyone who can use it. Several other volunteers are helping me. Contributions of time, money, programs and equipment are greatly needed."
>
> "GNU, que significa 'Gnu No es Unix', es el nombre de un sistema de software completamente compatible con Unix que estoy escribiendo para poder entregarlo libremente a quien pueda utilizarlo. Hay varios voluntarios ayudándome. Son muy necesarias las contribuciones de tiempo, dinero, programas y equipo."
>
> Richard Stallman, "The GNU Manifesto" (1985)

Este capítulo está dedicado íntegramente a estudiar más a fondo algunos de los proyectos de software libre más interesantes en cuanto a impacto en el mundo del software libre, resultados obtenidos, modelo de gestión, evolución histórica, etc. Por supuesto, el número de proyectos que se van a presentar es muy pequeño en comparación con el número de proyectos libres totales (decenas de miles), por lo que este capítulo no se puede considerar completo, ni se podría considerar nunca completo. Aun así, esperamos que tras su lectura, el lector pueda tener al menos una percepción de cómo se ha llevado a la práctica la teoría que se ha ido presentando a lo largo de este libro.

Los proyectos escogidos abarcan desde aplicaciones de bajo nivel –ésas que interactúan más bien con el sistema físico del ordenador en vez de con el usuario– hasta entornos de trabajo para el usuario final. Hemos incluido también proyectos de software libre que, en un principio, no son íntegramente de desarrollo. Tal es el caso de las distribuciones, cuyo trabajo es más bien de integración, ya que se dedican principalmente a tomar un conjunto amplio, pero limitado, de aplicaciones independientes y a crear un sistema en el que todo interactúe de manera efectiva, incluyendo también facilidades para instalar, actualizar y borrar nuevas aplicaciones según lo desee el usuario.

Los proyectos de más bajo nivel que vamos a ver serán Linux, el núcleo del sistema operativo libre más popular a día de hoy, y FreeBSD, que combina un núcleo de la familia BSD con una serie de aplicaciones y utilidades realizadas por terceros proyectos al más puro estilo de las distribuciones. Los entornos de trabajo de usuario final que estudiaremos serán KDE y GNOME, ciertamente los más extendidos y populares. Los servidores –uno de los puntos fuertes de los sistemas libres– estarán representados en este capítulo por Apache, líder en el mercado de los servidores WWW. Asimismo, presentaremos Mozilla, uno de los clientes WWW (en realidad, mucho más que eso) con los que contamos en el mundo del software libre. El último proyecto que se va a ver en este capítulo es OpenOffice.org, un paquete ofimático (*suite*) libre.

Para terminar, hemos creído conveniente profundizar en dos de las distribuciones más populares, Red Hat Linux y Debian GNU/Linux, y hacer una comparación en cuanto a tamaño con otros sistemas ampliamente utilizados, como pueden ser los de Microsoft Windows o Solaris.

Tras las presentaciones de los diferentes casos de estudio, mostraremos una tabla con las características más notables de cada aplicación o proyecto. Uno de los parámetros que puede sorprender más al lector son los resultados de estimación de coste, de duración y de número medio de desarrolladores. Estos resultados los hemos obtenido por medios tradicionales en la ingeniería del software, en especial, el modelo COCOMO. El modelo COCOMO (*Software Engineering Economics*, 1981) [93] toma como medida de entrada el número de líneas de código fuente y genera estimaciones de coste total, tiempo de desarrollo y esfuerzo dedicado. COCOMO es un modelo pensado para procesos de generación de software "clásicos" (desarrollo en cascada o en uve) y para proyectos de tamaño medio o grande, por lo que las cifras que nos ofrece en nuestro caso han de ser tomadas con mucho cuidado. De todas maneras, los resultados nos pueden dar una idea del orden de magnitud en el que nos movemos, informándonos sobre los esfuerzos óptimos necesarios si se hubiera utilizado un modelo de desarrollo propietario.

En general, lo que más asombra de los resultados de COCOMO es su estimación de costes. En dicha estimación se tienen en cuenta dos factores: el salario medio de un desarrollador y el factor de *overhead*. En el cálculo de la estimación de costes, se ha tomado el salario medio para un programador de sistemas a tiempo completo de la encuesta del año 2000 de acuerdo con "Salary survey 2000" [235]. El *overhead* es el sobrecoste que toda empresa ha de asumir para que el producto salga a la calle con independencia del salario de los programadores. En este apartado se incluyen desde el salario de las secretarias y el equipo de marketing hasta los costes de las fotocopias, la luz, los equipos de hardware, etc. En resumen, el coste calculado por COCOMO es el coste total que le supondría a una empresa crear un software del tamaño especificado, y no se ha de ver simplemente como el dinero que percibirían los programadores por realizar el software. Una vez dicho esto, los cálculos de costes dejan de parecer tan abultados.

9.1. Linux

El núcleo Linux es, sin duda, la aplicación estrella del software libre, hasta el punto de que, aun siendo una pequeña parte del sistema, ha dado nombre a su totalidad. Es más, se podría incluso afirmar que el propio software libre se confunde en multitud de ocasiones con Linux, lo que no deja de ser un gran desacierto, ya que existe software libre que se ejecuta sobre sistemas que no se basan en Linux (de hecho, una de las grandes metas del movimiento y de muchos proyectos de software libre es que las aplicaciones puedan ejecutarse

en multitud de entornos). Por otro lado, también existen aplicaciones que funcionan en Linux y que no son software libre (Acrobat Reader, el lector de documentos PDF, también tiene su versión para Linux).

Nota

Para evitar la asociación del software libre únicamente con sistemas Linux existen varios proyectos que se dedican a integrar y a distribuir aplicaciones libres que se ejecutan en sistemas Windows. Uno de los pioneros en esta actividad (y probablmente el que llegó a ser más conocido y completo) fue GNUWin, que distribuía en un CD autoarrancable más de un centenar de aplicaciones libres para sistemas Win32. La mayoría de estas aplicaciones también se encuentran disponibles en las distribuciones GNU/Linux comunes, por lo que era una buena herramienta para ir preparando el cambio de sistemas Windows a GNU/Linux de manera sosegada. A principios de 2007 pueden encontrarse otros sistemas parecidos, como WinLibre.

9.1.1. Historia de Linux

La historia de Linux es una de las más conocidas dentro del mundo del software libre, seguramente porque se asemeja más a un cuento que a la historia de un programa de ordenador. En 1991, un estudiante finés llamado Linus Torvalds decidió que quería aprender a utilizar el modo protegido 386 en una máquina que su limitado bolsillo le había permitido adquirir. Por entonces –y aún hoy en día– en los cursos de sistemas operativos universitarios existía un núcleo de sistema operativo gestado con fines académicos llamado Minix. A la cabeza del grupo de desarrollo de Minix –basado en los sistemas Unix tradicionales– se encontraba uno de los profesores de universidad más prestigiosos, Andrew Tanenbaum. Minix era un sistema limitado, pero bastante capaz y bien diseñado que contaba con una gran comunidad –académica e ingenieril– a su alrededor.

Minix tenía una licencia de libre distribución y se podía utilizar sin problema para fines académicos, pero tenía la gran desventaja de que personas ajenas a la Universidad de Ámsterdam no podían integrar mejoras en él, sino que estas mejoras las debían hacer de manera independiente, generalmente por medio de parches. Esto suponía que en la práctica existiera una versión de Minix oficial que todo el mundo usaba, y luego una larga serie de parches que había que aplicar posteriormente para obtener funcionalidades adicionales.

A mediados de 1991, Linus –el todavía anónimo estudiante finés– mandó un mensaje al grupo de noticias de Minix anunciando que iba a empezar desde cero un núcleo de sistema operativo basado en Minix, pero sin incluir código del mismo. Para entonces, Linus –aun cuando no dijera explícitamente que lo fuera a publicar con una licencia libre– apuntó que el sistema que iba a crear no iba a tener las *barreras* que tenía Minix, por lo que –sin saberlo, y probablemente sin quererlo– estaba dando el primer paso para quedarse con la comunidad que entonces se aglutinaba alrededor de Minix.

La versión 0.02, que data de octubre de 1991, aunque muy limitada, ya podía ejecutar terminales *bash* y el compilador GCC. En los meses siguientes el número de aportaciones externas fue creciendo hasta el punto de que ya en mar-

zo de 1992 Linus publicaba la versión 0.95, que fue ampliamente reconocida como casi estable. El camino hacia la versión 1.0 –que suele asociarse con la estabilidad– fue, sin embargo, todavía largo: en diciembre de 1993 se publicaría, por ejemplo, la versión 0.99pl14 (que viene a ser como la decimocuarta versión corregida de la versión 0.99) y finalmente, en marzo de 1994, Linux 1.0 vio la luz. Ya para entonces, Linux se publicaba bajo las condiciones de la licencia GPL, según el propio Torvalds una de las mejores decisiones que ha tomado, ya que ayudó sobremanera a la distribución y a la popularización de su núcleo. En "Evolution in open source software: a case study" [128] se puede encontrar un análisis exhaustivo de la evolución de las diferentes versiones del núcleo Linux en cuanto a tamaño y modularidad.

Nota

Para los anales de la historia del software libre queda también el debate que tuvo lugar a finales de enero de 1992 en el grupo de noticias de Minix entre Andrew Tanenbaum y Linus Torvalds. Tanenbaum, probablemente ya un poco molesto por el éxito de Torvalds con su "juguete", atacó de manera un poco desproporcionada a Linux y a Linus. El punto esencial es que Linux era un sistema monolítico (el núcleo es una pieza que integra todos los manejadores y demás) y no micronúcleo o *microkernel* (el núcleo tiene un diseño modular, lo que permite que sea más pequeño y que se puedan cargar módulos bajo demanda). Se puede seguir la discusión original tal como ocurrió en el grupo de noticias en "The Tanenbaum-Torvalds debate" [214].

9.1.2. El modo de trabajo de Linux

La forma de trabajar de Torvalds no era muy común en aquellos tiempos. El desarrollo se basaba principalmente en una lista de correo[9]. En la lista de correo no sólo se discutía, sino que también se desarrollaba. Y es que a Torvalds le gustaba sobremanera que toda la vida del proyecto se viera reflejada en la lista, por lo que pedía que los parches se enviaran a la misma. En contra de lo que uno pudiera esperar (que se enviara el parche como adjunto), Linus prefería que se mandara el código en el cuerpo del mensaje para que él y los demás pudieran comentar el código. En cualquier caso, aun cuando muchos opinaran y enviaran correcciones o nuevas funcionalidades, lo que era indiscutible es que la última palabra, el que decidía lo que entraba en Linux, correspondía a Linus Torvalds. Hasta cierto punto, este modo de funcionamiento sigue vigente en 2007.

(9)La dirección de la lista de correo electrónico es linux-kernel@vger.kernel.org. Se puede ver el histórico de todos los mensajes en http://www.uwsg.indiana.edu/hypermail/linux/kernel/.

Nota

La consolidación de Linus Torvalds como "dictador benevolente" ha dado pie a un amplio anecdotario dentro del proyecto. Así, se comenta que si una idea gusta, se ha de implementar. Si no gusta, también se ha de implementar. El corolario, por tanto, es que las buenas ideas no sirven para nada (sin código, por supuesto). Por otro lado, si no gusta la implementación, hay que insistir. Conocido es el caso de Gooch, para quien el santo Job era un aprendiz. Gooch realizó hasta ciento cuarenta y seis parches paralelos hasta que Linus decidió finalmente integrarlo en la rama oficial del núcleo.

Otra de las ideas innovadoras de Torvalds fue el desarrollo paralelo de dos ramas del núcleo: la estable (cuyo segundo número de versión suele ser par, como por ejemplo 2.4.18) y la inestable (cuyo segundo número de versión es impar, como 2.5.12). Como no podía ser de otra forma, Torvalds es el que decide qué entra en uno y en otro sitio (muchas de las decisiones más polémi-

cas las podemos ubicar precisamente aquí). En cualquier caso, Linux no tiene una planificación de entregas con fechas fijas: estará listo cuando esté listo, mientras tanto habrá que esperar. Seguro que el lector habrá asumido a estas alturas que la decisión de si está listo o no corresponde, como no podía ser de otra manera, únicamente a Linus.

El método de desarrollo utilizado en Linux resulta, tal como se ha demostrado, muy eficaz en cuanto a resultados: Linux goza de una gran estabilidad y hay generalmente un intervalo ínfimo de corrección de errores (a veces del orden de los minutos), ya que cuenta con miles de desarrolladores. En esta situación, cuando existe un error, la probabilidad de que uno de ellos lo encuentre es muy alta, y en caso de que no lo pueda resolver el descubridor, ya habrá alguien que dé con la solución de manera rápida. En definitiva, podemos ver cómo Linux cuenta con miles de personas al mes dedicadas a su desarrollo, lo que indudablemente hace que su éxito no sea nada sorprendente.

Se debe mencionar que esta forma de trabajar es, sin embargo, muy cara en cuanto a recursos. No es inusual que existan muchas propuestas mutuamente excluyentes para una nueva funcionalidad o que se reciban una docena de parches para el mismo error. En la gran mayoría de los casos, solamente uno de ellos será incluido en el núcleo finalmente, por lo que se puede considerar que el resto del tiempo y esfuerzo dedicado por los desarrolladores ha sido en balde. El modelo de desarrollo de Linux es, por tanto, un modelo que funciona muy bien en Linux, pero que ciertamente no todos los proyectos se pueden permitir.

9.1.3. Estado actual de Linux

A principios de 2007 Linux se desarrolla sobre la rama 2.6, que ha supuesto (en cuanto a mejoras sobre la rama 2.4) la inclusión de NUMA (acceso a memoria no uniforme, utilizada de manera común en multiprocesadores), nuevos sistemas de ficheros, mejoras para la comunicación en redes inalámbricas y arquitecturas de sonido (ALSA) y otras muchas mejoras (si se está interesado en el detalle de los cambios con respecto a versiones anteriores se puede consultar "The wonderful world of Linux 2.6" [186]).

El modelo de desarrollo de Linux ha sufrido algunos cambios en los últimos años. Aunque la lista de correo de desarrollo sigue siendo el *alma* del proyecto, el código ya no ha de pasar necesariamente por ella. A ello contribuyó en gran manera BitKeeper, un sistema propietario de control de versiones desarrollado por la compañía BitMover siguiendo las estrictas recomendaciones de Linus Torvalds. El uso de una herramienta no libre generó una gran polémica, en la que se pudo constatar otra vez la posición pragmática de Linus, pues para él, y para muchos más, el sistema libre de control de versiones CVS está muy anticuado. La polémica quedó zanjada con el desarrollo de *git*, un sistema de control de versiones distribuido con características similares a BitKeeper y utilizado actualmente en el desarrollo de Linux. En concreto, el proceso de de-

sarrollo de Linux sigue una jerarquía piramidal, en la que los desarrolladores proponen parches, compartidos vía correo entre niveles, que deben ser aceptados por el nivel superior de mantenedores de controladores y de ficheros. En un nivel superior están los mantenedores de subsistemas, y en el nivel más alto, Linus Torvalds y Andrew Morton, que tienen la última palabra para las admisiones de los parches.

A modo de resumen, en la tabla siguiente se ofrece una radiografía del proyecto Linux en la que se ve cómo ha superado los cinco millones de líneas de código, de manera que se puede incluir entre los proyectos libres más grandes (junto con Mozilla y OpenOffice.org). En cuanto a la estimación de tiempo de ejecución y de número medio de desarrolladores que lo harían óptimo, podemos observar que el primero es ciertamente menor que los años de historia con los que cuenta Linux. Esto, por otro lado, queda compensado con creces teniendo en cuenta el segundo, ya que el número medio de desarrolladores a tiempo completo es superior del que dispone Linux.

Nota

La estimación de costes que nos ofrece COCOMO es de 215 millones de dólares americanos, cifra que si ponemos en el contexto de las que se suelen manejar más asiduamente, supone el doble de lo que los grandes clubes de fútbol suelen pagar por una gran estrella.

Tabla 4. Análisis de Linux

Página web	http://www.kernel.org
Inicio del proyecto	Primer mensaje en news.comp.os.minix: agosto 1991
Licencia	GNU GPL
Versión analizada	2.6.20 (versión estable del 20/02/2007)
Líneas de código fuente	5.195.239
Estimación de coste (según COCOMO básico)	$ 215.291.772
Estimación de tiempo de ejecución (según COCOMO básico)	8,83 años (105,91 meses)
Estimación de número medio de desarrolladores (según COCOMO básico)	180,57
Número aproximado de desarrolladores	Se cuentan por miles (aunque en los créditos aparecen sólo centenares [219])
Herramientas de ayuda al desarrollo	Lista de correo y *git*

La composición de Linux en cuanto a lenguajes de programación muestra un claro predominio de C, considerado como un lenguaje ideal para la realización de sistemas críticos en cuanto a velocidad. Cuando la velocidad es un requisito tan estricto que ni siquiera C puede alcanzar, se programa directamente en lenguaje ensamblador, un hecho que como podemos ver, ocurre con cierta frecuencia. El lenguaje ensamblador tiene la desventaja, en comparación con

C, que no es tan portable: cada arquitectura tiene su juego de instrucciones particular, por lo que mucho código escrito para una arquitectura en lenguaje ensamblador ha de ser portado a las demás arquitecturas. La presencia del resto de los lenguajes, como se puede observar en la tabla adjunta, es marginal y se limita a funciones de instalación y utilidades de desarrollo. La versión analizada para este libro ha sido Linux 2.6.20 tal como fue publicada el 20 de febrero de 2007 (sin la aplicación de ningún parche posterior).

Tabla 5. Lenguajes de programación utilizados en Linux

Lenguaje de programación	Líneas de código	Porcentaje
C	4.972.172	95,71%
Ensamblador	210.693	4,06%
Perl	3.224	0,06%
Yacc	2.632	0,05%
Shell	2.203	0,04%

9.2. FreeBSD

Como ya se ha comentado en el capítulo dedicado a la historia del software libre, existen otro tipo de sistemas operativos libres, además del popular GNU/Linux. Una familia de ellos son los "herederos" de las distribuciones de la Universidad de Berkeley, en California (EE.UU.): los sistemas de tipo BSD. El más antiguo y conocido de los sistemas BSD es FreeBSD, cuya historia se remonta a principios de 1993, cuando Bill Jolitz dejó de publicar las actualizaciones no oficiales a 386BSD. Con el apoyo de la empresa Walnut Creek CDROM, que más tarde pasó a llamarse BSDi, un grupo de voluntarios decidió proseguir con el esfuerzo de realizar este sistema operativo libre.

El objetivo principal del proyecto FreeBSD es la creación de un sistema operativo que pueda ser utilizado sin ningún tipo de obligaciones ni ataduras, pero con todas las ventajas de la disponibilidad del código y de un cuidadoso proceso que asegure la calidad del producto. El usuario tiene la libertad de hacer con el software lo que desee, bien modificándolo a su antojo o bien redistribuyéndolo –de forma abierta o incluso cerrada y bajo las condiciones que desee– con o sin modificaciones. Como su propio nombre indica, el proyecto FreeBSD se guía, por tanto, por la filosofía de las licencias BSD.

9.2.1. Historia de FreeBSD

La versión 1.0 apareció a finales de 1993 y estaba basada en 4.3BSD Net/2 y 386BSD. 4.3BSD Net/2 disponía de código procedente de los años setenta, cuando Unix era desarrollado por AT&T, lo que a la postre supuso una serie de problemas legales que no se resolvieron hasta que en 1995 FreeBSD 2.0

fue publicado sin contar con código originario de AT&T, esta vez basándose en 4.4BSD-Lite, una versión *light* de 4.4BSD (en la que se habían suprimido muchos módulos por problemas legales, aparte de que el puerto *–port–* para sistemas Intel todavía estaba incompleto) liberada por la Universidad de California.

La historia de FreeBSD no sería completa si no se contara nada sobre sus distribuciones "hermanas", NetBSD y OpenBSD. NetBSD apareció con una versión 0.8 a mediados de 1993. Su principal objetivo era ser muy portable (aunque en sus comienzos sólo fuera una adaptación para i386), por lo que su lema fue: "Por supuesto que ejecuta NetBSD". OpenBSD surgió de una escisión de NetBSD fundamentada en diferencias filosóficas (y también personales) entre desarrolladores a mediados de 1996. Su principal foco de atención es la seguridad y la criptografía –dicen que son el sistema operativo más seguro que existe–, aunque al basarse en NetBSD también conserva una gran portabilidad.

9.2.2. Desarrollo en FreeBSD

El modelo de desarrollo utilizado por el proyecto FreeBSD está fuertemente basado en dos herramientas: el sistema de versiones CVS y el sistema de informe de error GNATS. En torno a estas dos herramientas gira todo el proyecto, como se puede comprobar por el hecho de que se ha creado una jerarquía a partir de las mismas. Así, sobre los *commiters* (aquellos desarrolladores con derecho a escritura al CVS) es sobre quien recae toda la soberanía del proyecto, bien directamente o bien indirectamente mediante la elección del *core group*, tal como se verá en el apartado siguiente.

Para realizar informes de error en GNATS no hace falta ser *commiter*, por lo que cualquiera que así lo desee podrá notificar una errata. Todas las contribuciones abiertas (*open*) en GNATS son evaluadas por un *commiter*, que podrá asignar (*analysed*) la tarea a otro *commiter* o pedir más información a la persona que realizó el informe original (*feedback*). Hay situaciones en las que la errata ha sido solucionada para algunas ramas recientes, lo que se especifica con el estado "suspendida" (*suspended*). En cualquier caso, la meta es que el informe se cierre (*closed*) tras haber corregido el error.

FreeBSD distribuye su software de dos formas: por un lado, los puertos, un sistema que descarga los códigos fuente, los compila e instala la aplicación en el ordenador local, y por otro, los paquetes, que no son más que los códigos fuente de los puertos precompilados y, por lo tanto, en binario. La ventaja más importante de los puertos sobre los paquetes es que los primeros permiten al usuario configurar y optimizar el software para su ordenador. En cambio, el sistema de paquetes, al estar ya precompilados, permite instalar el software de una manera más rápida.

9.2.3. Toma de decisiones en FreeBSD

El consejo de directores de FreeBSD, conocido popularmente como *core team*, se encarga de marcar la dirección del proyecto y de velar porque se cumplan sus objetivos, así como de mediar en caso de que haya conflictos entre *commiters*. Hasta octubre de 2000 era un grupo cerrado al que sólo se entraba a formar parte por invitación expresa del propio *core team*. A partir de entonces, sus miembros son elegidos periódica y democráticamente por los *commiters*. La normativa más importante para la elección del *core team* es la siguiente:

1) Podrán votar los *commiters* que hayan realizado al menos un *commit* en el último año.

2) El Consejo de Directores se renovará cada dos años.

3) Los miembros del consejo de directores podrán ser "expulsados" con el voto de dos tercios de los *commiters*.

4) Si el número de miembros del consejo de directores es menor de siete, se celebrarán nuevas elecciones.

5) Se celebrarán nuevas elecciones si así lo pide un tercio de los *commiters*.

6) El cambio de normativa requiere un quórum de dos tercios de los *commiters*.

9.2.4. Empresas en torno a FreeBSD

Existen multitud de empresas que ofrecen servicios y productos basados en FreeBSD, de las cuales FreeBSD lleva buena cuenta en su página del proyecto. En esta presentación de FreeBSD conoceremos un poco más a fondo las más significativas: BSDi y Walnut Creek CDROM.

FreeBSD nació en parte debido a la fundación en 1991 de la compañía BSDi por gente del CSRG (Computer Systems Research Group) de la Universidad de Berkeley, compañía que se dedicaría al soporte comercial para el nuevo sistema operativo. Además de la versión comercial del sistema operativo FreeBSD, BSDi también desarrolló otros productos, como un servidor de Internet y de pasarela.

Walnut Creek CDROM nació con el objetivo de comercializar FreeBSD como producto final, de manera que se pudiera considerar como una distribución al estilo de las que existen para GNU/Linux, pero con FreeBSD. En noviembre de 1998, Walnut Creek amplió sus horizontes con la creación del portal FreeBSD Mall, que se dedicaría a comercializar todo tipo de productos sobre

FreeBSD (desde la distribución en sí misma hasta camisetas, revistas, libros, etc.), a anunciar productos de terceros en su página web y a dar soporte profesional de FreeBSD.

En marzo de 2000, BSDi y Walnut Creek se unieron bajo el nombre BSDi para hacer frente de manera conjunta al fenómeno Linux, que estaba dejando claramente en la sombra a los sistemas BSD en general y a FreeBSD en particular. Un año más tarde, en mayo de 2001, Wind River adquirió la parte dedicada a la generación de software de BSDi, con la clara intención de potenciar el desarrollo de FreeBSD para su uso en sistemas empotrados y dispositivos inteligentes conectados a la Red.

9.2.5. Estado actual de FreeBSD

Según los últimos datos de la encuesta que realiza periódicamente Netcraft, el número de servidores web que ejecutan FreeBSD se acerca a los dos millones de unidades. Un nuevo usuario que quisiera instalarse FreeBSD podría elegir entre la versión 6.2 (que se podría considerar como la versión "estable") o la rama más avanzada o "de desarrollo". Mientras que la primera ofrece mayor estabilidad –sobre todo en áreas como el multiprocesamiento simétrico, que han sido totalmente reelaboradas en las nuevas versiones–, la segunda permite disfrutar de las últimas novedades. También es importante tener en cuenta que las versiones de desarrollo suelen incluir código de pruebas, lo que hace que la velocidad del sistema se vea afectada sensiblemente.

Una de las utilidades estrella de FreeBSD son las llamadas *cárceles* (*jail*, en inglés). Las cárceles permiten minimizar el daño causado por un ataque a servicios de red básicos, como podrían ser Sendmail para el correo electrónico o BIND (Berkeley Internet Name Domain) para gestionar los nombres. Los servicios son introducidos en una cárcel para que se ejecuten en un entorno de aislamiento. La gestión de las cárceles se puede realizar mediante una serie de utilidades incluidas en FreeBSD.

9.2.6. Radiografía de FreeBSD

Como hemos ido comentando a lo largo de este último apartado, la labor de BSD no se restringe únicamente al desarrollo de un núcleo del sistema operativo, sino que también incluye la integración de multitud de utilidades que se distribuyen conjuntamente al estilo de las distribuciones de GNU/Linux. El hecho de que el proceso de desarrollo de FreeBSD esté íntimamente ligado al sistema de control de versiones CVS hace que el estudio del mismo nos pueda dar una buena aproximación de todo lo que contiene. Las cifras que se muestran a continuación son las correspondientes al análisis de FreeBSD efectuado el 21 de agosto de 2003.

Uno de los aspectos más interesantes de FreeBSD es que sus números se asemejan mucho a los que ya hemos podido observar en KDE y en GNOME: el tamaño del software supera ampliamente los cinco millones de líneas de código, el número de ficheros ronda los 250.000 y el número total de *commits* se sitúa en torno a los dos millones. Sin embargo, es interesante observar que la principal diferencia entre GNOME y KDE con respecto a FreeBSD es la *edad* del proyecto. FreeBSD acaba de cumplir recientemente la década de existencia y casi dobla en tiempo a los entornos de escritorio con los que la estamos comparando. Que el tamaño sea similar, aun cuando el tiempo de desarrollo ha sido superior, se debe en gran parte a que el número de desarrolladores que ha atraído FreeBSD es más pequeño. Con derecho de escritura en el CVS (*commiter*) hay listados unos cuatrocientos, mientras que los colaboradores que se mencionan en el manual de FreeBSD son cerca de un millar. Por eso la actividad que registra el CVS de FreeBSD es menor en media (quinientos *commits* diarios) que la que registraban tanto GNOME (novecientos) como KDE (mil setecientos contando los *commits* automáticos).

Hemos considerado como sistema básico de FreeBSD todo aquello que cuelga del directorio *src/src* del módulo *root* del CVS. La actividad que ha ido registrando el sistema básico a lo largo de los últimos diez años es de más de medio millón de *commits*. Su tamaño supera los cinco millones de líneas de código, aunque hay que comentar que en él no sólo está incluido el núcleo, sino multitud de utilidades adicionales, incluso juegos. Si tenemos en cuenta sólo el núcleo (que se encuentra bajo el subdirectorio *sys*), su tamaño es de 1,5 millones de líneas de código fuente, predominantemente en C.

Resulta interesante ver cómo la estimación de tiempo dada por COCOMO concuerda a la perfección con el tiempo real del proyecto FreeBSD, aunque la estimación del número medio de desarrolladores es, en mucho, mayor que la real. Hay que decir que en el último año sólo unos setenta y cinco *commiters* han estado activos, mientras que COCOMO supone que durante los diez años de desarrollo el número de desarrolladores debería ser de 235.

Por último, cabe destacar, como se ha dicho anteriormente, que la actividad principal de FreeBSD se sitúa en torno al CVS y al sistema de control de erratas y actividades GNATS.

Tabla 6. Análisis de FreeBSD

Página web	http://www.FreeBSD.org
Inicio del proyecto	1993
Licencia	De tipo BSD
Versión analizada	4.8
Líneas de código fuente	7.750.000

Líneas de código fuente (sólo núcleo)	1.500.000
Número de ficheros	250.000
Estimación de coste	$ 325.000.000
Estimación de tiempo de ejecución	10,5 años (126 meses)
Estimación de número medio de desarrolladores	235
Número aproximado de desarrolladores	400 *commiters* (1.000 colaboradores)
Número de *commiters* activos en el último año	75 (menos 20% del total)
Número de *commiters* activos en los dos últimos años	165 (alrededor del 40% del total)
Número de *commits* en el CVS	2.000.000
Número medio de *commits* (totales) al día	Unos 500
Herramientas de ayuda al desarrollo	CVS, GNATS, listas de correo y sitio de noticias

C es el lenguaje predominante en FreeBSD, y guarda una distancia con respecto a C++ superior a la de los demás casos que hemos estudiado en este capítulo. Es interesante observar que el número de líneas de código de lenguaje ensamblador contenidas en FreeBSD concuerda en el orden de magnitud con las que tiene Linux, aunque las correspondientes al núcleo sólo son unas veinticinco mil en total. En resumen, se podría decir que en FreeBSD lo que manda son los lenguajes más *clásicos* dentro del software libre –C, Shell y Perl– y que la penetración de los lenguajes que hemos observado en otras aplicaciones y proyectos –C++, Java, Python...– no ha tenido lugar.

Tabla 7. Lenguajes de programación utilizados en FreeBSD

Lenguaje de programación	Líneas de código	Porcentaje
C	7.080.000	92,0%
Shell	205.000	2,7%
C++	131.500	1,7%
Ensamblador	116.000	1,5%
Perl	90.900	1,20%
Yacc	5.800	0,75%

9.2.7. Estudios académicos sobre FreeBSD

Aun siendo ciertamente un proyecto muy interesante (podemos obtener toda su historia –¡desde hace 10 años!– mediante el análisis del sistema de versiones), la atención que ha despertado FreeBSD entre la comunidad científica ha sido más bien pequeña. De esta falta de interés se salva un equipo de investigación que ha estudiado el proyecto FreeBSD desde varios puntos de vista ("Incremental and decentralized integration in FreeBSD") [149], y que ha puesto especial atención en cómo se resuelven los problemas de la integración de software de manera incremental y descentralizada.

9.3. KDE

Aunque con probabilidad no fue la primera solución en cuanto a entornos de escritorio *amigables* para el usuario, la difusión a mediados de 1995 del sistema operativo Windows 95™ supuso un cambio radical en la interacción de los usuarios de a pie con los ordenadores. De los sistemas unidimensionales de línea de instrucciones (los terminales), se pasó a la metáfora de entorno del escritorio bidimensional, donde el ratón ganó terreno al teclado. Windows 95™, más que una innovación tecnológica, debe ser acreditado como el sistema que consiguió adentrarse en todos los entornos personales y de oficina, marcando las pautas a seguir (reglas técnicas y sociales que, a principios del siglo XXI, a veces todavía seguimos padeciendo).

Con anterioridad a los sistemas de escritorio, cada aplicación gestionaba la apariencia y la forma de interactuar con el usuario de manera autónoma. En los escritorios, por el contrario, las aplicaciones han de contar con propiedades comunes y un aspecto compartido entre aplicaciones de forma que esto suponga un alivio para el usuario, que puede *reutilizar la interacción* aprendida de una aplicación a otra. También resultó ser un alivio para los desarrolladores de aplicaciones, ya que no se tenían que enfrentar con el problema de crear los elementos interactivos desde cero (una tarea siempre complicada), sino que podían partir de un marco y unas reglas predefinidas.

9.3.1. Historia de KDE

Los seguidores de Unix rápidamente se hicieron eco del notable éxito de Windows 95 y, a la vista de que los entornos Unix carecían de sistemas tan intuitivos a la vez que libres, decidieron ponerse manos a la obra. Fruto de esta preocupación nació en 1996 el proyecto KDE –K Desktop Environment–, ideado por Matthias Ettrich (creador de LyX, un programa de edición en modo gráfico de TeX) y otros *hackers*. El proyecto KDE se planteó los objetivos siguientes:

Nota

Originariamente el nombre KDE significaba Kool Desktop Environment, pero con el tiempo se decidió que pasara a llamarse simplemente K Desktop Environment. La explicación oficial fue que la letra K es la que precede en el alfabeto latino a la L de Linux.

- Dotar a los sistemas Unix de un entorno amigable que fuera al mismo tiempo abierto, estable, de confianza y poderoso.

- Desarrollar un conjunto de bibliotecas para escribir aplicaciones estándar sobre el sistema gráfico para Unix X11.

- Crear una serie de aplicaciones que permitieran al usuario acometer sus objetivos de manera eficaz.

Cuando los integrantes del recién creado proyecto KDE decidieron utilizar una biblioteca orientada a objetos llamada Qt, propiedad de la firma noruega Trolltech™, que no estaba amparada bajo una licencia de software libre, surgió una gran polémica. Se daba la circunstancia de que, a pesar de que las aplicaciones de KDE estaban licenciadas bajo la GPL, enlazaban con esta biblioteca, de manera que se hacía imposible su redistribución. Consecuentemente, se estaba violando una de las cuatro libertades del software libre enunciadas por Richard Stallman en el Manifiesto del Software Libre [117]. Desde la versión 2.0, Trolltech distribuye Qt bajo una licencia dual que especifica que si la aplicación que hace uso de la biblioteca es GPL, entonces la licencia válida para Qt es la GPL. Gracias a ello, uno de los debates más calientes y airados dentro del mundo del software libre tuvo, por suerte, un final feliz.

9.3.2. Desarrollo de KDE

KDE es de los pocos proyectos de software libre que cumplen con un calendario de lanzamiento de nuevas versiones de manera generalizada (recordemos, por ejemplo, que habrá una nueva versión de Linux "cuando esté lista", mientras que, como veremos más adelante, en el caso de GNOME se han dado retrasos significativos a la hora de publicar nuevas versiones). La numeración de las versiones sigue una política perfectamente definida. Las versiones de KDE constan de tres números de versión: uno mayor y dos menores. Por ejemplo, en KDE 3.1.2, el número mayor sería el 3, y los menores, el 1 y el 2. Versiones con un mismo número mayor cuentan con compatibilidad binaria, por lo que no hace falta recompilar las aplicaciones. Hasta ahora, los cambios en el número mayor se han venido dando paralelamente con cambios en la biblioteca Qt, por lo que se puede ver que los desarrolladores quisieron aprovechar las nuevas funcionalidades de la biblioteca Qt en la versión inminente de KDE.

En cuanto a los números menores, las versiones con un único número menor son versiones en las que se han incluido tanto nuevas funcionalidades como corrección de las erratas encontradas. Las versiones con un segundo número menor no incluyen nuevas funcionalidades sobre las versiones con primer número menor, y sólo contienen corrección de errores. Para aclararlo con un ejemplo: KDE 3.1 es una versión de la tercera generación de KDE (número mayor 3) a la que se le han añadido nuevas funcionalidades, mientras que KDE 3.1.1 es la versión anterior –con las mismas funcionalidades–, pero con las erratas que se han encontrado corregidas.

KDE se constituyó, poco después de empezar, en una asociación registrada en Alemania (KDE e.V.) y, como tal, tiene unos estatutos que la obligan a contar con un consejo directivo. La influencia de este consejo directivo sobre el desarrollo es nula, ya que su tarea es fundamentalmente la administración de la asociación, en especial de las donaciones que percibe el proyecto. Para la promoción y la difusión de KDE –que incluye a empresas interesadas– se creó la Liga KDE, de la que hablaremos a continuación.

9.3.3. La Liga KDE

La Liga KDE (KDE League, en su denominación original en inglés) es un grupo de empresas y de particulares de KDE que tiene el objetivo de facilitar la promoción, la distribución y el desarrollo de KDE. Las empresas y los particulares que participan en la Liga KDE no tienen por qué estar directamente involucrados en el desarrollo de KDE (aunque se anima a todos los miembros a hacerlo), sino que simplemente representan un marco industrial y social amigo de KDE. Los objetivos de la Liga KDE son los siguientes:

- Promover, proveer y facilitar la educación formal e informal de las funcionalidades, las capacidades y otras cualidades de KDE.
- Animar a corporaciones, gobiernos, empresas e individuos a usar KDE.
- Animar a corporaciones, gobiernos, empresas e individuos a participar en el desarrollo de KDE.
- Proveer de conocimientos, información, dirección y posicionamiento alrededor de KDE en cuanto a su uso y su desarrollo.
- Promocionar la comunicación y la cooperación entre los desarrolladores de KDE.
- Promocionar la comunicación y la cooperación entre los desarrolladores de KDE y el público mediante publicaciones, artículos, sitios web, encuentros, participación en congresos y exposiciones, notas de prensa, entrevistas, material promocional y comités.

Las empresas que participan en la KDE League son principalmente distribuciones (SuSE, ahora parte de Novell, Mandriva, TurboLinux, Lindows y Hancom, una distribución de software libre coreana), empresas de desarrollo (Trolltech y Klarälvdalens Datakonsult AB), además del gigante IBM y de una empresa creada con la finalidad de promocionar KDE (KDE.com). De entre todas ellas, cabe destacar por su implicación fundamental Trolltech, Novell y Mandriva Software, cuyo modelo de negocio está íntimamente ligado al proyecto KDE:

- Trolltech es una compañía noruega afincada en Oslo que desarrolla Qt, la biblioteca que hace las veces de interfaz gráfica de usuario y API para

el desarrollo de aplicaciones, aunque también puede funcionar como elemento empotrado en PDA (como por ejemplo en los Sharp Zaurus). La importancia del proyecto KDE en Trolltech se puede constatar es dos elementos básicos de su estrategia comercial: por un lado, reconoce a KDE como su principal forma de promoción, alentando el desarrollo del escritorio y aceptando e implementando las mejoras o las modificaciones propuestas; por otro lado, algunos de los desarrolladores más importantes de KDE trabajan profesionalmente para Trolltech –el caso más conocido es el del propio Mathias Ettrich, fundador del proyecto–, lo que sin duda beneficia tanto al proyecto KDE como a la propia compañía. La implicación de Trolltech en el proyecto KDE no se limita exclusivamente a la biblioteca Qt, como se puede ver por el hecho de que uno de los desarrolladores principales de KOffice –el paquete ofimático de KDE– tenga en la actualidad un contrato a tiempo parcial con ellos.

- SuSE (ahora parte de Novell) siempre ha demostrado una especial predilección por el sistema de escritorio KDE, en parte debido a que una gran mayoría de sus desarrolladores son de origen alemán o centroeuropeo, al igual que la propia compañía. Conocedora del hecho de que cuanto mejor y más fácil sea el entorno de escritorio que ofrezca su distribución, mayor será su implantación y, por tanto, las ventas y la petición de soporte, SuSE ha tenido siempre una política muy activa en cuanto a la dedicación de presupuesto para profesionalizar posiciones clave dentro del proyecto KDE. De esta manera, en la actualidad, el administrador del sistema de control de versiones y otro par de desarrolladores principales tienen una nómina de SuSE. Asimismo, dentro de la plantilla de SuSE hay una docena de desarrolladores que pueden dedicar parte de su tiempo laboral al desarrollo de KDE.

- La distribución Mandriva es otro de los grandes benefactores de KDE, y cuenta en su plantilla con varios de los desarrolladores principales. Su situación económica en 2003 –con suspensión de pagos incluida– ha hecho que en los últimos tiempos haya perdido influencia.

9.3.4. Estado actual de KDE

Tras la publicación de KDE 3 en mayo de 2002, la opinión generalizada es que los escritorios libres se encuentran a la altura de sus competidores propietarios. Entre sus grandes logros se encuentra la incorporación de un sistema de componentes (KParts) que permite empotrar unas aplicaciones en otras (un trozo de hoja de cálculo de KSpread en el procesador de textos KWord) y el desarrollo de DCOP, un sistema de comunicación entre procesos simple y con autenticación. DCOP fue la apuesta del proyecto en detrimento de las tecnologías CORBA, un tema de amplio debate dentro de los escritorios libres, en especial entre GNOME –que se decidió a utilizar tecnologías CORBA– y KDE. La historia parece haber puesto a cada tecnología en su sitio, como se puede

ver con la propuesta de DBUS (una especie de DCOP mejorado) por parte del FreeDesktop.org, un proyecto interesado en fomentar la interoperabilidad y el uso de tecnologías conjuntas en los escritorios libres, que casualmente está liderado por uno de los *hackers* de GNOME más reconocidos.

La tabla resumen siguiente contiene las características más importantes del proyecto KDE. Las licencias que acepta el proyecto dependen de si se trata de una aplicación o de una biblioteca. Las licencias de las bibliotecas permiten una mayor "flexibilidad" a terceros; en otras palabras, posibilitan que terceros puedan crear aplicaciones propietarias que enlacen con las bibliotecas.

La última versión de KDE es, a principios de 2007, la 3.5.6, y para mediados de año está prevista la cuarta generación, KDE 4, que se basará en Qt4. Los cambios de generación suponen un gran esfuerzo de adaptación, una tarea tediosa y costosa en tiempo. Sin embargo, esto no ha de suponer que las aplicaciones "antiguas" dejen de funcionar. Generalmente, para que sigan funcionando se incluyen también las antiguas versiones de las bibliotecas sobre las que se basan, aunque ello signifique tener que cargar varias versiones de las bibliotecas en memoria simultáneamente, con el consiguiente desperdicio de recursos del sistema. Este hecho es visto por los desarrolladores de KDE como algo inherente a la propia evolución del proyecto y, por tanto, como un mal menor.

9.3.5. Radiografía de KDE

En cuanto al tamaño de KDE, las cifras que se van a mostrar a continuación corresponden al estado del CVS en agosto de 2003, por lo que han de tomarse con las precauciones tradicionales que ya se han comentado y una más: alguno de los módulos que se han considerado en este estudio todavía se encuentra en fase de desarrollo y no cumple el requisito de ser un producto acabado. Esto no nos ha de molestar en absoluto para nuestros propósitos, ya que estamos más interesados en el orden de magnitud de los resultados que en la cifra exacta.

El código fuente incluido en el CVS de KDE suma en total más de seis millones de líneas de código en diferentes lenguajes de programación, tal como mostraremos más adelante. El tiempo que se necesitaría para crear KDE rondaría los nueve años y medio, una cifra superior a los siete años que tiene el proyecto, y el número medio estimado de desarrolladores a tiempo completo rondaría los doscientos. Si tenemos en cuenta que KDE cuenta con unas ochocientas personas con acceso de escritura en el CVS en 2003 (de las cuales la mitad han estado inactivas en los últimos dos años) y que el número de desarrolladores de KDE contratados a tiempo completo no ha superado en ningún momento la veintena, podemos ver que la productividad de KDE es, en mucho, superior a la estimación que ofrece COCOMO.

Nota

Una empresa que quisiera desarrollar un producto de ese tamaño desde cero necesitaría invertir más de 250 millones de dólares, una cifra que a modo de comparación es lo que invirtió una firma de automóviles en la creación de una nueva planta de producción en el este de Europa o lo que una conocida petrolera planea gastar para abrir doscientas gasolineras en España.

Es interesante ver que una gran parte del esfuerzo –casi la mitad que el de desarrollo– del proyecto KDE lo podemos situar en la traducción de la interfaz de usuario y de la documentación. Aunque muy pocas (unas miles) de las líneas de programación se dedican a esta labor, el número de ficheros dedicados a este cometido asciende a los setenta y cinco mil para traducciones (cifra que se eleva hasta los cien mil si incluimos la documentación en sus diferentes formatos), lo que viene a suponer casi la cuarta (tercera) parte de los 310.000 ficheros que hay en el CVS. La actividad conjunta del CVS es de mil doscientos *commits* diarios, por lo que el tiempo medio entre *commits* es de cerca de un minuto[10].

[10]Hay que hacer dos observaciones a este resultado: la primera es que se considera que un *commit* que incluya varios ficheros es como si se hubiera hecho un *commit* por separado para cada fichero; la segunda es que la cifra de *commits* es una cifra estimada, ya que el proyecto dispone de una serie de *scripts* que realizan *commits* de manera automática.

En cuanto a las herramientas, los lugares de información y los eventos de ayuda al desarrollo, vemos que el abanico de posibilidades que ofrece KDE es mucho más amplio que el usado en Linux. Además del sistema de control de versiones y de las listas de correo, KDE cuenta con una serie de sitios web donde se puede encontrar información y documentación técnica y no técnica del proyecto. Entre estos sitios también existe un sitio de noticias donde se presentan nuevas soluciones y se debaten propuestas. El sitio de noticias, sin embargo, no puede considerarse como sustituto de las listas de correo –que, al igual que en Linux, es donde se encuentran los verdaderos debates y la toma de decisiones y estrategias de futuro–, sino como un punto de encuentro con los usuarios. Por último, KDE organiza desde hace tres años reuniones periódicas en las que los desarrolladores y los colaboradores se reúnen durante alrededor de una semana para presentar las últimas novedades, desarrollar, debatir, conocerse y pasárselo bien (no necesariamente en ese orden).

Tabla 8. Análisis de KDE

Página web	http://www.kde.org
Inicio del proyecto	1996
Licencia (para aplicaciones)	GPL, QPL, MIT, Artistic
Licencia (para bibliotecas)	LGPL, BSD, X11
Versión analizada	3.1.3
Líneas de código fuente	6.100.000
Número de ficheros (código, documentación, etc.)	310.000 ficheros
Estimación de coste	$ 255.000.000

Estimación de tiempo de ejecución	9,41 años (112,98 meses)
Estimación de número medio de desarrolladores	200,64
Número aproximado de desarrolladores	Unos 900 *commiters*
Número de *commiters* activos en el último año	Alrededor de 450 (aproximadamente el 50% del total)
Número de *commiters* activos en los dos últimos años	Unos 600 (aproximadamente el 65% del total)
Número aproximado de traductores (activos)	Unos 300 traductores para más de 50 lenguas (incluido el esperanto)
Número de *commits* (de desarrolladores) en el CVS	Aproximadamente 2.000.000 (cifra estimada sin *commits* automáticos)
Número de *commits* (de traductores) en el CVS	Aproximadamente 1.000.000 (cifra estimada sin *commits* automáticos)
Número medio de *commits* (totales) al día	1.700
Herramientas, documentación y eventos de ayuda al desarrollo	CVS, listas de correo, sitio web, sitio de noticias, reuniones anuales

En cuanto a los lenguajes de programación utilizados en KDE, predomina el uso de C++. Esto se debe principalmente a que éste es el lenguaje nativo de Qt, aunque se lleva a cabo un gran esfuerzo por realizar enlaces para permitir el desarrollo en otros lenguajes de programación. Ciertamente, el número de líneas de código en los lenguajes minoritarios corresponde casi íntegramente al propio proyecto de creación del enlace, aunque esto no quiere decir que no se utilicen en absoluto, ya que existen multitud de proyectos externos a KDE que los utilizan.

Tabla 9. Lenguajes de programación utilizados en KDE

Lenguaje de programación	Líneas de código	Porcentaje
C++	5.011.288	82,05%
C	575.237	9,42%
Objective C	144.415	2,36%
Shell	103.132	1,69%
Java	87.974	1,44%
Perl	85.869	1,41%

9.4. GNOME

El proyecto GNOME tiene como principal objetivo crear un sistema de escritorio para el usuario final que sea completo, libre y fácil de usar. Asimismo, pretende que GNOME sea una plataforma muy potente de cara al desarrollador.

GNOME es el acrónimo en inglés de *GNU Network Object Model Environment*. Desde los inicios de GNOME se han propuesto varias formas de traducirlo al castellano, pero no se ha encontrado ninguna que haya satisfecho a todos. Sin embargo, de su nombre podemos ver que GNOME es parte del proyecto GNU. En la actualidad, todo el código contenido en GNOME debe estar bajo licencia GNU GPL o GNU LGPL. También vemos que las redes y el modelado orientado a objetos tienen una importancia capital.

9.4.1. Historia de GNOME

Mientras se seguía discutiendo acerca de la libertad de KDE, la historia quiso que en el verano de 1997, Miguel de Icaza y Nat Friedman coincidieran en Redmond en unas jornadas organizadas por Microsoft™. Es probable que este encuentro propiciara en ambos un giro radical que supuso tanto la creación de GNOME por parte de Miguel de Icaza a su vuelta a México (junto con Federico Mena Quintero), como su admiración por las tecnologías de objetos distribuidos. De Icaza y Mena decidieron crear un entorno alternativo a KDE, ya que consideraron que una reimplementación de una biblioteca propietaria hubiera sido una tarea destinada a fracasar. Había nacido GNOME.

Desde aquellos tiempos lejanos de 1997 hasta la actualidad, GNOME ha ido creciendo paulatinamente con sus reiteradas publicaciones. En noviembre de 1998 se lanzó la versión 0.99, pero la primera realmente popular, distribuida prácticamente por cualquier distribución de GNU/Linux, sería GNOME 1.0, de marzo de 1999. Cabe destacar que la experiencia de esta primera versión *estable* de GNOME no fue muy satisfactoria, ya que muchos la consideraron llena de erratas críticas. Por eso, GNOME October (GNOME 1.0.55) es tratada como la primera versión del entorno de escritorio GNOME realmente estable. Como se puede observar, con GNOME October se intentó evitar versiones de publicación numeradas para no entrar en una "carrera" de versiones con KDE. La realización de la primera GUADEC, la conferencia de desarrolladores y usuarios europeos de GNOME, celebrada en París en el año 2000, no coincidió en el tiempo por poco con la publicación de una nueva versión de GNOME, llamada GNOME April. Fue la última que llevó un mes como nombre de publicación, ya que se mostró que ese sistema causaba más confusión que otra cosa (por ejemplo, GNOME April es posterior a GNOME October, aunque el sentido común nos haría pensar lo contrario). En octubre de ese año, tras ser debatida durante meses en diferentes listas de correo, se creó la Fundación GNOME, que será presentada más adelante.

GNOME 1.2 fue un paso adelante en cuanto a la arquitectura utilizada por GNOME, arquitectura que se siguió usando en GNOME 1.4. Esta época estuvo caracterizada por la segunda edición de la GUADEC, esta vez en Copenhague. Lo que había empezado siendo una reunión minoritaria de algunos *hackers*, acabó convirtiéndose en un gran evento que atrajo miradas de toda la industria del software.

Mientras tanto, el litigio sobre la libertad de KDE se resolvió con el cambio de postura de Trolltech™, que terminó licenciando Qt bajo una licencia dual, que era de software libre para las aplicaciones que son software libre. Hoy en día no cabe ninguna duda de que tanto GNOME como KDE son entornos de escritorio libres, por lo que podemos considerar que el desarrollo de GNOME ha propiciado el hecho de no tener un sólo entorno de escritorio libre, sino dos.

9.4.2. La Fundación GNOME

El problema más difícil de abordar cuando se oye hablar de GNOME por primera vez es la organización de los más de mil contribuyentes al proyecto. Resulta paradójico que un proyecto cuya estructura es más bien anárquica, llegue a fructificar y saque adelante unos objetivos complejos y al alcance de pocas multinacionales del sector de la informática.

Aunque GNOME nació con una clara intención de realizar un entorno amigable y potente al que se iban añadiendo nuevos programas, pronto se vio la necesidad de crear un órgano que tuviera ciertas competencias que permitieran potenciar el uso, el desarrollo y la difusión de GNOME: de esta forma, en octubre de 2000 se dio paso a la creación de la Fundación GNOME, cuya sede se encuentra en Boston, EE.UU.

La Fundación GNOME es una organización sin fines de lucro, no un consorcio industrial, que tiene las funciones siguientes:

- Coordina las publicaciones.
- Decide qué proyectos son parte de GNOME.
- Es la voz oficial (para la prensa y para organizaciones tanto comerciales como no comerciales) del proyecto GNOME.
- Patrocina conferencias relacionadas con GNOME (como la GUADEC).
- Representa a GNOME en otras conferencias.
- Crea estándares técnicos.
- Promueve el uso y el desarrollo de GNOME.

Además, la Fundación GNOME permite la recepción de fondos económicos para patrocinar e impulsar las funciones antes mencionadas, hecho que antes de su creación era imposible realizar de manera transparente.

En la actualidad, la Fundación GNOME cuenta con un empleado a tiempo completo que se encarga de solventar todos los trabajos burocráticos y organizativos que se dan en una organización sin fines de lucro que realiza reuniones y conferencias de manera periódica.

En términos generales, la Fundación GNOME se estructura en dos grandes consejos: un consejo directivo y un consejo consultor.

El consejo directivo (*Board of Directors*) está integrado, a lo sumo, por catorce miembros elegidos democráticamente por los miembros de la Fundación GNOME. Se sigue un modelo "meritocrático", lo que quiere decir que para ser miembro de la Fundación GNOME se debe de haber colaborado de alguna u otra manera con el proyecto GNOME. La aportación no tiene por qué ser necesariamente código fuente; también existen tareas de traducción, organización, difusión, etc., por las que uno puede pedir ser miembro de la Fundación GNOME y tener derecho a voto. Por tanto, son los miembros de la Fundación los que se pueden presentar al consejo directivo y los que, democráticamente, eligen a sus representantes en el mismo de entre los que se hayan presentado. En la actualidad, la votación se lleva a cabo por correo electrónico. La duración del cargo como consejero director es de un año, período tras el cual se vuelven a convocar elecciones.

Existen unas normas básicas para garantizar la transparencia del consejo directivo. La más llamativa es la limitación de miembros afiliados a una misma empresa, la cual no puede exceder de cuatro empleados. Es importante hacer hincapié en que los miembros del consejo directivo lo son siempre a nivel personal, y nunca en representación de una compañía. Aun así, y después de una larga discusión, se aceptó incluir esta cláusula para evitar suspicacias.

El otro consejo dentro de la Fundación GNOME es el consejo consultor, órgano sin capacidad de decisión que sirve como vehículo de comunicación con el consejo directivo. Está compuesto por compañías comerciales de la industria del software, así como por organizaciones no comerciales. En la actualidad sus miembros incluyen a Red Hat, Novell, Hewlett-Packard, Mandrake, SUN Microsystems, Red Flag Linux, Wipro, Debian y la Free Software Foundation. Para formar parte del consejo de consultores se exige una cuota a todas las empresas con más de diez empleados.

9.4.3. La industria en torno a GNOME

GNOME ha conseguido adentrarse de manera sustancial en la industria, de manera que varias empresas han participado muy activamente en su desarrollo. De todas ellas, los casos más significativos son los de Ximian Inc., Eazel, los RHAD Labs de Red Hat y, más recientemente, SUN Microsystems. A continuación, se describen, para cada caso, tanto las motivaciones de las compañías, como sus aportaciones más importantes al entorno de escritorio GNOME:

- Ximian Inc. (en sus comienzos Helix Inc.) es el nombre de la empresa que fundaron en 1999 Miguel de Icaza, cofundador de GNOME, y Nat Friedman, uno de los *hackers* de GNOME. Su cometido principal era reunir bajo un mismo paraguas a los desarrolladores más importantes de GNOME para potenciar su desarrollo, por lo que no es de extrañar que cuente o que haya contado entre sus empleados con una veintena larga de los desarrolladores más activos de GNOME. La aplicación en la que Ximian puso mayor empeño desde sus comienzos fue Evolution, un completo sistema de gestión de información personal al estilo de Microsoft Outlook que incluía cliente de correo electrónico, agenda y directorio de contactos. Los productos que Ximian comercializaba son el Ximian Desktop (una versión de GNOME con fines más corporativos), Red Carpet (principalmente, aunque no exclusivamente, un sistema de distribución de software de GNOME) y finalmente MONO (una reimplementación de la plataforma de desarrollo .NET), aunque este último proyecto, por ahora, no tenga nada que ver con GNOME. Ximian también ha desarrollado una aplicación que sirve para que Evolution interactúe con un servidor Exchange 2000. Esta aplicación, aunque bastante pequeña, fue muy polémica porque se publicó con una licencia no libre (posteriormente, en 2004, este componente pasó también a licenciarse como software libre). En agosto de 2003 Novell, como parte de su estrategia para entrar en el escritorio de GNU/Linux, compró Ximian.

- Eazel fue fundada en 1999 por un grupo de personas proveniente de Apple con la finalidad de hacer el escritorio en GNU/Linux tan fácil como lo es en Macintosh. La aplicación en la que centraron su esfuerzo recibió el nombre de Nautilus y debía ser el gestor de ficheros que jubilara al mítico Midnight Commander, desarrollado por Miguel de Icaza. Su falta de modelo de negocio y la crisis de las puntocom –los inversores de riesgo retiraron el capital necesario para que la empresa pudiera seguir funcionando– provocó que el 15 de mayo de 2001 Eazel se declarara en quiebra y cerrara sus puertas. Todavía tuvo tiempo para publicar la versión 1.0 de Nautilus, aunque su numeración fuera más bien artificial, ya que la estabilidad que se presupone a una versión 1.0 no aparecía por ningún lado. Dos años después de la quiebra de Eazel, se pudo ver cómo Nautilus había evolucionado y se había convertido en un completo y manejable gestor de ficheros integrado en GNOME, por lo que la historia de Eazel y Nautilus se puede considerar como un caso paradigmático de un programa que sobrevive a la desaparición de la empresa que lo creaba –algo casi sólo posible en el mundo del software libre.

- Red Hat creó los Red Had Advanced Development Labs ('laboratorios de desarrollo avanzado de Red Hat'), RHAD, con la intención de que el escritorio GNOME ganara en usabilidad y potencia. Para ello contrató a media docena de los *hackers* más importantes de GNOME y les dio libertad para desarrollar en lo que ellos decidieran que era conveniente. De los RHAD Labs salió ORBit, la implementación de CORBA utilizada por el proyecto GNOME, conocida como "la más rápida del oeste". También es destacable

la labor que se hizo en la nueva versión de GTK+ y en el sistema de configuración de GNOME, GConf.

- SUN Microsystems se involucró tardíamente en el desarrollo de GNOME, ya que para septiembre de 2000 GNOME era ya un producto relativamente maduro. La intención de SUN era utilizar GNOME como el sistema de escritorio del sistema operativo Solaris. Para ello, creó un equipo de colaboración con GNOME, cuyos méritos más importantes giran en torno a la usabilidad y la accesibilidad de GNOME. En junio de 2003, SUN anunció que distribuiría GNOME 2.2 con la versión 9 de Solaris.

9.4.4. Estado actual de GNOME

GNOME se encuentra, a principios de 2007, en su versión 2.18. La mayoría de las tecnologías clave en las que se basa se hallan maduras, como se puede desprender de su número de versión. Así, el *broker* CORBA utilizado es ORBit2, mientras que el entorno gráfico y API, GTK+, acogió los cambios fruto de la experiencia acumulada en las versiones anteriores de GNOME. Como gran novedad aparece la inclusión de una biblioteca de accesibilidad, propuesta por SUN, que permite que las personas que tengan problemas de accesibilidad puedan utilizar el entorno GNOME. Mención especial también requiere Bonobo, el sistema de componentes de GNOME. Bonobo marcó una época dentro de GNOME a la par que se iba desarrollando el gestor de información personal Evolution. Sin embargo, el tiempo ha demostrado que las expectativas levantadas por Bonobo fueron demasiado altas y que la reutilización de esfuerzo mediante el uso de componentes no ha sido la que en un principio se esperaba.

Nota

La biblioteca ATK es una biblioteca de clases abstractas que permite hacer accesibles las aplicaciones. Esto quiere decir que las personas con alguna discapacidad (ciegos, daltónicos, gente con problemas de vista... que no pueden manejar el ratón, el teclado, etc.) pueden hacer uso de GNOME. El interés de SUN por la accesibilidad parte de que para poder ofrecer sus productos al Gobierno de los Estados Unidos ha de cumplir una serie de estándares en cuanto a accesibilidad. Tan en serio se lo ha tomado que en el equipo de desarrollo de GNOME que trabaja en SUN hay incluso un programador invidente. La arquitectura de accesibilidad de GNOME recibió en septiembre de 2002 el premio Hellen Keller Achievement Award.

9.4.5. Radiografía de GNOME

Los datos y las cifras que se muestran en la tabla 10 nos permitirán cerrar la presentación de GNOME. Las cifras corresponden al estado del CVS de GNOME el 14 de agosto de 2003. Ese día había más de nueve millones de líneas de código hospedadas en el repositorio CVS que tiene el proyecto GNOME. Aun cuando una comparación con KDE sería lo más natural, hemos de advertir al lector que las diferencias en cuanto a la organización de los proyectos la hacen desaconsejable si se quiere hacer en igualdad de condiciones. Esto se debe, por ejemplo, a que el CVS de GNOME incluye GIMP (un programa de creación

y manipulación de gráficos), responsable él solo de más de 660.000 líneas de código, o a la biblioteca GTK+ sobre la que se centra el desarrollo en GNOME, que cuenta a su vez con 330.000 líneas. Si a esto se le añade que el repositorio CVS de GNOME es más proclive a abrir nuevos módulos para programas (en total cuenta con setecientos) que el de KDE (que cuenta con menos de cien), podremos entender por qué GNOME cuenta con un número de líneas superior al de KDE a pesar de ser un año y medio más joven. El repositorio de GNOME acoge más de 225.000 ficheros, que han sido añadidos y modificados casi dos millones de veces (*vid.* el número de *commits* unas filas más abajo, en la tabla).

Nota

Una empresa cuyo deseo fuera crear un software del tamaño de GNOME debería contratar de media a unos doscientos cincuenta desarrolladores durante más de once años para conseguir un producto de tamaño similar, según el modelo COCOMO utilizado a lo largo de todo este capítulo. El coste asociado ascendería a unos 400 millones de dólares, una cifra pareja a la que una empresa de telefonía móvil ya asentada invertió en 2003 en reforzar su capacidad de red o similar a la que desembolsará una compañía de automóviles para abrir una planta de producción en Barcelona.

GNOME cuenta con unos recursos humanos de casi mil desarrolladores con acceso de lectura al sistema de control de versiones CVS, entre los que casi una veintena se dedican a GNOME de manera profesional (a tiempo completo o tiempo parcial). De ellos, sólo un 25% se ha mostrado activo en el último año, cifra que asciende al 40% si tenemos en cuenta los dos últimos años. El número medio de *commits* diarios registrados desde los inicios del proyecto casi llega al millar. Las herramientas de ayuda al desarrollo que utiliza el proyecto GNOME son básicamente las mismas que las que se usan en KDE, por lo que no se incidirá en ellas en este apartado.

Tabla 10. Análisis de GNOME

Página web	http://www.gnome.org
Inicio del proyecto	Septiembre 1997
Licencia	GNU GPL y GNU LGPL
Versión analizada	2.2
Líneas de código fuente	9.200.000
Número de ficheros (código, documentación, etc.)	228.000
Estimación de coste	$ 400.000.000
Estimación de tiempo de ejecución	11,08 años (133,02 meses)
Estimación de número medio de desarrolladores	250 aproximadamente
Número de subproyectos	Más de 700 módulos en el CVS
Número aproximado de desarrolladores	Casi 1.000 con acceso de escritura al CVS
Número de *commiters* activos en el último año	Alrededor de 500 (aproximadamente el 55% del total)
Número de *commiters* activos en los dos últimos años	Unos 700 (el 75% del total)

Número de *commits* en el CVS	1.900.000
Número medio de *commits* (totales) al día	Unos 900
Herramientas de ayuda al desarrollo	CVS, listas de correo, sitio web, sitio de noticias, reuniones anuales

Mientras que en KDE, C++ es indiscutiblemente el lenguaje más utilizado, en GNOME el puesto más alto corresponde a C. En GNOME, al igual que en KDE, este hecho se debe a que la biblioteca principal está escrita en C, por lo que el lenguaje *nativo* es ése, mientras que para programar con el resto de lenguajes se ha de esperar a que aparezcan los enlaces. El enlace más avanzado de GNOME es el que está incluido en gnome--, que no es otro que el de C++, por lo que no resulta sorprendente que el segundo lenguaje en la clasificación sea ése. Perl desde siempre ha tenido una amplia aceptación dentro de la comunidad GNOME y se ha puesto como ejemplo del hecho de que en GNOME se puede programar en multitud de lenguajes. Su puesta en práctica, sin embargo, no ha sido tan amplia como cabría esperar y supera ligeramente a Shell. Por otro lado, la aceptación de Python y de Lisp en GNOME ha sido bastante grande, como se puede desprender de su relativa importancia en esta clasificación, mientras que Java nunca ha llegado a despegar –probablemente debido a un enlace incompleto.

Tabla 11. Lenguajes de programación utilizados en GNOME

Lenguaje de programación	Líneas de código	Porcentaje
C	7.918.586	86,10%
C++	576.869	6,27%
Perl	199.448	2,17%
Shell	159.263	1,73%
Python	137.380	1,49%
Lisp	88.546	0,96%

9.4.6. Estudios académicos sobre GNOME

Los estudios más significativos de GNOME en el ámbito académico son los dos siguientes: "Results from software engineering research into open source development projects using public data" [158] y "The evolution of GNOME" [132].

- [158] es uno de los primeros grandes estudios de ingeniería del software en el campo del software libre. Los autores del mismo aprovecharon que los datos del desarrollo suelen estar públicamente accesibles para realizar mediciones de esfuerzo y compararlas con los modelos de estimación de costes, esfuerzos y tiempos clásicos. Uno de los modelos clásicos con los

que se compararon fue el que se ha utilizado en este capítulo, el modelo COCOMO.

- [132] hace un rápido repaso de los objetivos de GNOME y su breve historia, así como del uso que hace el proyecto GNOME de las tecnologías.

9.5. Apache

El servidor HTTP Apache es una de las aplicaciones estrella del mundo del software libre, ya que es el servidor web de mayor implantación según la encuesta que realiza en tiempo real (http://news.netcraft.com/archives/2003/08/01/august_2003_web_server_survey.html) [167]. Así, en mayo de 1999 el 57% de los servidores web corrían bajo Apache, mientras que en mayo de 2003 el porcentaje había aumentado hasta el 68%. Apache está disponible para todos los sabores de Unix (BSD, GNU/Linux, Solaris...), Microsoft Windows y otras plataformas minoritarias.

9.5.1. Historia de Apache

En marzo de 1989, Tim Berners Lee, un científico inglés que trabajaba en el CERN (Suiza) propuso una nueva forma para gestionar la ingente cantidad de información de los proyectos del CERN. Se trataba de una red de documentos hiperenlazados (hipertexto, tal como Ted Nelson lo había denominado ya en 1965); nacía el WWW. Hubo que esperar hasta noviembre de 1990 para que el primer software WWW viera la luz: en un paquete llamado WorldWideWeb se incluía un navegador web de interfaz gráfica y un editor WYSIWYG ("what you see is what you get", es decir, "lo que ve en la pantalla es lo que obtiene como resultado"). Dos años después, la lista de servidores WWW contaba con una treintena de entradas, entre las cuales ya se encontraba el NCSA HTTPd.

La verdadera historia de Apache comienza cuando en marzo de 1995, Rob McCool abandona el NCSA. Apache 0.2 vería la luz el 18 de marzo de 1995 basado en el servidor NCSA HTTPd 1.3, realizado por el propio Rob McCool durante su estancia en NCSA. Durante esos primeros meses, Apache era una colección de parches aplicados al servidor NCSA, hasta que Robert Thau lanzó Shambhala 0.1, una reimplementación casi completa que ya incluía la API para los módulos que ha resultado ser tan exitosa.

Nota

El nombre del proyecto Apache se debe a su filosofía de desarrollo y de organización. Al igual que la tribu de los apaches, los desarrolladores de Apache decidieron que su forma organizativa debía fundamentarse en los méritos personales de los desarrolladores para con el resto de la comunidad Apache. Se ha extendido, sin embargo, la leyenda de que el nombre Apache en realidad se debe a que en los primeros tiempos no dejaba de ser un servidor NCSA parcheado, en inglés *a patchy server*.

Habría que esperar a enero de 1996 para poder disfrutar de la primera versión estable de Apache, la Apache 1.0, que incluía la carga de módulos en tiempo de ejecución a modo de pruebas además de otras funcionalidades interesantes. Los primeros meses de ese año fueron especialmente fructíferos para el proyecto, ya que la versión 1.1, que contaba con módulos de autenticación contra bases de datos (como MySQL), se publicó apenas dos meses después. Desde entonces hasta la actualidad, los hitos más grandes del proyecto han sido la total conformidad con el estándar HTTP 1.1 (incluido en abril de 1997 en Apache 1.2), la inclusión de la plataforma Windows NT (que comenzó ya en julio de 1997 con las versiones en pruebas de Apache 1.3), la unificación de los archivos de configuración en uno solo (para lo cual habría que esperar ya a octubre de 1998, a Apache 1.3.3) y el lanzamiento, todavía en pruebas, de la siguiente generación de Apache, Apache 2.

Entre tanto, en junio de 1998, IBM decidió que el motor de su producto WebSphere fuera Apache, en lugar de desarrollar un servidor HTTP propio. Esto se vio como un gran espaldarazo por parte del gigante azul al proyecto Apache y al software libre en general, aunque para facilitar este hecho hubiera que cambiar ligeramente la licencia Apache original.

9.5.2. Desarrollo de Apache

El servidor HTTP Apache es el proyecto central dentro de los muchos que gestiona la Apache Software Foundation. El diseño modular de Apache ha permitido que existan una serie de proyectos satélite –algunos incluso más grandes en tamaño que el propio Apache– en torno a Apache. De esta forma, el servidor HTTP Apache contiene el núcleo del sistema con las funcionalidades básicas, mientras que las funcionalidades adicionales las aportan los diferentes módulos. Los módulos más conocidos son *mod_perl* (un intérprete del lenguaje de guión Perl empotrado en el servidor web) y Jakarta (un potente servidor de aplicaciones). En los siguientes párrafos se va a describir solamente el proceso de desarrollo seguido para el servidor HTTP, sin tener en cuenta los demás módulos, que pueden tener modelos parecidos o no.

El desarrollo del servidor HTTP Apache se fundamenta en el trabajo de un reducido grupo de desarrolladores denominado Apache Group. El Apache Group lo constituyen aquellos desarrolladores que han colaborado durante un período de tiempo prolongado, generalmente más de seis meses. El desarrollador, después de ser nominado por un miembro del Apache Group para formar parte del mismo, es votado entre todos los miembros del Apache Group. En sus comienzos, el Apache Group constaba de ocho desarrolladores, luego de doce, y en la actualidad cuenta con veinticinco personas.

Sobre el Apache Group recae la responsabilidad de la evolución del servidor web y, por tanto, de las decisiones puntuales de desarrollo en cada momento. Hay que diferenciar el Apache Group del núcleo de desarrolladores (*core group*) activo en cada momento. El carácter voluntario de la mayoría de los

desarrolladores hace que sea improbable que todos los que componen el Apache Group puedan estar activos todo el tiempo, por lo que el *core* se define como aquéllos que en un espacio de tiempo pueden ocuparse de las tareas en Apache. En líneas generales, las decisiones que han de tomar los desarrolladores pertenecientes al núcleo se limitan a la votación de la inclusión de código –aunque esto se reserve en realidad sólo para grandes cambios– y a cuestiones de diseño. Por otra parte, en general suelen tener derecho de escritura en el repositorio CVS, por lo que sirven como puerta de entrada del código asegurando que sea correcto y su calidad.

9.5.3. Radiografía de Apache

Las cifras que se exponen a continuación corresponden a la versión del servidor HTTP Apache tal como se podía descargar del servidor CVS del proyecto Apache el 18 de abril de 2003. No se han tenido en cuenta ninguno de los numerosos módulos con los que cuenta el proyecto Apache. Como se puede observar, Apache es un proyecto relativamente pequeño en comparación con los demás casos de estudio considerados en este capítulo. Aunque ya se ha comentado con anterioridad, es importante hacer hincapié en la modularidad de Apache, que permite precisamente esto: que el núcleo sea pequeño y manejable. El repositorio CVS del proyecto Apache, que contiene el núcleo del servidor web y muchos módulos adicionales, alberga en total más de cuatro millones de líneas de código fuente, una cifra ligeramente inferior a proyectos como KDE y GNOME.

La versión 1.3 de Apache contaba con poco más de 85.000 líneas de código fuente, una cifra que según el modelo COCOMO requeriría un esfuerzo de desarrollo de veinte desarrolladores a tiempo completo de media durante un año y medio. El coste total del proyecto rondaría entonces los 4 millones de dólares. En la elaboración del servidor web de Apache se cuentan hasta sesenta *commiters* diferentes, mientras que el número de desarrolladores que han aportado se calcula que es de unos cuatrocientos.

Tabla 12. Análisis de Apache

Página web	http://www.apache.org
Inicio del proyecto	1995
Licencia	Apache Free Software License
Versión analizada	2.2.4
Líneas de código fuente	225.065
Número de ficheros	2.807
Estimación de coste	$ 7.971.958
Estimación de tiempo de ejecución	2,52 años (30,27 meses)

Estimación de número medio de desarrolladores	23,4
Número aproximado de desarrolladores	60 *commiters* (400 desarrolladores)
Herramientas de ayuda al desarrollo	CVS, listas de correo, sistema de notificación de errores

Apache 1.3 está escrito casi íntegramente en lenguaje C, y la presencia de otros lenguajes de programación es escasa, sobre todo si tenemos en cuenta que la gran mayoría de las líneas escritas en el segundo lenguaje, Shell, corresponden a ficheros de configuración y de ayuda a la compilación.

Tabla 13. Lenguajes de programación utilizados en Apache

Lenguaje de programación	Líneas de código	Porcentaje
C	208.866	92,8%
Shell	12.796	5,69%
Perl	1.649	0,73%
Awk	874	0,39%

9.6. Mozilla

El proyecto Mozilla trabaja un conjunto de aplicaciones integrado para Internet, libres y multiplataforma, cuyos productos más destacados son el navegador web Mozilla Firefox y el cliente de correo Mozilla Thunderbird. Este conjunto está pensado también como plataforma de desarrollo de otras aplicaciones, de manera que son muchos los navegadores que utilizan Gecko, el motor de HTML de Mozilla (como Galeon).

El proyecto está gestionado por la Fundación Mozilla, una organización sin fines de lucro dedicada a la creación de software libre, con la misión específica de "mantener la elección y la innovación en Internet". Por ello los productos de Mozilla tienen en cuentra tres principios básicos: ser software libre, respetar los estándares y ser portables a diferentes plataformas.

9.6.1. Historia de Mozilla

La historia de Mozilla es larga y enrevesada, pero al mismo tiempo muy interesante, ya que permite seguir la historia del propio WWW. Y es que si trazamos las personas e instituciones que han estado involucradas en el desarrollo de Mozilla, llegaremos al punto de partida de la web, con el lanzamiento del primer navegador web completo.

Al igual que con el antecesor de Apache, fue en el NCSA donde también "nació" el primer navegador web completo en 1993, Mosaic. Muchos de los miembros del equipo de desarrollo, con Marc Andreessen y Jim Clark a la cabeza, crearon una pequeña empresa para escribir, empezando desde cero (ya que había problemas con los derechos de autor del código de Mosaic y el diseño técnico del programa tenía sus limitaciones –*vid. Speeding the Net: the inside story of Netscape and how it challenged Microsoft* [189]–), lo que más tarde sería el navegador Netscape Communicator, que fue líder indiscutible del mercado de los navegadores web hasta la llegada de Microsoft Internet Explorer. Además de la innovación puramente tecnológica que supuso el navegador Netscape, Netscape Inc. también fue innovadora en la forma de conseguir copar el mercado. Contra todo el sentido común de la época, su aplicación estrella, el navegador WWW, se podía obtener (y distribuir con ciertas limitaciones) de manera gratuita. Este hecho, hasta entonces insólito dentro del mundo empresarial, causó cierta sorpresa, pero demostró a la larga ser un acierto para la estrategia de Netscape Inc., que sólo un gigante como Microsoft supo atajar con una táctica más agresiva (y probablemente lesiva con la libre competencia).

Hacia 1997, la cota de mercado de Netscape había caído en picado debido a la implantación de Microsoft Explorer, por lo que desde Netscape Inc. se estudiaban nuevas fórmulas para volver a ganarlo. Un informe técnico del ingeniero Frank Hecker ("Setting up shop: the business of open-source software", 1998) [142] propuso que la mejor solución al problema era liberar el código fuente del navegador y beneficiarse de los efectos de la comunidad del software libre tal como Eric Raymond describía en "La catedral y el bazar". En enero de 1998 Netscape Inc. anunció oficialmente que liberaría el código fuente de su navegador, marcando un hito de gran importancia dentro de la corta historia del mundo del software libre: una empresa iba a publicar todo el código fuente de una aplicación hasta entonces comercial bajo una licencia de software libre. La fecha indicada para el lanzamiento era el 31 de marzo de 1998.

En los dos meses que van de enero a marzo, la actividad en Netscape para que todo estuviera a punto fue frenética. La lista de tareas era enorme y compleja ("Freeing the source: the story of Mozilla", 1999) [134]. En el plano técnico, había que contactar con las empresas que habían realizado módulos para pedir su consentimiento en el cambio de licencia; en caso de negativa, el módulo debía ser eliminado. Además, todas las partes escritas en Java debían ser reimplementadas, ya que se consideró que Java no era libre. Se decidió llamar al proyecto libre Mozilla, tal como los propios desarrolladores de Netscape llamaban al componente principal de Netscape, y se adquirió el dominio Mozilla.org para construir una comunidad de desarrolladores y colaboradores en torno a ese sitio web. Al final del proceso, se liberaron más de un millón y medio de líneas de código fuente.

Nota

El nombre de Mozilla es un juego de palabras con un toque humorístico del equipo de desarrollo en Netscape Inc. Mozilla es el producto de la adaptación de Godzilla, un monstruo que causaba pánico en las películas de terror japonesas desde la década de los cincuenta, para que sonara como *Mosaic Killer*, ya que se pretendía que Mosaic quedara obsoleto gracias a este nuevo navegador, que presentaba tecnologías mucho más avanzadas.

Por otro lado estaba el plano legal. Las licencias libres existentes en aquel momento no convencían a los ejecutivos de Netscape, que veían que no "congeniaban" con el carácter comercial de una compañía. Netscape quería una licencia más *flexible* que permitiera llegar a acuerdos con terceros para incluir su código indiferentemente de su licencia o que otros desarrolladores comerciales contribuyeran pudiendo defender sus intereses económicos como desearan. Y aunque en un principio no se había previsto crear una nueva licencia, se llegó a la conclusión de que era la única forma de conseguir lo que deseaban. Así es como se fraguó la Netscape Public License (NPL), una licencia que se basaba en los principios básicos de las licencias de software libre, pero que concedía unos derechos adicionales a Netscape Inc. –lo que la hacía también una licencia *no libre* bajo el prisma de la Free Software Foundation. Cuando se publicó el borrador de la NPL para su discusión pública, llovieron las críticas sobre la cláusula de derechos adicionales para Netscape. Netscape Inc. reaccionó rápidamente ante estos comentarios y creó una licencia adicional, la Mozilla Public License (MPL), que era idéntica a la NPL pero sin que Netscape tuviera ningún derecho adicional.

La decisión final fue liberar el código de Netscape bajo la licencia NPL, que otorgaba derechos adicionales a Netscape, y que el código nuevo que fuera incluido estuviera bajo la MPL (o compatible). Las correcciones al código original (licenciado con la NPL) debían estar también bajo esa licencia.

Nota

En la actualidad, Mozilla acepta contribuciones bajo la licencia propia MPL, la GPL y la LGPL. El cambio de licencia no fue nada fácil, ya que hubo que buscar a todos los que habían contribuido con código alguna vez para que dieran su visto bueno al cambio de NPL/MPL a MPL/GPL/LGPL. Con el objeto de poder relicenciar todo el código, se creó una página web que contenía un listado de trescientos desarrolladores "perdidos" ("Have you seen these hackers?") [38]. En mayo de 2007, aún se sigue buscando a dos de estos desarrolladores.

El desarrollo con el código original de Netscape Communicator fue, sin lugar a dudas, más complicado de lo que inicialmente se esperaba. Las condiciones de partida ya eran malas de por sí, porque lo que fue liberado en ocasiones estaba incompleto (se habían quitado todos los módulos de terceros que no habían dado su visto bueno a la liberación) y funcionaba a duras penas. Por si fuera poco, al problema técnico de pretender que Mozilla funcionara sobre una gran cantidad de sistemas operativos y plataformas, había que añadir los vicios adquiridos de Netscape Inc., con ciclos de liberación largos e ineficientes para el mundo de Internet y que no distinguía entre sus intereses y los de una comunidad en torno a Mozilla. Todo esto llevó a que justo un año después,

uno de los programadores más activos antes y después de la liberación, Jamie Zawinsky, decidiera tirar la toalla en una amarga carta ("Resignation and post-mortem", 1999) [237] en la que mostraba su desesperación y su desolación.

El 15 de julio de 2003 Netscape Inc. (propiedad de America On Line) anunció que dejaba de desarrollar el navegador Netscape y, por tanto, la tutela activa del proyecto Mozilla. A modo de "finiquito" aprobó la creación de la Fundación Mozilla, a la que apoyó con una aportación de dos millones de dólares. Asimismo, todo el código que se encontraba bajo la NPL (la licencia pública de Netscape) se donó a la Fundación y se redistribuyó con las licencias promulgadas ya anteriormente por el proyecto Mozilla: MPL, LGPL y GPL.

El 10 de marzo de 2005 la Fundación Mozilla anunció que no se publicarían más versiones oficiales de Mozilla Application Suite, que era sustituda por Mozilla SeaMonkey, que incluye navegador web, cliente de correo electrónico, libreta de contactos, editor de páginas y cliente de IRC. Por otro lado, el proyecto Mozilla aloja varias aplicaciones independientes, entre las que se pueden destacar Mozilla Firefox (navegador web), sin duda la más conocida, Mozilla Thunderbird (cliente de correo y lector de noticias), Mozilla Sunbird (calendario), Mozilla Nvu (editor web), Camino (navegador para Mac OS X) y Bugzilla (herramienta basada en web para el seguimiento de informes de error).

Con el tiempo, y a pesar de muchas dudas y de largos períodos en los que a muchos les parecía que estaba abocado al fracaso, el proyecto parece gozar de buena salud. La versatilidad y la portabilidad de sus aplicaciones ha hecho que, aun estando en muchos casos muy necesitadas de recursos de ejecución, sean consideradas (en general, pero muy especialmente Firefox) como la pareja de OpenOffice.org en el escritorio del usuario final.

9.6.2. Radiografía de Mozilla

Las medidas que se van a presentar en este apartado corresponden al estudio de Firefox, la aplicación más conocida del proyecto. Según las estimaciones del modelo COCOMO, una compañía que quisiera crear un software de tales dimensiones tendría que invertir unos 111 millones de dólares para obtenerlo. El tiempo que habría de esperar se situaría en torno a los siete años y el número medio de programadores a tiempo completo que debería emplear rondaría los ciento veinte.

Tabla 14. Estado actual de Mozilla Firefox

Página web	www.mozilla-europe.org/es/products/firefox/
Inicio del proyecto	2002
Licencia	MPL/LGPL/GPL

Versión	2.0
Líneas de código fuente	2.768.223
Estimación de coste	$ 111.161.078
Estimación de tiempo de ejecución	6,87 años (82,39 meses)
Estimación de número medio de desarrolladores	120
Número aproximado de desarrolladores	50 *commiters*
Herramientas de ayuda al desarrollo	CVS, listas de correo, IRC, Bugzilla...

En cuanto a los lenguajes de programación, C++ y C son, por este orden, los más utilizados. La presencia de Perl se debe en gran parte a que las herramientas de ayuda al desarrollo llevadas a cabo por el proyecto Mozilla, como BugZilla o Tinderbox, están realizadas en este lenguaje. Lo que sí que resulta algo sorprendente es el alto número de líneas de código en lenguaje ensamblador en una aplicación de usuario final. La inspección del código en el repositorio ha mostrado que, efectivamente, existen bastantes ficheros codificados en lenguaje ensamblador.

Tabla 15. Lenguajes de programación utilizados en Mozilla Firefox

Lenguaje de programación	Líneas de código	Porcentaje
C++	1.777.764	64,22%
C	896.551	32,39%
Ensamblador	34.831	1,26%
Perl	26.768	0,97%
Shell	16.278	0,59%
C#	6.232	0,23%
Java	5.352	0,19%
Python	3.077	0,11%
Pascal	459	0,02%

9.7. OpenOffice.org

OpenOffice.org es una de las aplicaciones estrella del panorama actual del software libre. Se trata de un paquete ofimático (*suite*) multiplataforma que incluye las aplicaciones clave en un entorno de escritorio de oficina, tales como procesador de texto (Writer), hoja de cálculo (Calc), gestor de presentaciones (Impress), programa de dibujo (Draw), editor de fórmulas matemáticas (Math) y finalmente, editor de lenguaje HTML (incluido en Writer). La interfaz que

ofrece OpenOffice.org es homogénea e intuitiva, similar en aspecto y funcionalidades a otros paquetes ofimáticos, en especial al más arraigado en la actualidad, Microsoft Office.

Escrito en C++, OpenOffice.org incluye la API de Java y tiene su propio sistema de componentes empotrables, que permite incluir, por ejemplo, tablas de la hoja de cálculo en el procesador de textos de una manera sencilla e intuitiva. Entre sus ventajas, podemos decir que maneja una gran cantidad de formatos de archivo, incluidos los de Microsoft Office. Sus formatos de archivo nativos, a diferencia de los del paquete ofimático de Microsoft, están basados en XML, por lo que se muestra como una clara apuesta por la versatilidad, la facilidad de transformación y la transparencia. En la actualidad, OpenOffice.org está traducido a más de veinticinco lenguas y se ejecuta en Solaris (su sistema nativo), GNU/Linux y Windows. En un futuro no muy lejano se esperan versiones para FreeBSD, IRIX y Mac OS X.

OpenOffice.org adoptó su nombre definitivo (OpenOffice –tal como lo conoce todo el mundo– más la coletilla *.org*) después de un litigio en el que fue demandado por usurpación de nombre de marca por otra empresa.

9.7.1. Historia de OpenOffice.org

A mediados de la década de los ochenta, se fundó en la República Federal de Alemania la empresa StarDivision, que tuvo como objetivo principal la creación de un paquete ofimático: StarOffice. En verano de 1999, SUN Microsystems –con la clara intención de arrebatarle un mercado conquistado ya por aquel entonces a Microsoft– decidió adquirir la empresa StarDivision y hacer una fuerte apuesta por StarOffice. Así, en junio de 2000 lanzó la versión 5.2 de StarOffice, que podía ser descargada gratuitamente en la Red.

Sin embargo, el éxito de StarOffice fue limitado, ya que el mercado estaba fuertemente dominado por el paquete ofimático de Microsoft. SUN decidió cambiar su estrategia y, al igual que Netscape con el proyecto Mozilla, decidió aprovechar las ventajas del software libre para ganar en importancia e implantación. De esta forma, las futuras versiones de StarOffice (producto propietario de SUN) se crearían utilizando OpenOffice.org (producto libre) como fuente, respetando las interfaces de programación (API) y los formatos de fichero, y sirviendo como implementación de referencia.

9.7.2. Organización de OpenOffice.org

OpenOffice.org pretende tener una estructura decisoria en la que todos los miembros de la comunidad se sientan partícipes. Por eso, se ha ideado un sistema para que la toma de decisiones tenga el mayor consenso posible. El proyecto OpenOffice.org se divide en una serie de subproyectos que cuentan con unos miembros del proyecto, los colaboradores, y un único líder. Por supues-

to, los miembros de un proyecto pueden participar en más de un proyecto, al igual que el líder. Sin embargo, no se puede liderar más de un proyecto a la vez. Los proyectos se dividen en tres categorías:

- Proyectos aceptados. Pueden ser tanto de carácter técnico como no técnicos. Los líderes de cada proyecto aceptado cuentan con un voto a la hora de la toma de decisiones globales.

- Proyectos *native-lang*. Son todos los proyectos de internacionalización y localización de OpenOffice.org. En la actualidad, como se ha comentado con anterioridad, existen más de veinticinco equipos que trabajan para traducir las aplicaciones de OpenOffice.org a las diferentes lenguas y convenciones. En conjunto, *native-lang* cuenta con un único voto para la toma de decisiones globales.

- Proyectos en la incubadora. Se trata de proyectos patrocinados por la comunidad (generalmente experimentales o pequeños). Pueden pasar a ser aceptados tras un período de seis meses. De esta forma, la comunidad OpenOffice.org puede garantizar que los proyectos aceptados están basados en un interés real, ya que la *mortalidad* de proyectos nuevos en el mundo del software libre es muy grande. En total, los proyectos en la incubadora cuentan con un voto en la toma de decisiones.

9.7.3. Radiografía de OpenOffice.org

El paquete ofimático OpenOffice.org está compuesto por cerca de cuatro millones de líneas de código fuente distribuidas a lo largo de cuarenta y cinco mil ficheros.

El modelo COCOMO estima el esfuerzo necesario para realizar un "clon" de OpenOffice.org en ciento ochenta programadores que trabajen durante casi ocho años a tiempo completo. El coste de desarrollo ascendería, según las estimaciones de COCOMO, a unos 215 millones de dólares.

Los resultados que se comentan en este apartado fueron obtenidos del estudio del código fuente de la versión estable 2.1 de OpenOffice.org.

Tabla 16. Estado actual de OpenOffice.org

Página web	http://www.openoffice.org
Inicio del proyecto	Junio de 2000 (primera versión libre)
Licencia	LGPL y SISSL
Versión	2.1
Líneas de código fuente	5.197.090

Estimación de coste	$ 215.372.314
Estimación de tiempo de ejecución	8,83 años (105,93 meses)
Estimación de número medio de desarrolladores	180
Número aproximado de desarrolladores	200 *commiters*
Herramientas de ayuda al desarrollo	CVS, listas de correo

En cuanto a los lenguajes de programación utilizados en OpenOffice.org, el primer lugar lo ocupa C++. Es interesante observar cómo la adquisición por parte de SUN implicó la integración de mucho código Java en el paquete ofimático, que superó incluso a C.

Tabla 17. Lenguajes de programación utilizados en OpenOffice.org

Lenguaje de programación	Líneas de código	Porcentaje
C++	4.615.623	88,81%
Java	385.075	7,41%
C	105.691	2,03%
Perl	54.063	1,04%
Shell	12.732	0,24%
Yacc	6.828	0,13%
C#	6.594	0,13%

9.8. Red Hat Linux

Red Hat Linux fue una de las primeras distribuciones comerciales de GNU/Linux. Hoy en día es probablemente una de las más conocidas, y seguramente la que se puede considerar como la "canónica" de entre las distribuciones comerciales. El trabajo de los distribuidores está básicamente relacionado con tareas de integración, y no tanto con el desarrollo de software. Por supuesto, tanto Red Hat como otras distribuciones pueden tener desarrolladores entre sus empleados, pero su labor es secundaria para los objetivos de una distribución. En general, se asume que el trabajo que realizan las distribuciones es simplemente tomar los paquetes fuente (generalmente los archivos que publican los propios desarrolladores) y empaquetarlos de forma que cumplan ciertos criterios (tanto técnicos como organizativos). El producto de este proceso es una distribución: una serie de paquetes convenientemente organizados que posibilitan al usuario su instalación, su desinstalación y su actualización.

Las distribuciones también son responsables de la calidad del producto final, un aspecto muy importante si tenemos en cuenta que muchas de las aplicaciones que incluyen han sido elaboradas por voluntarios en su tiempo libre. Aspectos de seguridad y estabilidad son, por tanto, de capital importancia para una distribución.

9.8.1. Historia de Red Hat

Red Hat Software Inc. fue fundada en 1994 por Bob Young y Marc Ewing. Su principal objetivo era compilar y comercializar una distribución GNU/Linux que se llamó (y todavía se sigue llamando) Red Hat Linux [236]. Básicamente, se trataba de una versión empaquetada de lo que existía en la Red en aquellos tiempos, que incluía documentación y soporte. Durante el verano de 1995, la versión 1.0 de esta distribución vio la luz. Unos meses más tarde, en otoño, se publicó la versión 2.0, que incluía la tecnología RPM (*RPM package manager*, 'gestor de paquetes RPM'). El sistema de paquetes RPM se ha convertido en un estándar *de facto* para los paquetes de sistemas GNU/Linux. En 1998, Red Hat llegó al gran público con la versión 5.2. Para una historia completa de los nombres de las diferentes versiones de Red Hat, se puede consultar "The truth behind Red Hat names" [201].

Nota

Desde la versión 1.1 de la Linux Standard Base (una especificación cuya meta es conseguir compatibilidad binaria entre las distribuciones GNU/Linux de la que se encarga el Free Standards Group), RPM ha sido elegido como el sistema de paquetes estándar. El proyecto Debian continúa con su formato de paquete propio, así como muchas de las distribuciones dependientes del sistema de paquetes Debian, y se ajustan al formato estandarizado mediante una herramienta de conversión que se llama *alien*.

Antes de la existencia del sistema de gestión de paquetes RPM, casi todas las distribuciones de GNU/Linux ofrecían la posibilidad de instalar el software mediante un procedimiento dirigido por menús, pero hacer modificaciones a una instalación ya hecha, especialmente agregar paquetes de software nuevos tras la instalación, no era una tarea fácil. RPM permitió dar ese paso más allá del estado del arte al proveer a los usuarios de la posibilidad de gestionar sus paquetes ("Maximum RPM. Taking the Red Hat package manager to the limit", 1998) [83], lo que permitiría borrar, instalar o actualizar cualquier paquete software existente en la distribución de manera mucho más sencilla. El sistema de paquetes RPM sigue siendo el sistema de paquetes más utilizado entre las diferentes distribuciones de GNU/Linux. Las estadísticas de Linux Distributions, "Facts and figures", 2003 [92], un sitio web que contiene información cualitativa y cuantitativa acerca de un gran número de distribuciones, muestran que en mayo de 2003 una gran mayoría de las ciento dieciocho distribuciones computadas utilizan el gestor RPM, en total sesenta y cinco (lo que viene a ser un 55% del total). En comparación, el sistema de paquetes de Debian (conocido como *deb*) lo utilizan únicamente dieciséis distribuciones (un 14% del total).

Sin embargo, Red Hat Inc. no sólo es conocida por su distribución de software basada en Linux. En agosto de 1999, Red Hat salió a bolsa y sus acciones obtuvieron la octava ganancia de primer día más grande en toda la historia de Wall Street. Cuatro años más tarde, el valor de las acciones de Red Hat es alrededor de un centésima parte del máximo valor que llegaran a alcanzar antes de la crisis de las puntocom. Aun así, sus comienzos exitosos en el mercado de valores sirvieron para que Red Hat fuera portada en periódicos y revistas no directamente relacionados con temas informáticos. En cualquier caso, parece ser que Red Hat ha sabido superar los problemas de otras compañías del mundo de los negocios en torno al software libre y anunció números negros por primera vez en su historia en el último cuarto del año 2002.

Otro de los hechos históricos más importantes de Red Hat fue la adquisición en noviembre de 1999 de Cygnus Solutions, una empresa fundada una década antes y que ya había demostrado cómo con una estrategia integral basada en software libre se puede ganar dinero ("Future of Cygnus Solutions. An entrepreneur's account") [216]. Cygnus escogió el exigente mercado de los compiladores para hacerse un hueco. Su estrategia comercial se basaba en el desarrollo y la adaptación de las herramientas de desarrollo de software GNU (básicamente GCC y GDB) a petición del cliente.

En septiembre de 2003, Red Hat decidió concentrar sus esfuerzos de desarrollo en la versión corporativa de su distribución y delegó la versión común a Fedora Core, un proyecto abierto independiente de Red Hat.

En junio de 2006 Red Hat adquirió la compañía JBoss Inc., convirtiéndose en la responsable del desarrollo del servidor más importante de aplicaciones J2EE de código abierto.

9.8.2. Estado actual de Red Hat

En la actualidad, los productos estrella de Red Hat Inc. son Fedora Core y Red Hat Network, un servicio de actualización de software por medio de la Red. Este tipo de servicios están más bien orientados al usuario final y no tanto al entorno empresarial, pero sirven a Red Hat como buen reclamo y para asegurar su estrategia de marca.

La "verdadera" estrategia comercial de Red Hat se encuentra en sus productos dirigidos al mundo empresarial. Este tipo de productos son mucho menos conocidos, pero suponen una gran parte de la facturación de Red Hat, muy superior a la que percibe por sus productos estrella más populares en el sentido literal.

Red Hat cuenta con una distribución orientada a la empresa, integrada en torno a un servidor de aplicaciones y llamada Red Hat Entreprise Linux AS. Con la adquisición de este software, el cliente tiene derecho a soporte. El servicio análogo a Red Hat Network para usuarios comerciales es Red Hat Enterprise

Network, que incluye la gestión del sistema y la posibilidad de actualizaciones. Por otro lado, Red Hat ofrece también servicios de consultoría informática y un programa de certificación similar al que existe en el mundo Windows ofrecido por Microsoft.

9.8.3. Radiografía de Red Hat

Red Hat ha superado recientemente la barrera de los cincuenta millones de líneas de código, que hacen de ella una de las mayores distribuciones de software que han llegado a existir –superando, como se verá más adelante en este capítulo, el tamaño de sistemas operativos propietarios. La versión 8.1 de Red Hat estaba constituida por 792 paquetes, por lo que podemos asumir que también habrá franqueado el listón de los ochocientos paquetes en su última versión, si tenemos en cuenta que su número suele incrementarse levemente de versión en versión.

Al igual que en los casos anteriores, se ha aplicado el modelo COCOMO para estimar la inversión y el esfuerzo que sería necesario invertir en la generación de un software de idéntico tamaño. Sin embargo, para el caso de Red Hat se ha considerado que se trata de un producto realizado a partir de una serie de aplicaciones independientes. Por eso, se ha realizado una estimación mediante COCOMO independiente para cada uno de los paquetes de Red Hat, para luego sumar el coste y el personal total necesario. En el caso del tiempo de ejecución óptimo para Red Hat, se ha considerado el del paquete más grande, ya que idealmente, como todos los paquetes son independientes, podrían realizarse de manera paralela en el tiempo. Por esto el tiempo de ejecución óptimo para Red Hat es parecido al de los proyectos que se han presentado con anterioridad en este capítulo.

Según COCOMO, se necesitarían alrededor de siete años y medio y un equipo de programadores compuesto de media por mil ochocientos desarrolladores para realizar la distribución Red Hat Linux 8.1 desde cero. El coste de desarrollo final ascendería a unos 1.800 millones de dólares.

Nota

Mil ochocientos millones de dólares es el presupuesto que el Ministerio de Defensa español destinará a renovar su flota de helicópteros. De todo el montante, la mitad irá destinado a la adquisición de veinticuatro helicópteros, por lo que el precio de Red Hat sería el equivalente a cuarenta y ocho helicópteros de combate. Asimismo, la cifra de 1.800 millones de dólares es la que obtuvo de beneficios a nivel mundial la película *Titanic*.

Tabla 18. Estado de Red Hat Linux

Página web	http://www.redhat.com
Inicio del proyecto	1993
Licencia	

Versión	9.0
Líneas de código fuente	Más de 50.000.000
Número de paquetes	792
Estimación de coste	$ 1.800.000.000
Estimación de tiempo de ejecución	7,35 años (88,25 meses)
Estimación de número medio de desarrolladores	1.800
Número aproximado de desarrolladores	Empleados de Red Hat (generalmente sólo integración)
Herramientas de ayuda al desarrollo	CVS, listas de correo

Debido a la presencia de un amplio número de paquetes, la clasificación de lenguajes en Red Hat tiene mayor diversidad que las que hemos visto en las aplicaciones más importantes de software libre. En términos generales, se percibe la gran importancia de C, con más de un sesenta por ciento de las líneas de código. En segundo lugar, con más de diez millones de líneas de código, se encuentra C++, seguido de lejos por Shell. Es interesante observar que a Perl se unen después Lisp (mayoritariamente debido a su uso en Emacs), el código en lenguaje ensamblador (del cual una cuarta parte corresponde al que viene con Linux) y un lenguaje en franco retroceso en cuanto a su uso, Fortran.

Tabla 19. Lenguajes de programación utilizados en Red Hat

Lenguaje de programación	Líneas de código	Porcentaje
C	30.993.778	62,13%
C++	10.216.270	20,48%
Shell	3.251.493	6,52%
Perl	1.106.082	2,22%
Lisp	958.037	1,92%
Ensamblador	641.350	1,29%
Fortran	532.629	1,07%

9.9. Debian GNU/Linux

Debian es un sistema operativo libre que en la actualidad utiliza el núcleo de Linux para llevar a cabo su distribución (aunque se espera que haya distribuciones Debian basadas en otros núcleos, como es el caso de "the HURD", en el futuro). Actualmente, está disponible para varias arquitecturas diferentes, que incluyen Intel x86, ARM, Motorola, 680x0, PowerPC, Alpha y SPARC.

Debian no es sólo la mayor distribución GNU/Linux de la actualidad, sino también una de las más estables, y disfruta de varios premios relacionados con la preferencia de los usuarios. Aunque su base de usuarios sea difícil de estimar, ya que el proyecto Debian no vende CD u otros medios con su software y el software que contiene puede ser redistribuido por cualquiera que así lo desee, podemos suponer, sin faltar mucho a la verdad, que se trata de una distribución importante dentro del mercado de GNU/Linux.

En Debian existe una categorización según la licencia y los requisitos de distribución de los paquetes. El núcleo de la distribución Debian (la apartado llamada *main*, que aglutina una gran variedad de paquetes) está compuesto sólo por software libre de acuerdo con las directrices de Debian (DFSG, Debian Free Software Guidelines) [104]. Está disponible en Internet para ser descargado y muchos redistribuidores lo venden en CD u otros medios.

Las distribuciones de Debian son creadas por cerca de un millar de voluntarios (generalmente profesionales de la informática). La labor de estos voluntarios radica en tomar los programas fuente –en la mayoría de los casos de sus autores originales–, configurarlos, compilarlos y empaquetarlos de manera que un usuario típico de una distribución Debian sólo tenga que seleccionar el paquete para que el sistema lo añada sin mayores problemas. Esto que a simple vista puede parecer simple, se torna complejo en cuanto se introducen factores como las dependencias entre los diferentes paquetes (el paquete A necesita, para poder funcionar, el paquete B) y las diferentes versiones de todos estos paquetes.

La labor de los integrantes del proyecto Debian es la misma que la que se lleva a cabo en cualquier otra distribución: la integración de software para su correcto funcionamiento conjunto. Además del trabajo de adaptación y de empaquetamiento, los desarrolladores de Debian se encargan de mantener una infraestructura de servicios basados en Internet (sitio web, archivos en línea, sistema de gestión de errores, listas de correo de ayuda, soporte y desarrollo, etc.), varios proyectos de traducción e internacionalización, el desarrollo de varias herramientas específicas de Debian y, en general, cualquier elemento que haga posible la distribución Debian.

Aparte de su naturaleza voluntaria, el proyecto Debian tiene una característica que lo hace especialmente singular: el contrato social de Debian (http://www.debian.org/social_contract.html) [106]. Este documento contiene no sólo los objetivos principales del proyecto Debian, sino también los medios que se utilizarán para llevarlos a cabo.

Debian también es conocida por tener una política de paquetes y versiones muy estricta con el fin de conseguir una mayor calidad del producto ("Debian policy manual") [105]. Así, en todo momento existen tres *sabores* diferentes de Debian: una versión estable, una versión inestable y otra en pruebas. Como su propio nombre indica, la versión estable es la recomendada para siste-

mas y personas no aptas a sobresaltos. Su software ha de pasar un período de congelación en el que sólo se corrigen erratas. La norma es que en la versión estable de Debian no ha de haber ningún error crítico conocido. En cambio, esta versión estable no suele tener las últimas versiones del software (lo más novedoso).

Para los que deseen tener una versión con el software más actual existen otras dos versiones de Debian coetáneas a la estable. La versión inestable incluye paquetes en vías de estabilización, mientras que la versión en pruebas, como su propio nombre indica, es la más proclive a fallar y contiene lo último de lo último en lo que a novedades de software se refiere.

En el momento en que se realizó un primer estudio, la versión estable de Debian era Debian 3.0 (también conocida como Woody), la inestable recibía el sobrenombre de Sid y la que se encontraba en pruebas era Sarge. Pero en el pasado, Woody pasó también por una etapa inestable y, antes de eso, por otra en pruebas. Esto es importante, porque lo que vamos a considerar en este artículo son las diferentes versiones estables de Debian, desde que se publicara la versión 2.0, allá por 1998. Así, tenemos Debian 2.0 (alias Hamm), Debian 2.1 (Slink), Debian 2.2 (Potato) y, por último, Debian 3.0 (Woody).

Nota

Los apodos de las versiones de Debian corresponden a los protagonistas de la película de dibujos animados *Toy story*, una tradición que se implantó medio en serio medio en broma cuando se publicó la versión 2.0 y Bruce Perens, entonces líder del proyecto y después fundador de la Open Source Initiative y del término *open source*, trabajaba para la empresa que se encargaba de realizar esta película. Se pueden encontrar más detalles sobre la historia de Debian y la distribución Debian en general en "A brief history of Debian" [122].

9.9.1. Radiografía de Debian

Debian GNU/Linux es probablemente la mayor compilación de software libre que funciona de forma coordinada, y sin duda, uno de los productos software más grandes jamás construidos. La versión 4.0, liberada en abril de 2007 (denominada Etch), consta de más de diez mil cien paquetes fuente, que suman más de 288 millones de líneas de código.

El número de líneas de código en Debian 3.0 asciende a 105 millones. Según el modelo COCOMO habría que desembolsar una cantidad aproximada de 3.600 millones de dólares para obtener un software como el que viene empaquetado con esta distribución. Al igual que en el caso de Red Hat, se ha computado por separado el esfuerzo necesario para cada paquete y luego se han sumado todas las cantidades. Por la misma razón, el tiempo de desarrollo de Debian es de sólo siete años, ya que se considera que cada paquete puede realizarse paralelamente en el tiempo a los demás. Eso sí, de media habría que movilizar a cerca de cuatro mil desarrolladores durante esos siete años para lograrlo.

Nota

Tres mil seiscientos millones de dólares es el presupuesto que tiene asignado el VI Programa Marco de la Comisión Europea para actividades de investigación y desarrollo relacionadas con la sociedad de la información. También es la cifra que Telefónica prevé invertir en Alemania para implantar UMTS.

Tabla 20. Estado de Debian

Página web	http://www.debian.org
Inicio del proyecto	16/08/1993
Licencia	Las que cumplan las DFSG
Versión estudiada	Debian 4.0 (alias Etch)
Líneas de código fuente	288.500.000
Número de paquetes	10.106
Estimación de coste	$ 10.140 millones
Estimación de tiempo de ejecución	8,84 años
Número aproximado de desarrolladores (*maintainers*)	Unos 1.500
Herramientas de ayuda al desarrollo	Listas de correo, sistema de notificación de errores

El lenguaje de programación más utilizado en Debian 4.0 es C, con más de un 51% de las líneas de código. Sin embargo, como se mostrará un poco más adelante en este apartado, la importancia de C va remitiendo con el tiempo, ya que las primeras versiones de Debian contaban con hasta un 80% del código en C. Buena parte de la "culpa" del retroceso de C lo tiene el segundo lenguaje, C++, pero sobre todo la irrupción de los lenguajes de guión (*scripting languages*) como Perl, Python y PHP. Entre todos éstos se cuelan lenguajes como Lisp o Java (que en Debian está infrarrepresentado por su política de no admitir código que dependa de la máquina virtual privativa de Sun).

Tabla 21. Lenguajes de programación utilizados en Debian GNU/Linux 4.0

Lenguaje de programación	Líneas de código (en millones)	Porcentaje
C	155	51%
C++	55	19%
Shell	30	10%
Perl	8,1	2,9%
Lisp	7,7	2,7%
Python	7,2	2,5%
Java	6,9	2,4%
PHP	3,5	1,24%

En la tabla 22 se muestra la evolución de los lenguajes más significativos en Debian.

Tabla 22. Lenguajes más utilizados en Debian

Lenguaje	Debian 2.0		Debian 2.1		Debian 2.2		Debian 3.0	
C	19.400.000	76,67%	27.800.00	74,89%	40.900.000	69,12%	66.500.000	63,08%
C++	1.600.000	6,16%	2.800.000	7,57%	5.980.000	10,11%	13.000.000	12,39%
Shell	645.000	2,55%	1.150.000	3,10%	2.710.000	4,59%	8.635.000	8,19%
Lisp	1.425.000	5,64%	1.890.000	5,10%	3.200.000	5,41%	4.090.000	3,87%
Perl	425.000	1,68%	774.000	2,09%	1.395.000	2,36%	3.199.000	3,03%
Fortran	494.000	1,96%	735.000	1,98%	1.182.000	1,99%	1.939.000	1,84%
Python	122.000	0,48%	211.000	0,57%	349.000	0,59%	1.459.000	1,38%
Tcl	311.000	1,23%	458.000	1,24%	557.000	0,94%	1.081.000	1,02%

Existen lenguajes que podríamos considerar minoritarios que alcanzan puestos bastante altos en la clasificación. Esto se debe a que aun encontrándose presentes en un número reducido de paquetes, éstos son bastante grandes. Tal es el caso de Ada, que en tres paquetes (GNAT, un compilador de Ada; libgtkada, un enlace a la biblioteca GTK, y ASIS, un sistema para gestionar fuentes en Ada) aglutina 430.000 de un total de 576.000 líneas de código fuente que se han contabilizado en Debian 3.0 para Ada. Otro caso parecido es el de Lisp, que cuenta sólo en GNU Emacs y XEmacs con más de 1.200.000 líneas de las alrededor de cuatro millones en toda la distribución.

9.9.2. Comparación con otros sistemas operativos

Si dicen que todas las comparaciones son odiosas, las de software libre con software propietario lo son todavía más. Las radiografías tan detalladas de Red Hat Linux y Debian han sido posibles por su condición de software libre. El acceso al código (y a otra información que ha sido expuesta en este capítulo) es indispensable para estudiar a fondo las diferentes versiones en cuanto a número de líneas, paquetes, lenguajes de programación utilizados... Pero las ventajas del software libre van más allá, porque además, facilitan la revisión de terceras personas, bien grupos de investigación o bien sencillamente personas interesadas.

En los sistemas propietarios, en general, realizar un estudio así es tarea imposible. De hecho, las cuentas que se ofrecen a continuación tienen sus fuentes en las propias compañías que hay detrás del desarrollo de software, por lo que no podemos avalar su veracidad. Para más inri, en muchos casos no sabemos si se está hablando de líneas de código fuente físicas tal como hemos venido haciendo a lo largo de este capítulo, o si también incluyen en sus cuentas las líneas en blanco y las de comentarios. A esto hay que añadir que tampoco

sabemos a ciencia cierta lo que consideran en su software, de manera que para algunas versiones de Microsoft Windows no sabemos si incluyen el paquete de Microsoft Office o no.

En cualquier caso, y teniendo en cuenta todo lo que se ha comentado al respecto en párrafos anteriores, pensamos que incluir esta comparativa es interesante, ya que nos ayuda a situar las diferentes distribuciones de Red Hat y de Debian dentro de un panorama más amplio. Lo que parece estar fuera de toda duda es que tanto Debian como Red Hat, pero especialmente el primero, son las mayores colecciones de software vistas jamás por la humanidad hasta el momento.

Los números que se citan a continuación proceden de Mark Lucovsky [168] para Windows 2000, SUN Microsystems [171] para StarOffice 5.2, Gary McGraw [169] para Windows XP y Bruce Schneier [200] para el resto de sistemas. En la tabla 23 se muestra la comparativa en orden creciente.

Tabla 23. Comparación con sistemas propietarios

Sistema	Fecha de publicación	Líneas de código (aprox.)
Microsoft Windows 3.1	Abril 1992	3.000.000
SUN Solaris 7	Octubre 1998	7.500.000
SUN StarOffice 5.2	Junio 2000	7.600.000
Microsoft Windows 95	Agosto 1995	15.000.000
Red Hat Linux 6.2	Marzo 2000	18.000.000
Debian 2.0	Julio 1998	25.000.000
Microsoft Windows 2000	Febrero 2000	29.000.000
Red Hat Linux 7.1	Abril 2001	32.000.000
Debian 2.1	Marzo 1999	37.000.000
Windows NT 4.0	Julio 1996	40.000.000
Red Hat Linux 8.0	Septiembre 2002	50.000.000
Debian 2.2	Agosto 2000	55.000.000
Debian 3.0	Julio 2002	105.000.000

9.10. Eclipse

La plataforma Eclipse consiste en un entorno de desarrollo integrado (IDE, *integrated development environment*) abierto y extensible. Un IDE es un programa compuesto por un conjunto de herramientas útiles para un desarrollador de software. Como elementos básicos, un IDE cuenta con un editor de código, un compilador/intérprete y un depurador. Eclipse sirve como IDE Java y dispone

de numerosas herramientas de desarrollo de software. También da soporte a otros lenguajes de programación, como C/C++, Cobol, Fortran, PHP o Python. A la plataforma base de Eclipse se le pueden añadir extensiones (*plug in*) para extender la funcionalidad.

El término Eclipse además identifica a la comunidad de software libre para el desarrollo de la plataforma Eclipse. Este trabajo se divide en proyectos que tienen el objetivo de proporcionar una plataforma robusta, escalable y de calidad para el desarrollo de software con el IDE Eclipse. Está coordinado por la Fundación Eclipse, que es una organización sin fines de lucro creada para la promoción y la evolución de la plataforma Eclipse y que da soporte tanto a la comunidad como al ecosistema Eclipse.

9.10.1. Historia de Eclipse

Gran parte de la programación de Eclipse fue realizada por IBM antes de que se creara el proyecto Eclipse como tal. El antecesor de Eclipse fue VisualAge y se construyó usando Smalltalk en un entorno de desarrollo llamado Envy. Con la aparición de Java en la década de los noventa, IBM desarrolló una máquina virtual válida tanto para Smalltalk como para Java. El rápido crecimiento de Java y sus ventajas con miras a una Internet en plena expansión obligaron a IBM a plantearse el abandono de esta máquina virtual dual y la construcción de una nueva plataforma basada en Java desde el principio. El producto final resultante fue Eclipse, que ya había costado unos 40 millones de dólares a IBM en el año 2001.

A finales de 2001 IBM, junto a Borland, crearon la fundación sin fines de lucro Eclipse, abriéndose así al mundo de código abierto. A este consorcio se han unido poco a poco importantes empresas del desarrollo de software a escala mundial: Oracle, Rational Software, Red Hat, SuSE, HP, Serena, Ericsson y Novell, entre otras. Hay dos ausencias significativas: Microsoft y Sun Microsystems. Microsoft ha sido excluida por su posición de monopolio del mercado, y Sun Microsystems cuenta con su propio IDE, la principal competencia de Eclipse: NetBeans. De hecho, el nombre de Eclipse fue elegido porque el objetivo era crear un IDE capaz de "eclipsar a Visual Studio" (Microsoft) así como "eclipsar al sol" (Sun Microsystem).

La última versión estable de Eclipse se encuentra disponible para los sistemas operativos Windows, Linux, Solaris, AIX, HP-UX y Mac OS X. Todas las versiones de Eclipse necesitan tener instalado en el sistema una máquina virtual Java (JVM), preferiblemente JRE (Java Runtime Environment) o JDK (Java Developer Kit) de Sun, que a principios de 2007 no son libres (aunque hay un anuncio por parte de Sun de que lo serán).

9.10.2. Estado actual de Eclipse

Todo el trabajo desarrollado para el consorcio Eclipse se organiza en proyectos. Estos proyectos, a su vez, dividen el trabajo en subproyectos, y los subproyectos en componentes. Los proyectos de alto nivel son gestionados por comités de la Fundación Eclipse (PMC, *project management committees*). A continuación se enumeran los proyectos de alto nivel:

- Eclipse. Plataforma base para el resto de componentes. Dicha plataforma será libre, robusta, completa y de calidad para el desarrollo de aplicaciones ricas de cliente (RCP, *rich client plaftorm*) y herramientas integradas (*plug in*). El núcleo de ejecución de la plataforma Eclipse se llama Equinox y es una implementación de la especificación OSGi (Open Services Gateway Initiative), que describe una arquitectura orientada a servicios (SOA) para aplicaciones.

- Herramientas (ETP, *Eclipse tools project*). Herramientas varias y componentes comunes para la plataforma Eclipse.

- Web (WTP, *web tools project*). Herramientas para el desarrollo de aplicaciones web y JEE (Java Enterprise Edition).

- Pruebas y rendimiento (TPTP, *test and performance tools project*). Herramientas de pruebas y medida de rendimientos para que los desarrolladores puedan monitorizar sus aplicaciones y hacerlas más productivas.

- Informes web (BIRT, *business intelligence and reporting tools*). Sistema de generación de informes web.

- Modelado (EMP, *Eclipse modeling project*). Herramientas de desarrollo basado en modelos.

- Datos (DTP, *data tools platform*). Soporte para tecnologías centradas en el manejo de datos.

- Dispositivos empotrados (DSDP, *device software development platform*). Herramientas para el desarrollo de aplicaciones destinadas a ser ejecutadas en dispositivos limitados en hardware, esto es, dispositivos empotrados.

- Arquitectura orientada a servicios (SOA, *service oriented architecture*). Herramientas para el desarrollo de proyectos orientados a servicios.

- Tecnología Eclipse. Investigación, divulgación y evolución de la plataforma Eclipse.

Los principios que guían el desarrollo de la comunidad Eclipse siguen las líneas siguientes:

- Calidad. El software desarrollado en Eclipse debe seguir los patrones de calidad de la ingeniería del software.

- Evolución. La plataforma Eclipse, así como las herramientas en torno a ella, deben evolucionar dinámicamente de acuerdo con los requisitos de los usuarios.

- Meritocracia. Cuánto más se contribuye, más responsabilidades se tienen.

- Ecosistema Eclipse. Habrá recursos donados por la comunidad de código abierto al consorcio Eclipse. Estos recursos serán gestionados en beneficio de la comunidad.

El proceso de desarrollo seguido en Eclipse sigue unas fases determinadas. En primer lugar, hay una fase de prepropuesta, en la que un individuo o una empresa declaran su interés para establecer un proyecto. Si la propuesta es aceptada, se decide si será un proyecto de alto nivel o bien un subproyecto. El paso siguiente es validar el proyecto en términos de aplicabilidad y calidad. Tras esta etapa de incubación, tendrá lugar una revisión final. Si supera esta revisión, el proyecto habrá demostrado su validez de cara a la comunidad Eclipse, y se pasará a la fase de implementación.

9.10.3. Radiografía de Eclipse

Eclipse se distribuye bajo licencia EPL (Eclipse Public License). Esta licencia es considerada libre por la FSF y por la OSI. La licencia EPL permite usar, modificar, copiar y distribuir nuevas versiones del producto licenciado. El antecesor de EPL es CPL (Common Public License). CPL fue escrita por IBM, mientras que EPL es obra del consorcio Eclipse.

Estimar la inversión y el esfuerzo invertidos en Eclipse no es una tarea sencilla. Ello es debido a que el código fuente que compone el ecosistema Eclipse está distribuido en numerosos proyectos y repositorios de software.

A continuación, se muestra el resultado de aplicar el método COCOMO a la plaftorma Eclipse, que sirve de base al resto de *plug in*.

Tabla 24. Análisis de Eclipse

Página web	http://www.eclipse.org
Inicio del proyecto	2001
Licencia	Eclipse Public License

Versión analizada	3.2.2
Líneas de código fuente	2.163.932
Número de ficheros	15.426
Estimación de coste	$ 85.831.641
Estimación de tiempo de ejecución	6,22 años (74,68 meses)
Estimación de número medio de desarrolladores	102,10
Número aproximado de desarrolladores	133 *commiters*
Herramientas de ayuda al desarrollo	CVS, listas de correo, sistema de notificación de errores (Bugzilla)

En la tabla siguiente se muestran los lenguajes de programación usados en Eclipse 3.2.2:

Tabla 25. Lenguajes de programación utilizados en Eclipse

Lenguaje de programación	Líneas de código	Porcentaje
Java	2.066.631	95,50%
C	85.829	3,97%
Perl	3.224	0,06%
C++	5.442	0,25%
JSP	3.786	0,17%
Perl	1.325	0,06%
Lex	1.510	0,03%
Shell	849	0,04%
Python	46	0,00%
PHP	24	0,00%

10. Otros recursos libres

"If you want to make an apple pie from scratch, you must first create the universe."

"Si quieres hacer un pastel de manzana desde el principio, primero debes crear el Universo."

Carl Sagan

¿Se pueden extender las ideas de los programas libres a otros recursos? Podemos pensar que otros recursos de información fácilmente copiables electrónicamente son de naturaleza similar a los programas, por lo que les son aplicables las mismas libertades, reglas y modelos de desarrollo y negocio. Sin embargo, hay diferencias cuyas implicaciones han hecho que no se desarrollen con la misma fuerza que los programas. La principal es que basta con copiar los programas para que funcionen, mientras que desde que se copia otro tipo de información hasta que empieza a ser útil se ha de pasar por un proceso más o menos costoso, que puede ir desde el aprendizaje de un documento hasta la puesta en producción de un hardware descrito en un lenguaje apropiado.

10.1. Recursos libres más importantes

De la documentación de programas y otros documentos técnicos ya hemos hablado en el apartado 3.2.5. Hablaremos aquí de otro tipo de creaciones, que pueden ser también textuales, pero ya no relacionadas con el software, tanto en ámbitos científicos y técnicos como en el ámbito artístico.

10.1.1. Artículos científicos

El avance de la ciencia se debe en gran parte a que los investigadores que la hacen progresar para beneficio de la humanidad publican los resultados de sus trabajos en revistas de amplia difusión. Gracias a esa difusión los investigadores desarrollan un currículum que les permite progresar hacia puestos de mayor categoría y responsabilidad, a la vez que obtener ingresos a partir de contratos de investigación conseguidos gracias al prestigio alcanzado.

Así pues, esta difusión de artículos representa un *modelo de negocio* que se ha demostrado muy fructífero. Para que sea posible, se necesita una amplia difusión y calidad garantizada. La difusión se ve obstaculizada por gran cantidad de revistas existentes, de coste no despreciable, cuya adquisición sólo es posible con presupuestos generosos. La calidad se garantiza por medio de la revisión por parte de especialistas.

Por ello han surgido numerosas iniciativas de revistas en la Red, entre las que destacan la veterana *First Monday* ("*First Monday*: peer rewiewed journal on the Internet") [26] o el proyecto *Public Library Of Science* (PLOS –http://

www.publiclibraryofscience.org– [55]). En Directory of Open Access Journals" [22] se citan bastantes más. ¿Se debe permitir que personas distintas de los autores publiquen una modificación de un artículo? Hay objeciones que alegan desde una posible falta de calidad o una tergiversación de opiniones o resultados, hasta el peligro de plagio fácil que permite a algunos trepar sin esfuerzo y oscurecer los méritos de los verdaderos autores. Sin embargo, la obligación de citar al autor original y de pasar una revisión en una revista de prestigio puede contrarrestar esos problemas (*vid.* apartado 10.2.2).

Se ha querido establecer un paralelismo entre el software libre y la ciencia, ya que el modelo de desarrollo del primero implica la máxima difusión, la revisión por otros, presumiblemente expertos, y la reutilización de resultados ("Free software/free science", 2001) [154].

10.1.2. Leyes y estándares

Hay documentos cuyo carácter es normativo, que definen cómo deben hacerse las cosas, bien para facilitar la convivencia entre las personas, o bien para que programas o máquinas interoperen entre sí. Estos documentos requieren la máxima difusión, por lo que todo obstáculo a la misma es contraproducente. Por ello es comprensible que tengan un tratamiento especial, como ocurre con la Ley de la Propiedad Intelectual española:

> "No son objeto de propiedad intelectual las disposiciones legales o reglamentarias y sus correspondientes proyectos, las resoluciones de los órganos jurisdiccionales y los actos, acuerdos, deliberaciones y dictámenes de los organismos públicos, así como las traducciones oficiales de todos los textos anteriores."

La variante tecnológica de las leyes son las normas o estándares. En programación son especialmente importantes los protocolos de comunicaciones, bien entre máquinas remotas o bien entre módulos de la misma máquina. Es obvio que no debe limitarse su difusión, especialmente si queremos que florezcan los programas libres que interoperen con otros, pero a pesar de ello, tradicionalmente, los organismos de normalización, como la ISO[11] e ITU[12] , venden sus normas, incluso en formato electrónico, y prohíben su redistribución. Aunque esto pueda intentar justificarse para cubrir parcialmente los gastos, la libre difusión del texto de los estándares ha resultado mucho más productiva. éste es el caso de las recomendaciones del W3C[13] y, sobre todo, de las que gobiernan Internet, disponibles desde el principio en documentos llamados RFC, de *request for comments*, en formatos electrónicos legibles en cualquier editor de textos.

(11) International Organization for Standarlization

(12) International Telecommunications Union

(13) World Wide Web Consortium (Consorcio del Web)

Pero no es la disponibilidad la única causa del éxito de los protocolos de Internet. También lo es su *modelo de desarrollo*, muy similar al del software libre por su carácter abierto a la participación de cualquier interesado y por la utilización de listas de correo y de medios similares. En "The Internet standards process - revision 3" [94] y en "GNU make" [36] se describe este proceso.

¿Debe permitirse la modificación del texto de leyes y normas? Obviamente no si eso da lugar a confusión. Por ejemplo, sólo se admite que una RFC sea modificada para explicarla o añadirle comentarios aclaratorios, mientras que ni siquiera eso se permite sin autorización explícita para las recomendaciones del W3C (http://www.w3.org/Consortium/Legal/2002/copyright-documents-20021231) [65]. Las licencias son también documentos legales no modificables. ¿Debería permitirse la creación de nuevas normas derivadas de otras existentes a partir de los documentos originales? Probablemente eso llevaría a la proliferación fácil de normas similares e incompatibles que crearían confusión y podrían ayudar a empresas dominantes en el mercado a promover su propia variante incompatible, como de hecho ya está ocurriendo, especialmente en el ámbito del web. No obstante, en el caso de las legislaciones de los estados, muchas veces se han copiado literalmente leyes de otros países, adaptadas con pequeñas modificaciones a las particularidades locales.

¿Existe un modelo de negocio para las leyes y normas? En torno a las leyes hay multitud de profesionales que se encargan de su diseño, interpretación y de forzar su aplicación (legisladores, abogados, procuradores, jueces, etc.). En torno a las normas existen laboratorios que otorgan certificados de conformidad. Las organizaciones de normalización viven, o deberían vivir, de las aportaciones de miembros interesados en promover estándares, por ejemplo, porque su negocio esté basado en productos que interoperen.

De la misma manera que es conveniente tener una definición de *software libre* o *abierto*, también es necesaria una definición de *estándares abiertos*. Bruce Perens (http://perens.org/OpenStandards) [15] ha propuesto una basada en los principios siguientes:

1) Disponibilidad: si es posible, proporcionar incluso una implementación libre de referencia.

2) Maximización de las opciones del usuario final.

3) Sin tasas sobre la implementación (no así sobre la certificación, aunque aconseja la disponibilidad de herramientas libres de *autocertificación*).

4) Sin discriminación de implementador.

5) Permiso de extensión o restricción (no certificable).

6) Evitación de prácticas predatorias por parte fabricantes dominantes. Toda extensión propietaria debe tener una implementación libre de referencia.

10.1.3. Enciclopedias

En 1999 Richard Stallman lanzó la idea de una enciclopedia libre ("The free universal encyclopedia and learning resource", 2001) [210] como un mecanismo para evitar la apropiación del conocimiento y proporcionar acceso universal a documentación formativa. Estaría formada por artículos aportados por la comunidad, sin un control centralizado, donde distintos actores asumirían distintas funciones, entre las que se aconsejaba, pero no se obligaba, la de revisor. Esta enciclopedia no contendría solamente texto, sino también elementos multimedia y software educativo libre.

Han surgido varias iniciativas para realizar esta visión. Por ejemplo, Nupedia (http://www.nupedia.com) [178] trató sin éxito de construir una enciclopedia de calidad, quizá porque requería un formato relativamente difícil de aprender (TEI), aunque seguramente en mayor medida por el requisito de que todos los artículos necesitasen un editor, revisores científicos y de estilo, etc.

Heredera de Nupedia pero con mucho más éxito que ésta, ha sido Wikipedia (http://www.wikipedia.org) [69]. Wikipedia es una enciclopedia libre plurilingüe basada en la tecnología *wiki*. Wikipedia se escribe de forma cooperativa por voluntarios y permite que la gran mayoría de los artículos sean modificados por cualquier persona con acceso mediante un navegador web. Su éxito se basa en una estructura más flexible para la edición, que elimina los obstáculos que imponia Nupedia aproximándose más a la idea de libertad de Stallman. La palabra *wiki* proviene del hawaiano *wiki wiki* ('rápido'). La tecnología *wiki* permite a cualquiera editar cualquier documento por medio del sistema de texto estructurado, extraordinariamente simple, como se ha visto en el apartado 8.6.2. En febrero de 2007, el número de artículos en inglés en Wikipedia superaba la cifra de 1.500.000, y en español se han alcanzado los 200.000 artículos.

Nota

Wikipedia es un proyecto de la fundación sin fines de lucro Wikimedia, que mantiene además, con la misma fórmula que Wikipedia, los proyectos siguientes:

- Wikcionario (http://www.wiktionary.org) [66]. Se trata de un proyecto cooperativo para producir un diccionario multilingüe gratuito, con significados, etimologías y pronunciaciones, en aquellas lenguas en las que sea necesario.
- Wikilibros (http://www.wikibooks.org/) [67]. Es una colección de libros de contenido libre que tiene como objetivo poner a disposición de cualquier persona libros de texto, manuales, tutoriales u otros textos pedagógicos de contenido libre y de acceso gratuito.
- Wikiquote (http://www.wikiquote.org) [70]. Es una recopilación de frases célebres en todos los idiomas, que incluye las fuentes cuando éstas se conocen.
- Wikisource. Es una biblioteca de textos originales que se encuentran en dominio público o que se han publicado con licencia GFDL (licencia de documentación libre de GNU).
- Wikispecies (http://species.wikimedia.org/) [71]. Es un repertorio abierto de especies animales, vegetales, hongos, bacterias y de todas las formas de vida conocidas.
- Wikinoticias (http://wikinews.org/) [68]. Es una fuente de noticias de contenido libre en la que los usuarios son los redactores.
- Commons (http://commons.wikimedia.org/) [19]. Es un repositorio libre de imágenes y contenido multimedia.
- Wikiversidad (http://wikiversity.org/) [72]. Es una plataforma educativa en línea libre y gratuita, basada en proyectos de aprendizaje a cualquier nivel educativo.
- Meta-Wiki (http://meta.wikimedia.org/) [48]. Es el sitio web de apoyo a los proyectos de la Fundación Wikimedia.

Cabe destacar también la *Concise Encyclopedia of Mathematics*, con un concepto de libertad más limitado (sólo se puede consultar en la Red) y un modelo de desarrollo que necesariamente hace pasar todas las contribuciones por un comité editorial.

10.1.4. Cursos

Con la misma finalidad que las enciclopedias, se puede producir material docente libre, como apuntes, transparencias, ejercicios, libros, planificaciones o software didáctico. Existe una tendencia a ver a las universidades como un negocio de producción y venta de conocimiento que contradice sus principios. Los motivos por los que una universidad puede poner a disposición de todo el mundo estos materiales son los siguientes:

- El cumplimiento de su misión como agente difusor del conocimiento.
- El bajo coste que supone hacer disponibles materiales existentes a todo el mundo.
- El hecho de que estos materiales no sustituyen a la enseñanza presencial.

- La concepción de estos materiales como propaganda que puede atraer alumnos y que contribuye al prestigio de la universidad.

- La posibilidad de creación de una comunidad de docentes que revisen los materiales y los mejoren.

La iniciativa más notable en este sentido es la del MIT (http://ocw.mit.edu) [174], que prevé hacer accesibles de manera coherente y uniforme más de dos mil cursos bien catalogados.

10.1.5. Colecciones y bases de datos

La mera recolección de información siguiendo determinados criterios, ordenándola y facilitando su acceso es de por sí un producto de información valioso, independiente de la información en sí misma, sujeto por tanto a autoría y, por ello, a restricciones de las libertades de acceso, modificación y redistribución. Por tanto, si deseamos información libre, también podemos desear colecciones libres.

Por ejemplo, podemos querer clasificar la información relevante en Internet, organizando y comentando enlaces. Esto es lo que hace el ODP (Open Directory Project –http://dmoz.org– [109]), que opera Netscape y que mantienen editores voluntarios organizados según un esquema jerárquico. El directorio completo puede copiarse libremente en formato RDF y publicarse modificado de alguna manera, como hacen Google y otros muchos buscadores que lo aprovechan. Netscape, propietario del directorio, garantiza un *contrato social* ("Open Directory Project social contract") [53] inspirado en el de la distribución Debian (http://www.debian.org/social_contract.html) [106], que facilita la colaboración exterior asegurando que el Open Directory Project siempre será libre, con políticas públicas, autorregulado por la comunidad y con los usuarios como primera prioridad.

Otro ejemplo de colecciones interesantes para nosotros son las distribuciones de software libre, con los programas modificados para que encajen perfectamente entre sí y precompilados para que se puedan ejecutar fácilmente.

10.1.6. Hardware

La libertad en el hardware tiene dos aspectos. El primero es la necesidad de que las interfaces y los juegos de instrucciones sean abiertos, de manera que cualquiera pueda realizar un manejador de dispositivo o un compilador para una arquitectura. El segundo es disponer de la información y el poder suficientes como para reproducir un diseño hardware, modificarlo y combinarlo con otros. Los diseños pueden considerarse software en un lenguaje apropiado (VHDL, Verilog, etc.). Sin embargo, hacerlos funcionar no es fácil,

ya que hay que fabricarlos, lo cual es caro y lento. Sin embargo existen iniciativas en este sentido, entre las que podemos resaltar OpenCores (http://www.opencores.org) [52], para circuitos integrados.

10.1.7. Literatura y arte

Para terminar nuestro recorrido por los recursos libres, no podemos olvidar el arte y la literatura, cuyo fin último no es tanto utilidad como estética. ¿Qué razones puede tener un artista para conceder libertades de copia, modificación y redistribución? Por un lado, darse a conocer y favorecer la difusión de su obra, lo que puede permitirle obtener ingresos por otras actividades, como conciertos y obras de encargo, y por otro, favorecer la experimentación y la creatividad. En el arte ocurre lo mismo que en la técnica: la innovación es incremental, y a veces es difícil distinguir el plagio de la pertenencia a un mismo movimiento o corriente artística.

Obviamente no son lo mismo la creación que la interpretación, ni la música que la literatura. La música, la pintura, la fotografía y el cine son muy parecidos a los programas, en el sentido de que se los hace "funcionar" inmediatamente en un ordenador, mientras que con la escultura, por ejemplo, no se puede. No existen muchas iniciativas en arte y literatura libres, y éstas son muy diversas. Podemos mencionar las novelas del colectivo Wu Ming (http://www.wumingfoundation.com) [29].

10.2. Licencias de otros recursos libres

Las licencias de software libre han sido fuente de inspiración para otros recursos intelectuales, de tal modo que muchos de ellos las han adoptado directamente, especialmente en el caso de la documentación, y otras veces las han adaptado ligeramente, como la pionera Open Audio License (http://www.eff.org/IP/Open_licenses/eff_oal.html) [114]. La mayoría de ellas son de tipo *copyleft* si permiten trabajos derivados.

Para cualquier tipo de textos se ha utilizado y se utiliza bastante la licencia de documentación libre de GNU (*vid.* apartado 10.2.1) aunque cada vez se están aceptando más las licencias de Creative Commons (*vid.* apartado 10.2.2).

Incluso se han utilizado licencias de programas (GPL y LGPL) para hardware, aunque esta materia es compleja y difícil de conciliar con la legalidad vigente. En efecto, los diseños y los diagramas pueden ser utilizados, sin ser copiados físicamente, para extraer ideas que se utilicen para nuevos diseños cerrados. Por ejemplo, la OpenIPCore Hardware General Public License ("OpenIPCore hardware general public license") [155] recoge la prohibición de esta apropiación, pero su legalidad es dudosa [209]. La única posibilidad de proteger esas

ideas es por medio de algún tipo de patente libre, algo aún no desarrollado y fuera del alcance de los que no tienen intención o posibilidad de hacer negocio con ellas.

10.2.1. Licencia de documentación libre de GNU

Una de las licencias de tipo *copyleft* más conocidas para la documentación técnica, bien de programas o bien de cualquier otra cosa, es la de la Free Software Foundation. Después de darse cuenta de que un documento no es lo mismo que un programa, Richard Stallman promovió una licencia para los documentos que acompañasen a los programas y para otros documentos de carácter técnico o didáctico.

Para facilitar el desarrollo de versiones derivadas, hay que poner a disposición de quien lo requiera una copia *transparente* del documento, en el sentido de lo que se ha explicado en el apartado 3.2.5, además de las copias *opacas*, en una analogía entre los códigos fuente y los objetos de los programas.

Una de las preocupaciones de la licencia es reconocer la autoría e impedir que se tergiversen ideas u opiniones expresadas por el autor. Para ello exige que las obras derivadas exhiban en la portada un título distinto de los de las versiones anteriores (salvo permiso expreso) y que se nombre expresamente de dónde se puede conseguir el original. También deben listarse como autores los más importantes de los originales, además de los de las modificaciones, y se han de conservar todas las notas sobre derechos de autor. Además, se deben conservar agradecimientos y dedicatorias, y se debe respetar el apartado de historia, si lo tiene, a la hora de añadir las modificaciones nuevas. Incluso, y esto es lo más criticado de la licencia, pueden nombrarse secciones invariantes y textos de cubiertas, que nadie puede modificar ni eliminar, si bien la licencia sólo permite considerar invariantes textos *no técnicos*, que la misma llama *secundarios*.

Esta licencia ha generado una gran polémica en el mundo del software libre, hasta tal punto que en la distribución Debian se está discutiendo (en el momento de publicar este libro) si se elimina o se pasa a la apartado extraoficial *no libre* todo documento con la misma. Incluso aunque no existan secciones invariantes, como los trabajos derivados deben regirse por la misma licencia, las pueden añadir. Se argumenta, por ejemplo, que puede haber secciones invariantes incorrectas u obsoletas, pero que deben mantenerse. En todo caso la licencia es incompatible con las directrices de software libre de Debian (http://www.debian.org/social_contract.html#guidelines) [104], pero la cuestión quizá esté en si la documentación debe cumplirlas (por ejemplo, los textos de las licencias tampoco se pueden modificar).

Sugerencia

Las primeras versiones de este texto estuvieron amparadas por la licencia GFDL, pero posteriormente los autores decidimos usar una licencia de Creative Commons (*vid.* apartado 10.2.2), quizás más adaptada a las características de un libro.

10.2.2. Licencias de Creative Commons

Creative Commons (http://creativecommons.org) [21] es una organización sin fines de lucro fundada en 2001 por expertos en propiedad intelectual, derecho en la sociedad de la información e informática con el propósito de fomentar la existencia, la conservación y la accesibilidad de recursos intelectuales cedidos a la comunidad de diversas maneras. Está basada en la idea de que algunas personas pueden no querer ejercer sobre su obra todos los derechos de propiedad intelectual que les permite la ley, ya que eso puede impedir una amplia distribución.

A finales de 2002 vieron la luz las primeras licencias Creative Commons para trabajos creativos, que pasaron por diversas versiones. Han sido creadas para ser:

- robustas, como para resistir el escrutinio de un tribunal en diferentes países;
- sencillas, para poder ser usadas por personas que no son especialistas en asuntos legales;
- sofisticadas, para ser identificadas por varias aplicaciones de la web.

Las distintas licencias permiten al autor seleccionar qué tipo de libertades cede, además de la de copia, según cuatro dimensiones:

-  o  Reconocimiento (Attribution). El material creado puede ser distribuido, copiado y exhibido por terceras personas si se muestra en los créditos el autor original.
-  o  No comercial (Non Commercial). El material original y los trabajos derivados pueden ser distribuidos, copiados y exhibidos mientras su uso no sea comercial.
-  Sin obra derivada (No Derivative Works). El material creado puede ser distribuido, copiado y exhibido, pero no se puede utilizar para crear un trabajo derivado del original.
-  Compartir igual (Share Alike). El material creado puede ser modificado y distribuido, pero bajo la misma licencia que el material original.

En la versión 1.x de las licencias Creative Commons se recogían once tipos de licencias, que combinaban las cuatro características básicas arriba descritas. Un 98% de los autores elegía la opción "atribución", con lo que en la versión 2.x de las licencias Creative Commons la atribución es un requisito. De esta forma, se reducen los once tipos de licencia a seis, que son las siguientes:

- Reconocimiento.
- Reconocimiento - Sin obra derivada.
- Reconocimiento - Sin obra derivada - No comercial.
- Reconocimiento - No comercial.
- Reconocimiento - No comercial - Compartir igual.
- Reconocimiento - Compartir igual.

En la tabla siguiente se muestran de forma esquemática las licencias según su icono representativo. Este icono suele ser un enlace al resumen de la licencia, alojada en [21].

	Permite modificaciones.	Permite modificaciones si se comparte de la misma manera.	No permite modificaciones.
Permite uso comercial.	BY	BY SA	BY ND
No permite uso comercial.	BY NC	BY NC SA	BY NC ND

(14)

En lugar del icono representativo de la licencia se puede usar el icono genérico[14], aunque el enlace debe ser a la licencia que el autor elija. El código HTML del enlace de la licencia puede obtenerse de Creative Commons [21]. Una vez escogida la licencia y añadido el icono correspondiente, se habrá licenciado la obra, de manera que se obtendrá:

- *Commons deed*. Es un resumen del texto legal con los iconos relevantes de la licencia. Este resumen es el destino del enlace obtenido de Creative Commons [21].

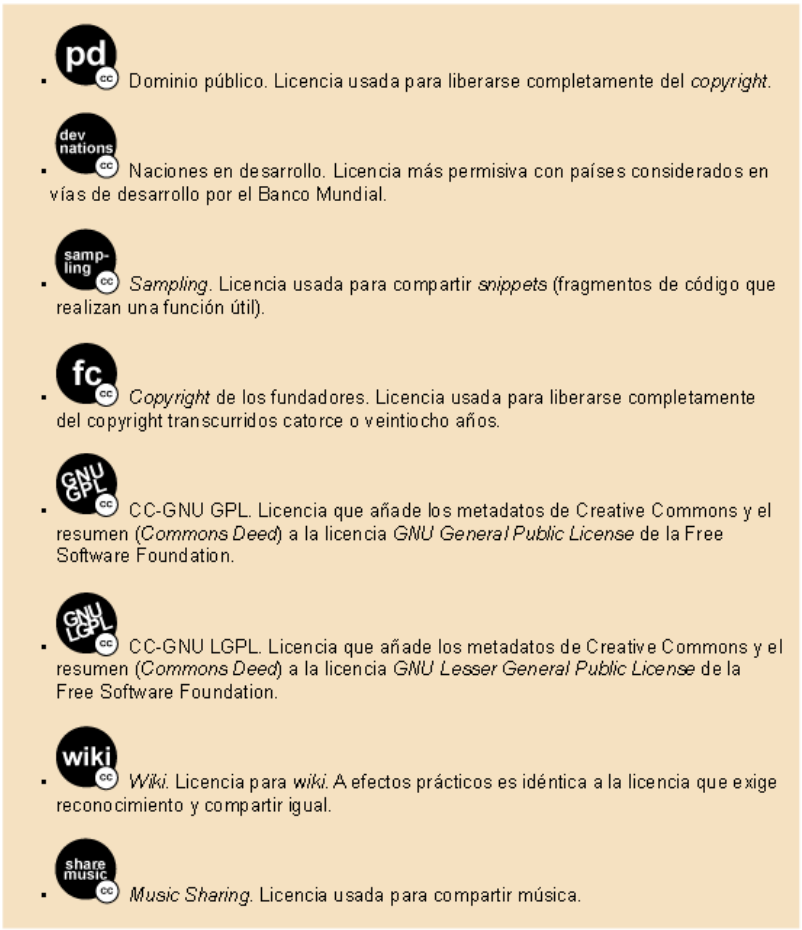
- *Legal Code*. Es el código legal completo en el que se basa la licencia. A este texto se puede acceder desde el resumen anterior.

- *Digital code*. Es la descripción RDF (*resource description framework*), que sirve para que los motores de búsqueda y otras aplicaciones identifiquen la licencia de la obra y sus condiciones de uso.

En febrero de 2007 se publicaron las licencias Creative Commons en su versión 3.0. Se trata de una actualización para corregir muchas pegas que se les atribuían. El primer gran cambio es que la licencia genérica deja de ser la estadounidense para ser una versión basada en la terminología del Tratado de Berna. En segundo lugar, se mencionan específicamente los derechos morales

y la sociedad de gestión, ya que antes en cada jurisdicción se habían tomado decisiones diferentes. En tercer y último lugar, se ha modificado el texto tanto del *commons deed* como del *legal code* que acompaña cada licencia para dejar claro que la cláusula de reconocimiento de autoría no permite al licenciatario dar a entender que tiene una relación o asociación con el licenciador.

Creative Commons permite además otro tipo de licencias para aplicaciones específicas. Son las siguientes:



No todas las licencias Creative Commons son consideradas libres desde sectores afines al software libre, ya que para ello deberían conceder las cuatro libertades esenciales (*vid.* apartado 1.1.1) Benjamin "Mako" Hill (desarrollador de Debian y de Ubuntu) ha creado la web Freedomdefined.org (http://freedomdefined.org/) [28], con el objetivo de definir mejor qué es cultura libre y qué no. Sobre la base de esto, de las seis licencias básicas Creative Commons sólo dos son estrictamente libres: reconocimiento (BY) y reconocimiento-compartir igual (BY-SA), esta última además con *copyleft*.

Bibliografía

[1] Aap Project: http://www.a-a-p.org

[2] Ada Core Technologies: http://www.gnat.com/

[3] Alcôve: http://www.alcove.com

[4] Alcôve-Labs: http://www.alcove-labs.org

[5] Alioth: http://alioth.debian.org

[6] Anjuta: http://www.anjuta.org

[7] The Apache Ant Project: http://ant.apache.org

[8] Arch Revision Control System: http://www.gnu.org/software/gnu-arch/

[9] artofcode LLC: http://artofcode.com/

[10] Autoconf: http://www.gnu.org/software/autoconf

[11] Barrapunto: http://barrapunto.com

[12] Bazaar GPL Distributed Version Control Software: http://bazaar-vcs.org/

[13] Berlios. The Open Source Mediator: http://berlios.de

[14] Bitkeeper Source Management: http://www.bitkeeper.com

[15] Bruce Perens: http://perens.com/OpenStandards/Definition.html

[16] Caldera: http://www.sco.com

[17] Cisco Enterprise Print System: http://ceps.sourceforge.net/

[18] Code::blocks: http://www.codeblocks.org

[19] Commons: http://commons.wikimedia.org/

[20] Concurrent Version System: http://ximbiot.com/cvs/

[21] Creative Commons. http://creativecommons.org

[22] Directory of Open Access Journals: http://www.doaj.org

[23] Eclipse - An Open Development Platform: http://www.eclipse.org

[24] eCos: http://sources.redhat.com/ecos/

[25] eCos license 2.0: http://www.gnu.org/licenses/ecos-license.html

[26] *First Monday. Peer Rewiewed Journal on the Internet*: http://firstmonday.org

[27] Free Software Foundation: http://www.fsf.org

[28] Freedom Defined (Free Cultural Works): http://freedomdefined.org/

[29] Fundación Wu Ming: http://www.wumingfoundation.com

[30] GForge: http://gforge.org

[31] Gettext: http://www.gnu.org/software/gettext

[32] GNU Automake: http://www.gnu.org/software/automake

[33] GNU Emacs: http://www.gnu.org/software/emacs/

[34] GNU Libc: http://www.gnu.org/software/libc

[35] GNU Libtool: http://www.gnu.org/software/libtool

[36] GNU Make: http://www.gnu.org/software/make/make.html

[37] GNU Troff: http://www.gnu.org/software/groff/groff.html

[38] "Have you seen these hackers?": http://www.mozilla.org/MPL/missing.html

[39] "History of TeX": http://www.math.utah.edu/software/plot79/tex/history.html

[40] IBM Public License Version 1.0: http://opensource.org/licesenses/ibmpl.php

[41] Jam Product Information: http://www.perforce.com/jam/jam.html

[42] KDevelop: http://www.kdevelop.org

[43] Launchpad: https://launchpad.net

[44] The Linux Documentation Project: http://www.tldp.org

[45] LinuxCare: http://www.levanta.com

[46] Mailman, the GNU Mailing List Manager: http://www.list.org

[47] The Malone Bug Tracker: https://launchpad.net/products/malone

[48] Metawiki: http://meta.wikimedia.org/

[49] Mozilla Public License 1.1: http://www.mozilla.org/MPL/MPL-1.1.html

[50] Mozilla Tinderbox: http://www.mozilla.org/tinderbox.html

[51] NetBeans: http://www.netbeans.org

[52] Open Cores: http://www.opencores.org

[53] Open Directory Project Social Contract:

[54] Open Source Initiative: http://www.opensource.org

[55] Public Library of Science: http://www.publiclibraryofscience.org

[56] Red Hat: http://www.redhat.com

[57] Savannah: http://savannah.gnu.org y http://savannah.nongnu.org

[58] Slashdot: News for Nerds. http://slashdot.org

[59] Sleepycat License: http://www.sleepycat.com/download/oslicense.html

[60] Sleepycat Software: http://www.sleepycat.com/

[61] SourceForge: Open Source Software Development Website: http://sourceforge.net

[62] Subversion: http://subversion.tigris.org

[63] Texinfo - The GNU Documentation System:

[64] Tigris.org: Open Source Software Engineering: http://tigris.org

[65] W3c Document License:

http://www.w3.org/Consortium/Legal/2002/copyright-documents-20021231

[66] Wikcionario: http://www.wiktionary.org

[67] Wikilibros: http://www.wikibooks.org/

[68] Wikinoticias: http://wikinews.org/

[69] Wikipedia: http://www.wikipedia.org

[70] Wikiquote: http://www.wikiquote.org

[71] Wikispecies: http://species.wikimedia.org/

[72] Wikiversidad: http://wikiversity.org/

[73] X Window System Release 11 License: http://www.x.org/Downloads_terms.html

[74] Ximian: http://www.novell.com/linux/ximian.html

[75] Zope Corporation: http://www.zope.com/

[76] Zope Public License 2.0: http://www.zope.org/Resources/ZPL

[77] Ley de Propiedad Intelectual. Real Decreto Legislativo 1/1996, de 12 de abril (abril 1996):

[78] Affero General Public License, 2002: http://www.affero.org/oagpl.html

[79] Ley de Propiedad Intelectual. Ley 23/2006, de 7 de julio (julio 2006):

[80] Flossimpact Study. Technical Report, European Comission, 2007: http://flossimpact.eu

[81] ISO JTC 1/SC 34. Standard Generalized Markup Language (SGML, ISO 8879), 1986:

[82] **Antoniades, I.; Samoladas, I.; Stamelos, I.; Bleris, G. L.** "Dynamical simulation models of the open source development process" En: Koch [157].

http://wwwai.wu-wien.ac.at/~koch/oss-book/

[83] **Bailey, E. C.** (1998). *Maximum RPM. Taking the Red Hat package manager to the limit*. http://rikers.org/rpmbook/

[84] **González Barahona, J. M.** (2000). "Software libre, monopolios y otras yerbas". *Todo Linux* (3). http://sinetgy.org/~jgb/articulos/soft-libre-monopolios/

[85] **González Barahona, J. M.** (2002). "¿Qué se hace con mi dinero?". *Todo Linux* (17).

http://sinetgy.org/~jgb/articulos/sobre-administracion/

[86] **González Barahona, J. M.; Robles, G.** Libre Software Engineering Web Site.

http://libresoft.dat.escet.urjc.es/

[87] **González Barahona, J. M.; Robles, G.** (2003, mayo). "Unmounting the *code god* assumption". En: *Proceedings of the Fourth International Conference on eXtreme Programming and Agile Processes in Software Engineering*. Génova, Italia.

[88] **González Barahona, J. M.; Robles, G.; Ortuño Pérez, M. A.; Rodero Merino, L.; Centeno González, J.; Matellán Olivera, V.; Castro Barbero, E. M.; De las Heras Quirós; P.** "Anatomy of two GNU/Linux distributions". En: Koch [157].

http://wwwai.wu-wien.ac.at/~koch/oss-book/

[89] **Barnson, M. P.***The Bugzilla guide*.

http://www.bugzilla.org/docs214/html/index.html

[90] **Baudis, P.** "Cogito manual page".

http://www.kernel.org/pub/software/scm/cogito/docs/

[91] **Bezroukov, N.** (1998, diciembre). "A second look at the cathedral and the bazaar". *First Monday*, 4(12).

http://www.firstmonday.org/issues/issue4_12/bezroukov/index.html

[92] **Bodnar, L.** (2003). "Linux distributions. Facts and figures".

http://www.distrowatch.com/stats.php?section=packagemanagement

[93] **Boehm, B. W.** (1981). *Software Engineering Economics*. Prentice Hall.

[94] **Bradner, S.** (1996, octubre). "The Internet standards process. Revision 3 (rfc 2026, bcp 9)".

http://www.ietf.org/rfc/rfc2026.txt

[95] **Cederqvist, P.; GNU** (1993). "CVS - concurrent versions system". http://www.gnu.org/manual/cvs/index.html

[96] **Collins-Sussman, B.; Fitzpatrick; B. W.; Pilato, C. M.** (2004). *Version control with Subversion*. O'Reilly & Associates (http://www.ora.com).

http://svnbook.red-bean.com/

[97] **Cunningham, W.** "Wiki design principles".

[98] **Dachary, L.** (2001). "Savannah, the next generation".

http://savannah.gnu.org/docs/savannah-plan.html

[99] **Junta de Andalucía** (2003, marzo). Decreto 72/2003, de 18 de marzo, de Medidas de Impulso de la Sociedad del Conocimiento en Andalucía.

http://www.andaluciajunta.es/SP/AJ/CDA/Ficheros/ArchivosPdf/DecretoConocimiento.pdf

[100] **De Boor, A.***Pmake. A tutorial*. http://docs.freebsd.org/44doc/psd/12.make/paper.html

[101] **De Icaza, M.** "The story of the GNOME Project".

http://primates.ximian.com/~miguel/gnome-history.html

[102] **Senado de la República Francesa**. Forum sur la proposition de loi tendant à généraliser dans

l'administration l'usage d'Internet et de logiciels libres.

http://www.senat.fr/consult/loglibre/index.htm

[103] **De las Heras Quirós, P.; González Barahona, J. M.** (2000). "Iniciativas de las administraciones públicas en relación al software libre". *Bole. TIC, Revista de ASTIC* (14).

[104] **Debian.** "Debian free software guidelines".

http://www.debian.org/social_contract.html#guidelines

[105] **Debian.***Debian policy manual*.

http://www.debian.org/doc/debian-policy/

[106] **Debian.** "Debian social contract".

http://www.debian.org/social_contract.html

[107] **Schriftenreihe der KBSt** (2003, julio). Leitfaden für die migration von basissoftwarekomponenten auf serverund arbeitsplatzsystemen. Technical report, Koordinierungs- und Beratungsstelle der Bundesregierung für Informationstechnik in der Bundesverwaltung (KBSt).

http://www.kbst.bund.de/download/mlf_v1_de.pdf

[108] **DiBona, C.; Ockman, S.; Stone, M.** (ed.) (1999). *Open sources. Voices from the open source revolution*. O'Reilly & Associates.

http://www.oreilly.com/catalog/opensource/

[109] **Open Directory Project**. http://dmoz.org

[110] **Ehrenkrantz, J. R.** (2003, mayo). "Release management within open source projects". En: *Proceedings of the 3rd Workshop on Open Source Software Engineering at the 25th International Conference on Software Engineering*.Portland, EE.UU.

[111] **Consejo Europeo** (1991). Directiva 91/250/CEE del Consejo, de 14 de mayo de 1991, relativa a la protección jurídica de programas de ordenador.

http://europa.eu.int/scadplus/leg/es/lvb/l26027.htm

[112] **Feller, J.; Fitzgerald, B; Hissam, S.; Lakhani, K.** (ed.) (2003). *Making sense of the bazaar*. O'Reilly.

[113] **Fogel, K.; Bar, M.** (2001). *Open source code development with CVS* (2ª edición). Paragliph Press.

http://cvsbook.red-bean.com

[114] **Electronic Frontier Foundation.** Open Audio.

http://www.eff.org/IP/Open_licenses/eff_oal.html

[115] **Free Software Foundation.** GPLv3.

http://gplv3.fsf.org

[116] **Free Software Foundation.** LGPLv3. First discussion draft.

http://gplv3.fsf.org/pipermail/info-gplv3/2006-July/000008.html

[117] **Free Software Foundation** (1985): "The GNU Manifesto".

http://www.gnu.org/philosophy/

[118] **Free Software Foundation** (1991, junio). GNU General Public License, version 2. http://www.fsf.org/licenses/gpl.html

[119] **Free Software Foundation** (1999, febrero). GNU Lesser General Public License, version 2.1.

http://www.fsf.org/licenses/lgpl.html

[120] **Free Software Foundation.** "Free software definition".

http://www.gnu.org/philosophy/free-sw.html

[121] **Free Software Foundation.** "Licencias libres".

http://www.gnu.org/licenses/license-list.html

[122] **Garbee, B.; Koptein, H.; Lohner, N.; Lowe, W.; Mitchell, B.; Murdock, I.; Schulze, M.; Small, C.** "A brief history of Debian". En el paquete: *Debian-history*.

[123] **Germán, D.** (2002, mayo). "The evolution of GNOME. En: *Proceedings of the 2nd Workshop on Open Source Software Engineering at the 24th International Conference on Software Engineering*. Florida, EE.UU.

[124] **Germán, D.; Mockus, A.** (2003, mayo): "Automating the measurement of open source projects". En: *Proceedings of the 3rd Workshop on Open Source Software Engineering at the 25th International Conference on Software Engineering*. Portland, EE.UU.

[125] **Ghosh, R. A.** (1998, marzo). "Cooking pot markets: an economic model for the trade in free goods and services on the Internet. *First Monday*, 3(3).

http://www.firstmonday.dk/issues/issue3_3/ghosh/index.html

[126] **Ghosh, R. A.; Glott, R.; Krieger, B.; Robles, G.** (2002). *Free/libre and open source software: Survey and study*. Parte iv: "Survey of developers".

http://www.infonomics.nl/FLOSS/report/FLOSS_Final4.pdf

[127] **Ghosh, R. A.; Prakash, V. V.** (2000, julio). "The orbiten free software survey". *First Monday*, 5(7).

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

[128] **Godfrey, M. W.; Tu, Q.** (2000, agosto). "Evolution in open source software. A case study". En: *Proceedings of the 2000 International Conference on Software Maintainance*.

[129] **González, J. A.** (2002, marzo). "Carta al congresista Villanueva".

http://www.gnu.org. pe/mscarta.html

[130] **Goosens, M.; Rahtz, S.** (1999). *The LaTeX Web Companion*. Addison Wesley.

[131] **Grad, B.** (2002, enero-marzo). "A personal recollection: IBM's unbundling of software and services". En: *IEEE Annals of the History of Computing*, 24(1):64-71.

[132] **Working Group on Libre Software** (1999). "Free software / open source. Information society opportunities for Europe?".

http://eu.conecta.it/paper.pdf

[133] **GrULIC.** "Legislación sobre el uso de software libre en el Estado".

http://proposicion.org.ar/doc/referencias/index.html.es

[134] **Hamerly, J; Paquin, T.; Walton, S.** (1999). "Freeing the source. The story of Mozilla". http://www.oreilly.com/catalog/opensources/book/netrev.html

[135] **Hammel, M. J.** (1991, diciembre). "The history of xfree86". *Linux Magazine*.

http://www.linux-mag.com/2001-12/xfree86_01.html

[136] **Harris, S.** (2001, agosto). *The Tao of IETF. A novice's guide to the Internet engineering task force* (RFC 3160, FYI 17).

http://www.ietf.org/rfc/rfc3160.txt

[137] **Harrison, P.** (2002). "The rational street performer protocol".

http://www.logarithmic.net/pfh/RSPP

[138] **Hasan, R.** "History of Linux".

http://ragib.hypermart.net/linux/

[139] **Hauben, M.; Hauben, R.** (1997). *Netizens. On the history and impact of Usenet and the Internet*. IEEE Computer Society Press.

[140] **Healy, K.; Schussman; A.** (2003, enero). "The ecology of open source software development". http://opensource.mit.edu/papers/healyschussman.pdf

[141] **Hecker, F.** (1998, mayo). "Setting up shop. The business of open-source software".

http://www.hecker.org/writings/setting-up-shop.html

[142] **Hecker, F.** (1998). "Setting up shop. The business of open-source software".

http://www.hecker.org/writings/setting-up-shop.html

[143] **Hertel, G.; Niedner, S.; Herrmann, S.** (2003). "Motivation of software developers in open

source projects. An Internet-based survey of contributors to the Linux kernel".

http://opensource.mit.edu/papers/rp-hertelniednerherrmann.pdf

[144] **Himanen, P.** (2001). *The hacker ethic and the spirit of the information age*. Random House.

http://www.hackerethic.org

[145] **Hunt, F.; Johnson, P.** (2002). "On the Pareto distribution of SourceForge projects. Technical report". Centre for Technology Management, Cambridge University Engineering Department, Mill Lane, Cambridge CB2 1RX.

http://www-mmd.eng.cam.ac.uk/people/fhh10/Sourceforge/Sourceforge%20paper.pdf

[146] **Open Source Initiative.** "History of the OSI".

http://www.opensource.org/docs/history.php

[147] **Hamilton, J. R.** (embajador de EE.UU. en Perú) (2002, junio). "Carta al presidente del Congreso de la República".

http://www.gnu.org.pe/lobbyusa-congreso.html

[148] **Jones, P.** (2000, mayo). "Brook's law and open source. The more the merrier?".

http://www-106.ibm.com/developerworks/opensource/library/os-merrier.html?dwzone=opensource

[149] **Jorgensen, N.** "Incremental and decentralized integration in FreeBSD". En: Feller *et al*. [112]. http://www.dat.ruc.dk/~nielsj/research/papers/bazaar-freebsd.pdf

[150] **Brooks, F. P.** (1975). *The mythical man-month. Essays on software engineering*. Addison-Wesley.

[151] **Kalt, C.** (2000, abril). "Internet relay chat: architecture (RFC 2810)".

http://www.ietf.org/rfc/rfc2810.txt

[152] **Kelsey, J.; Schneier, B.** (1998, noviembre). "The street performer protocol". En: *Third USENIX Workshop on Electronic Commerce Proceedings*. USENIX Press.

http://www.counterpane.com/street_performer.html

[153] **Kelsey, J.; Schneier, B.** (1999, junio). "The street performer protocol and digital copyrights". *First Monday*, 4(6).

http://www.firstmonday.dk/issues/issue4_6/kelsey/

[154] **Kelty, C. M.** (2001, diciembre). "Free software/free science". *First Monday*, 6(12).

http://firstmonday.org/issues/issue6_12/kelty/index.html

[155] **Khatib, J.** "OpenIPCore Hardware General Public License".

http://www.opencores.org/OIPC/OHGPL_17.shtml

[156] **Knuth, D.** (1989). *The TeXbook*. Addison Welsley.

[157] **Koch, S.** (ed.) (2003). *Free/open source software development*. Idea Group Inc.

http://wwwai.wu-wien.ac.at/~koch/oss-book/

[158] **Koch, S.; Schneider, G.** (2000). "Results from software engineering research into open source development projects using public data". En: *Diskussionspapiere zum Tätigkeitsfeld Informationsverarbeitung und Informationswirtschaft, H.R. Hansen und W.H. Janko (Hrsg.), Nr. 22*, WirtschaftsuniversitätWien.

[159] **Kovács, G. L.; Drozdik, S.; Succi, G.; Zuliani, P.** (2004). "Open source software for the public administration". En: *Proceedings of the 6th International Workshop on Computer Science and Information Technologies* (CIST 2004). Budapest, Hungría.

[160] **Krishnamurthy, S.** (2002, mayo). "Cave or community? An empirical examination of 100 mature open source projects". *First Monday*, 7(6).

http://www.firstmonday.dk/issues/issue7_6/krishnamurthy/index.html

[161] **Laffitte; Trégouet; Cabanel** (1999). Proposition de loi numéro 495. Senado de la República Francesa.

http://www.senat.fr/consult/loglibre/texteloi.html

[162] **Laffitte; Trégouet; Cabanel** (2000). Proposition de loi numéro 117. Senado de la República Francesa.

http://www.senat.fr/consult/loglibre/texteloi.html

[163] **Lamport, L.** (1994). *LaTeX user's guide and reference manual* (2ª edición). Addison Welsley, Reading, Mass.

[164] **Lancashire, D.** (2001, diciembre). "Code, culture and cash. The fading altruism of open source development". *First Monday*, 6(12).

http://www.firstmonday.dk/issues/issue6_12/lancashire/index.html

[165] **Lehman, M. M.; Ramil, J. F; Wernick, P. D.** (1997, noviembre). "Metrics and laws of software evolution. The nineties view". En: *Proceedings of the 4th International Symposium on Software Metrics*.

http://www.ece.utexas.edu/~perry/work/papers/feast1.pdf

[166] **Leiner, B. M.; Cerf, V. G.; Kahn, R. E.; Clark, D. D.; Kleinrock, L.; Lynch, D. C.; Postel, J.; Roberts, L. G.; Wolff, S.** (1997). "A brief history of the Internet". En: *Communications of the ACM*.

http://www.isoc.org/internet/history/brief.shtml

[167] **Netcraft Ltd. August 2003 Web Server Survey, 2003.**

http://news.netcraft.com/archives/2003/08/01/august_2003_web_server_survey.html

[168] **Lucovsky, M.** (2000). "From NT OS/2 to Windows 2000 and beyond. A software-engineering odyssey".

http://www.usenix.org/events/usenix-win2000/invitedtalks/lucovsky_html/>

[169] **McGraw, G.** "Building secure software: how to avoid security problems the right way". Citado por: David A. Wheeler en http://www.dwheeler.com/sloc/

[170] **McKusick, M. K.** (1999). "Twenty years of Berkeley Unix. From AT&T owned to freely redistributable". En: DiBona *et al.* [108].

http://www.oreilly.com/catalog/opensources/

[171] **SUN Microsystems** (2000). "Sun microsystems announces availability of StarOffice™ source code on OpenOffice.org".

http://www.collab.net/news/press/2000/openoffice_live.html

[172] **Mockus, A.; Fielding, R. T.; Herbsleb, J. D.** (2000, junio). "A case study of open source software development: the Apache server". En: *Proceedings of the 22nd International Conference on Software Engineering (ICSE 2000)*, páginas 263–272. Limerick, Irlanda. ACM Press.

[173] **Molenaar, B.** "What is the context of charityware?".

http://www.moolenaar.net/Charityware.html

[174] **MT Open Course Ware.**

http://ocw.mit.edu

[175] **Nagel, L. W.** (1996, septiembre). "The life of SPICE". En: *1996 Bipolar Circuits and Technology Meeting*. Minneapolis, MN, EE.UU.

http://www.icsl.ucla.edu/aagroup/Life%20of%20SPICE.html

[176] **Narduzzo, A.; Rossi, A.** (2003, mayo). "Modularity in action: GNU/Linux and free/ open

source software development model unleashed".

http://opensource.mit.edu/papers/narduzzorossi.pdf

[177] **Newman, N.** (1999). "The origins and future of open source software".

http://www.netaction.org/opensrc/future/

[178] **Nupedia.**

http://www.nupedia.com

[179] **Villanueva Núñez, E.** (2002, abril). "Carta a Microsoft Perú".

http://www.gnu.org.pe/rescon.html

[180] **Danish Board of Technology** (2002, octubre). "Open-source software in e-Government, analysis and recommendations drawn up by a working group under the danish board of technology. Technical report".

[181] **Open Source Initiative.** "Licencias de fuente abierta".

http://www.opensource.org/licenses/index.html

[182] **Pareto, W.** (1896). "Course of Political Economy". Lausanne.

[183] **Perens, P.; The Open Source Initiative** (1998). "The open source definition". http://www.opensource.org/docs/definition_plain.html

[184] **GNU Perú.** "Proyectos ley de software libre en la Administración pública del Gobierno peruano, Congreso de la República".

http://www.gnu.org.pe/proleyap.html

[185] **Pinheiro, P.** (1999, diciembre). Proposição pl-2269/1999: Dispõe sobre a utilização de programas abertos pelos entes de direito público e de direito privado sob controle acionário da administração pública. Câmara dos Deputados do Brasil.

http://www.camara.gov.br/Internet/sileg/Prop_Detalhe.asp?id=17879

http://www.fenadados.org.br/software.htm

[186] **Pranevich, J.** (2003). "The wonderful world of Linux 2.6".

http://www.kniggit.net/wwol26.html

[187] **The Debian Project.** "Debian developer map".

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

[188] **Puigcercós Boixassa, J.** (2002). Proposición de Ley de Medidas para la Implantación del Software Libre en la Administración del Estado.

http://www.congreso.es/public_oficiales/L7/CONG/BOCG/B/B_244-01.PDF

[189] **Quittner, J.; Slatalla, M.** (1998). *Speeding the net: the inside story of Netscape and how it challenged Microsoft*. Atlantic Monthly Pr.

[190] **Rasch, C.** "A brief history of free/open source software movement".

http://www.openknowledge.org/writing/open-source/scb/brief-open-source-history.html

[191] **Rasch, C.** (2001, mayo). "The Wall Street performer protocol. Using software completion bonds to fund open source software development". *First Monday*, 6(6).

[192] **Raymond, E. R.** (2001, enero). *The cathedral and the bazaar. Musings on Linux and open source by an accidental revolutionary*. O'Reilly & Associates (http://www.ora.com).

http://catb.org/~esr/writings/cathedral-bazaar/

[193] **Reis, C R.; De Mattos Fortes, R. P.** (2002, febrero). "An overview of the software engineering process and tools in the Mozilla Project".

http://opensource.mit.edu/papers/reismozilla.pdf

[194] **Rideau, F. R.** (2000). "Patents are an economic absurdity".

http://fare.tunes.org/articles/patents.html

[195] **Roberts, L.** (1978, noviembre). "The evolution of packet switching". *Proceedings of the IEEE*, (66).

[196] **Robles, G.; González Barahona, J. M.; Centeno González, J.; Matellán Olivera, V.; Rodero Merino, L.** (2003, mayo). "Studying the evolution of libre software projects using publicly available data". En: *Proceedings of the 3rd Workshop on Open Source Software Engineering at the 25th International Conference on Software Engineering*. Portland, EE.UU.

[197] **Robles, G.; Scheider, H.; Tretkowski, I.; Weber, N.** (2001): "Who is doing it? Knowing more about libre software developers".

http://widi.berlios.de/paper/study.pdf

[198] **Rochkind, M.** (1986, mayo). "Interview with Dick Haight". *Unix Review*.

[199] **Scacchi, W.** (2003). "Understanding open source software evolution. Applying, breaking and rethinking the laws of software evolution".

http://www.ics.uci.edu/~wscacchi/Papers/New/Understanding-OSS-Evolution.pdf

[200] **Schneier, B.** (2000). "Software complexity and security".

http://www.counterpane.com/crypto-gram-0003.html

[201] **Smoogen, S. J.** "The truth behind Red Hat names".

http://www.smoogespace.com/documents/behind_the_names.html

[202] **Haggen So.** "Comparison of free/open source hosting (FOSPhost) sites available for hosting projects externally from project owners".

http://www.ibiblio.org/fosphost/exhost.htm

[203] **Stallman, R.** "GNU coding standards".

http://www.gnu.org/prep/standards.html

[204] **Stallman, R.** "Why *free software* is better than *open source*".

http://www.fsf.org/philosophy/free-software-for-freedom.html

[205] **Stallman, R.** (1998). "Copyleft: pragmatic idealism".

http://www.gnu.org/philosophy/pragmatic.html

[206] **Stallman, R.** (1998). "Why *free software* is better than *open source*".

http://www.gnu.org/philosophy/free-software-for-freedom.html

[207] **Stallman, R.** (1998). "Why software should not have owners".

http://www.gnu.org/philosophy/why-free.html

[208] **Stallman, R.** "The GNU Project". En: DiBona *et al.* [108].

http://www.fsf.org/gnu/thegnuproject.html

[209] **Stallman, R.** (1999, junio). "On free hardware". *Linux Today*.

http://features.linuxtoday.com/news_story.php3?ltsn=1999-06-22-005-05-NW-LF

[210] **Stallman, R.** (2001). "The free universal encyclopedia and learning resource".

http://www.gnu.org/encyclopedia/free-encyclopedia.html

[211] **Stallman, R.** (2002). *Free software, free society. Selected essays of Richard M. Stallman*. Joshua Gay.

[212] **Stallman, R.** (2003). "Some confusing or loaded words and phrases that are worth avoiding".

http://www.gnu.org/philosophy/words-to-avoid.html

[213] **Stoltz, M.** (1999). "The case for government promotion of open source software".

http://www.netaction.org/opensrc/oss-report.html

[214] **Tanenbaum, A.; Torvalds, L.** (1999). "The Tanenbaum-Torvalds debate".

http://www.oreilly.com/catalog/opensources/book/appa.html

[215] **The Open Source Initiative.** "The open source definition".

http://www.opensource.org/docs/definition_plain.html

[216] **Tiemann, M.** "Future of Cygnus Solutions. An entrepreneur's account". En: DiBona *et al*. [108].

http://www.oreilly.com/catalog/opensources/book/tiemans.html

[217] **Torvalds, L; Diamond; D.** (2001). *Just for fun: the story of an accidental revolutionary*. Texere.

[218] **Linus Torvalds, Hamano, J. C.; Ericsson, A.** "Git manual page".

http://www.kernel.org/pub/software/scm/git/docs/

[219] **Tuomi, I.** (2002). "Evolution of the Linux credits file: methodological challenges and reference data for open source research".

http://www.jrc.es/~tuomiil/articles/EvolutionOfTheLinuxCreditsFile.pdf

[220] **Varios autores.** "Carta abierta al director de la WIPO".

http://www.cptech.org/ip/wipo/kamil-idris-7july2003.pdf

[221] **Vigo i Sallent, P.; Benach i Pascual, E.; Huguet i Biosca; J.** (2002, mayo). Proposició de llei de programari lliure en el marc de l'Administració pública de Catalunya.

http://www.parlament-cat.es/pdf/06b296.pdf

http://www.hispalinux.es/
modules.php?op=modload&name=Sections&file=index&req=viewarticle&artid=49

[222] **Villanueva Núñez, E.** (2001, diciembre). Proyecto de ley software libre, número 1609.

http://www.gnu.org.pe/proley1.html

[223] **Villanueva Núñez, E.; Rodrich Ackerman, J.** (2002, abril). Proyecto de ley de uso de software libre en la Administración pública, número 2485.

http://www.gnu.org.pe/proley4.html

[224] **W3C** (2000). *Extensible markup language (xml) 1.0* (2ª edición).

[225] **Walsh, N.; Muellner, L.; Stayton, B.** (2002). *DocBook: the definitive guide*. O'Reilly.
http://docbook.org/tdg/en/html/docbook.html

[226] **Welke, L; Johnson, L.** (1998). How the ICP Directory began.

http://www.softwarehistory.org/history/Welke1.html

[227] **Wheeler, D. A.** (2000, julio). "Estimating Linux's size".

http://www.dwheeler.com/sloc

[228] **Wheeler, D. A.** (2001, junio). "More than a gigabuck: estimating GNU/Linux's".

http://www.dwheeler.com/sloc

[229] **Wiesstein, E.** "Concise encyclopedia of mathematics".

http://mathworld.wolfram.com/

[230] **Wikipedia.** "Gini coefficient".

http://www.wikipedia.org/wiki/Gini_coefficient

[231] **Wikipedia.** "Lorenz curve".

http://www.wikipedia.org/wiki/Lorenz_curve

[232] **Wikipedia.** "Pareto".

http://www.wikipedia.org/wiki/Pareto

[233] **Wikipedia.** "TeX".

http://www.wikipedia.org/wiki/TeX

[234] **Wilson, B.** "Netscape Navigator".

http://www.blooberry.com/indexdot/history/netscape.htm

[235] **Computer World** (2000). "Salary survey 2000".

http://www.computerworld.com/cwi/careers/surveysandreports

[236] **Young, R**. (1999). "Giving it away. how Red Hat software stumbled across a new economic model and helped improve an industry".

http://www.oreilly.com/catalog/opensources/book/young.html

[237] **Zawinsky, J. W.** (1999). "Resignation and postmortem".

http://www.jwz.org/gruntle/nomo.html