Wikilibro

Software Libre

Jesús M. González Barahona/ Gregorio Robles y Joaquín Seoane

Enero 2012

Este documento es una compilación del Wikilibro de Software Libre (versión de Enero 2012). Para una mayor actualización, se recomienda consultar el wiki de EOI en http//www.eoi.es/wiki.

Índice

Capítulo 1. ¿Qué es el software libre?

Resumen

En este capítulo vamos a empezar a explicar qué es el software libre, y cuáles son las consecuencias principales que está provocando, como toma de contacto con él. El concepto fue definido por Richard Stallman basándose en cuatro libertades: de ejecución, de estudio, de redistribución y 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 comentan también otros términos relacionados por similitud o contraposición, y que permiten aclarar los conceptos, y las consecuencias de la libertad del software para los principales actores implicados.

Sección 1.El concepto de libertad en el software

Desde el principio de los años 70 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 se lo preste 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 corregir sus errores, sin el permiso explícito del productor, que normalmente se reserva en exclusiva estas posibilidades. Pero esta 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.

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

Definición

El término software libre (o programas libres), tal como fue concebido por Richard Stallman en su definición fsf:definition, hace referencia a las libertades que puede ejercer quien lo recibe. En concreto, hace referencia a cuatro libertades:

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

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

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

Libertad para mejorar el programa y publicar las mejoras. También exige el código fuente.

El mecanismo que se utiliza para garantizarlas, de acuerdo con la legalidad vigente, es la distribución mediante una cierta licencia, 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 restricciones que pueda querer aplicar (como dar crédito a los autores originales en caso de redistribución). Para que la licencia sea considerada como 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, el término inglés 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, hay modelos de negocio basados precisamente en vender software, porque son muchas las circunstancias donde el consumidor está dispuesto a pagar si recibe ciertas contraprestaciones, como por ejemplo una cierta garantía, aunque sea subjetiva, sobre el software que recibe, 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 ellas, destacan por su importancia histórica la definición de software libre de la Free Software

Foundation fsf:definition, las directrices de Debian para decidir si un programa es libre debian:freesftwareguidelines y la definición de la Open Source Initiative del término "open source" osi:definition, muy similar a las anteriores.

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

Términos relacionados

Equivalente a software libre es el término Open Source Software (programas de fuente abierto), 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 Debianosd:open-source-definition:98. 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 Stallmanfsf:freeforfreedom y la Free Software Foundationfsf, 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 alguna manera con el software libre son:

Freeware. Programas gratuitos. Normalmente se distribuyen sólo en binario, y se pueden obtener sin coste. A veces se obtiene también permiso de redistribución, pero otras no, pudiendo obtenerse entonces sólo del sitio "oficial" mantenido a ese efecto. Es habitual que estos programas 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 fuentes, se pueden copiar libremente, pero no usar continuadamente sin pagarlos. La exigencia de pago puede estar incentivada por funcionalidad limitada o mensajes molestos, o una simple apelación a la moral del usuario, además de que las estipulaciones legales de la licencia podrían utilizarse en contra del infractor.

Charityware, Careware. Generalmente shareware, pero cuyo pago se exige 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 molenaar:charityware.

Dominio público. El autor renuncia absolutamente a todos sus derechos, en favor del común, lo cual tiene que estar declarado explícitamente 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 fuentes, el programa es libre.

Copyleft. Un caso particular de software libre cuya licencia obliga a que las modificaciones que se distribuyan sean también libres.

Privativo, propietario, Cerrado, No libre. Términos usados para denominar al software que no es libre ni de fuente abierta.

Sección 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, heredera de la cultura hacker, y partidaria del apelativo libre, que argumenta que el software es conocimiento que debe poder difundirse sin trabas, y que su ocultación es una actitud antisocial y que la posibilidad de modificar programas es una forma de libertad de expresión. Puede profundizarse en este aspecto en los ensayos de Stallman o en el análisis de Pekka Himanen

La motivación pragmática, abanderada por la Open Source Initiative y partidaria del apelativo fuente abierta, que argumenta ventajas técnicas y económicas, que repasaremos en la sección 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 a 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.

Sección 3. Consecuencias de la libertad del software

El software libre trae consigo numerosas ventajas y pocas desventajas, muchas de ellas

exageradas (o falseadas) por la competencia propietaria. De ellas la que más fundamento tiene es la económica, ya que como vimos no es posible obtener mucho dinero de la distribución y ésta la puede y suele hacer alguien distinto al autor. Es por ello que se necesitan modelos de negocio y otros mecanismos de financiación, que se desarrollan en el Capítulo 5. Otras, 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 a través de foros de usuarios y desarrolladores, y muchas veces tiene gran calidad.

Teniendo presentes los problemas económicos, hemos de observar que el modelo de costes del software libre es muy distinto del 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 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 ser 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 vía Internet y con propaganda gratuita en foros públicos destinados a ello.

Otra consecuencia de las libertades es la calidad que se deriva de la colaboración voluntaria de gente que contribuye o que descubre y reporta 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 mejor calidad.

Examinemos ahora las consecuencias beneficiosas según el destinatario.

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 disponiendo de fuente y de conocimientos, 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 fuentes. Nos olvidamos además 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 adoptarlos ahora es mucho más sencilla, ya que basta 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 es siempre 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 repare, muchas veces se hará en la versión siguiente, que podría tardar años en salir, y a veces además será necesario comprarla de nuevo. En software libre, sin embargo, lo podemos hacer nosotros, 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.

Para la administración pública

La administración pública es un gran usuario de características especiales, ya que tiene obligaciones especiales con el ciudadano, ya sea proporcionándole servicios accesibles, neutrales respecto a los fabricantes, ya garantizando la integridad, utilidad, privacidad y 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 que no es 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 la hace tener un gran impacto, que debería dirigirse a la creación de un tejido tecnológico generador de riqueza nacional. Ésta puede crearse fomentando empresas cuyo negocio sea, en parte, el desarrollo de nuevo software libre para la administración, o el mantenimiento, adaptación o auditoría del software existente. En el Capítulo 6 nos extendemos más en esta

cuestión.

Para el desarrollador

Para el desarrollador y productor de software, la libertad cambia mucho las reglas del juego. Con él le es más fácil competir siendo pequeño y adquirir tecnología punta. Puede aprovecharse del trabajo de los demás, compitiendo incluso con otro producto modificando 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 con detalle este tema.

Para el integrador

Para el integrador el software libre es el paraíso. No 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, disponiendo de un acervo ingente de software libre de donde extraer las piezas.

Para el que proporciona mantenimiento y servicios

El disponer de fuente lo cambia todo, situándonos 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.

Capítulo 2. Un poco de historia

Resumen

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 M.I.T. en 1971, pasé a formar parte de una comunidad de software compartido que había existido durante muchos años. El compartir código no era algo específico de nuestra comunidad: es algo tan antiguo como los ordenadores, como el compartir recetas es tan viejo como el 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 transportar 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")

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 completamente, y el software privativo dominó la escena, prácticamente en exclusiva, durante bastante tiempo. Fue durante época cuando se sentaron las bases del software libre como lo entendemos hoy día, y cuando, poco a poco, comenzaron 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 donde el software libre es una posibilidad a considerar en casi todos los ámbitos.

Esta historia es bastante desconocida, hasta el punto de 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 fueron plantadas ya a principios de los años 1980.

Sección 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 hasta bastantes años antes.

Y en el principio fue libre...

Durante los años 1960 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, y hasta cierto punto organizaban, estos intercambios. La sección 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 las modificaciones.

El 30 de junio de 1969 IBM anunció que, a partir de 1970, iba a vender parte de su software por separadograd02:_person_recol. Esto 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 que tenían los usuarios para compartir, modificar o estudiar el software. En otras palabras, se pasó a la situación que sigue siendo habitual en el mundo del software a principios del siglo XXI.

A mediados de la década de 1970 era ya absolutamente habitual, en cualquier ámbito informático, encontrarse con software privativo. Esto supuso un gran cambio cultural entre los profesionales que trabajaban con software, y fue el origen del florecimiento de un gran número de empresas alrededor del 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.

Años 1970 y primeros 1980

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 según lo definimos hoy día. Entre ellas cabe destacar SPICE, TeX y el caso mucho más complejo de Unix.

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 de diseño de circuitos integrados. Estando 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 se ha hecho durante su historia docenas de veces por 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 fue este 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).

Sugerencia: Puede consultarse más información sobre la historia de SPICE en nagel96:_life_spice. 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 un de los sistemas más grandes y más conocidos que podía considerarse software libre.

Sugerencia: Algunos hitos de la historia de TeX pueden consultarse en línea en histor_tex, y más detalles sobre TeX, en el artículo correspondiente de la Wikipedia wikipedia.

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 distribución libre, 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 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 fue muy importante, como veremos más adelante, 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 hauben-hauben: notha:97, 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 referencia: "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 fuentes hacía posible la identificación y 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 siguiente texto de rochkind:iwdh:86: "Ésta era una de las grandes cosas sobe 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 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. Durante el final 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 hizo tan popular a Unix entre los desarrolladores, cambió radicalmente, hasta el punto de que en 1991, AT&T puso una demanda a la Universidad de Berkeley por publicar el código de Unix BSD que ellos (el CSRG de Berkeley) había creado. Pero esa es otra historia, que retomaremos más adelante.

Sección 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 y hasta económicos, que se han ido desarrollando y completando hasta hoy día. Y como el nuevo fenómeno necesitaba un nombre, durante esos años se acuñó también el propio término software libre.

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 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 software completo, de propósito general, pero completamente libre stallman:gnu-project:99. 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 estaba preocupado 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 derechos (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 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 foundation91:_gnu_gener_public_licen.

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 como The GNU Manifesto stallman:gnu-manifesto:85. y Why Software Should Not Have Owners stallman:why-software-not-owners:98.

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 (habitualmente voluntarios) que desarrollaban alguna de las herramientas que luego encajarían perfectamente en el rompecabezas completo (el sistema GNU). La modularidad de Unix, en el que se inspiraba el desarrollo, encajaba perfectamente en esta idea. El método de trabajo generalmente implicaba el uso de Internet, pero ante la escasa implantación en aquellos días, la Free Software Foundation también vendía cintas en las que grababa las aplicaciones, siendo probablemente una de las primeras organizaciones en beneficiarse 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. Aún 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 aquella época 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.

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 hicieron importantes mejoras al núcleo y se le añadió mucha funcionalidad. Por ejemplo, durante la década de los años 1980, 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 a 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 como un ejemplo de licencia libre minimalista (que además de permitir la redistribución libre, también permitía 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 (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 el escepticismo de algunos miembros del CSRG, realizó un anuncio público pidiendo ayuda para realizar esta tarea, y poco a poco las utilidades (reescritas a partir de especificaciones) comenzaron a llegar a Berkeley. Mientras tanto, el

mismo proceso se realizó con el núcleo, reescribiéndose de forma independiente la mayor parte del código que no se había realizado por Berkeley o por 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 se distribuyó el conjunto bajo la licencia BSD, vendiéndose 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 aportando en la Red para mejorar 386BSD. Más adelante apareció FreeBSD como un intento en 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), realizada de forma independiente por 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 les acusaba de distribuir su propiedad intelectual sin permiso. Después de varias maniobras judiciales (incluyendo una contra-demanda 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 fue pronto utilizado 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 era 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.

Sugerencia: La historia de Unix BSD es muy ilustrativa de una forma peculiar de desarrollar software durante los años 1970 y 1980. Quien esté interesado en ella puede disfrutar de la lectura de mckusick:_twent_years_berkel_unix. En él 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.

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 tuvieron 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 1980 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 (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, la comunidad del software libre, durante el final de los años 1980 y la década de 1990, fue una de las primeras en explorar hasta el fondo las nuevas posibilidades que permitía Internet para la colaboración entre grupos geográficamente dispersos. Esta exploración fue la que 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 el WWW Consortium), en su época la libre disposición de todas las especificaciones y documentos de diseño, incluyendo las normas que definen los protocolos, fue algo revolucionario y fundamental para su desarrollo. Podemos leer en hauben-hauben:notha:97 (página 106):

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.

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

En otra cita, roberts:teops:78 (página 267) podemos leer:

Como ARPANET era un proyecto público que conectaba a 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 hasta que Internet, ya bien entrados los años 1990, se convirtiese sobre todo en un negocio, la comunidad de usuarios y su relación con los desarrolladores era clave. En esa 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 innovadores eran fruto no de grandes planes de investigación en empresas, sino de estudiantes o profesionales probando ideas y recogiendo la realimentación que les enviaban cientos de usuarios que usaban sus programas libres.

Como ya se ha dicho, Internet proporcionó al software libre también las herramientas básicas para colaborar a distancia. El correo electrónico, los grupos de News, 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 y 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.

Sugerencia: El lector interesado en una historia de la evolución de Internet, escrita por varios de sus protagonistas, puede consultar barry97:_brief_histor_inter.

Otros proyectos

Durante la década de 1980 vieron la luz otros importantes proyectos libres. Entre ellos destaca, por su importancia y proyección futura, el sistema X Window (sistema de ventanas para sistemas tipo Unix), desarrollado en el MIT, que fue uno de los primeros ejemplos de financiación en 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 1980 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, estaban sentando las bases de los primeros sistemas libres completos que aparecieron a principios de la década de 1990.

Sección 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 por otro 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 en muchos 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.

En busca de un núcleo

A finales de la década de los ochenta, principios de los noventa, el proyecto GNU contaba con una gama básica de utilidades y herramientas que permitían tener el sistema operativo al completo. Ya entonces muchas aplicaciones libres eran las mejores en su campo (utilidades Unix, compiladores), siendo especialmente interesante el caso de X Window. Sin embargo, para terminar 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.

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 6 ficheros para tenerlo (el resto ya había sido construido por el CSRG o sus colaboradores). A primeros de 1992 Bill Jolitz completa esos

ficheros y distribuye 386BSD, un sistema que funciona sobre arquitectura i386, y que con el tiempo dará lugar a los proyectos NetBSD, FreeBSD y OpenBSD. El desarrollo durante los meses siguientes es rápido, y a finales de año ya es suficientemente estable como para ser usado en producción en entornos no críticos, incluyendo ya, 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 hay componentes que usaban otras licencias (como los procedentes del proyecto GNU, que usaban la GPL), la mayor parte del sistema se distribuye bajo la licencia BSD.

Sugerencia: Algunos de los episodios de esta época son ilustrativos de la potencia de los modelos de desarrollo de software libre. El caso de Linus Torvalds, desarrollando Linux mientras era estudiante de segundo curso de la Universidad de Helsinki es bien conocido. Pero no es el único caso de un estudiante que se abrió camino gracias a sus desarrollos libres. Por ejemplo, Thomas Roel, un estudiante alemán, portó X11R4 (una versión del sistema X Window) a un PC basado en un 386. Este desarrollo le 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 [hammel91:_histor_xfree8].

Luego vino la demanda de USL, que hizo que muchos potenciales usuarios 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.

GNU/Linux entra en escena

En julio de 1991 Linus Torvalds (estudiante finés de 21 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 aparecieron nuevas versiones. En marzo de 1994 apareció la versión 1.0, la primera que fue denominada estable, pero el núcleo que había construido Linus era usable desde bastantes meses antes. Durante este periodo, 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, Linux (el núcleo) y gran parte de los componentes que se integran alrededor de él se distribuyen con la licencia GPL.

Sugerencia: La historia de Linux es probablemente una de las más interesantes (y conocidas) en

el mundo del software libre. Quien esté interesado en ella puede mirar en [hasan:_histor_linux]. Como curiosidad, puede consultarse el hilo en que Linus Torvalds anunciaba que estaba empezando a crear lo que luego fue Linux (en el grupo de News comp.os.minix) en http://groups.google.com/groups?th=d161e94858c4c0b9.

En él 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 aparecieron tan pronto como 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 durante 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 basándose todas en el 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 al selección, instalación, sustitución y desinstalación de estos paquetes, la instalación inicial en un ordenador, y la gestión y administración del sistema operativo.

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

Debian, desarrollada por una comunidad de desarrolladores voluntarios.

Red Hat Linux, primero desarrollada internamente por la empresa Red Hat, pero adoptando más adelante un modelo más comunitario, dando lugar a Fedora Core.

Suse, que dio lugar a OpenSuSE, en una evolución similar a la de Red Hat.

Mandriva (sucesor de Mandrake Linux y de Conectiva).

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

Sección 4. Tiempos de maduración

A mediados de la década de los 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 están empezando 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, se está empezando a plantear 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 llegue a ser más que una moda pasajera, que con el tiempo sólo será recordada con nostalgia. Pero también podría ser (y cada vez parece más que lo es) un nuevo modelo que está aquí para quedarse, y quizás para cambiar radicalmente una de las industrias más jóvenes, pero también de las más influyentes.

Finales de los 1990

A mediados de la década de 1990 el software libre ofrece ya entornos completos (distribuciones de GNU/Linux, sistemas *BSD) que permiten el trabajo diario de mucha gente, sobre todo de desarrolladores de software. Aún hay muchas asignaturas pendientes (la mayor de ellas, el disponer de mejores interfaces gráficas de usuario, en una época donde Windows 95 es considerado el estándar), pero ya hay unos cuantos miles de personas, en todo el mundo, que sólo usan software libre en su trabajo diario. Los anuncios de nuevos proyectos se suceden y el software libre comienza 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 [raymond:cathedral-bazaar]. Aunque mucho de lo expuesto en él era ya bien conocido por la comunidad de desarrolladores de software libre, el 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 [hecker:setting-shop-business:98], de Frank Hecker, que por primera vez expuso los modelos de negocio posibles alrededor del 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 Navigator 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 CDs con sistemas GNU/Linux listos para usar podía ser un modelo de negocio. Red Hat comenzó distribuyendo su Red Hat Linux, con 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, Red Hat 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 consiguió 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 fue más tarde adquirida por esta última).

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

En esta época nacen también muchas empresas basadas en modelos de negocio alrededor del software libre. Sin salir a bolsa y no lograr tan estupendas capitalizaciones, han sido sin embargo muy importantes para el desarrollo del software libre. Por ejemplo, aparecieron muchas otras empresas que empezaron distribuyendo sus propias versiones de GNU/Linux, como SuSE (Alemania), Conectiva (Brasil) o Mandrake (Francia, que más tarde se unió a la anterior para formar Mandriva). Otras proporcionan servicios a empresas que ya demandan mantenimiento y adaptación de productos libres: LinuxCare (EE.UU.), Alcove (Francia), ID Pro (Alemania). Y muchas más.

Por su lado, los gigantes del sector también empiezan a posicionarse ante el software libre. Algunas empresas, como IBM, lo incorporan directamente en su estrategia. Otras, como Sun Microsystems, mantienen 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.) exploran el modelo del software libre con diversas estrategias, que van desde la liberación selectiva de software hasta el simple porte a Linux de sus productos. Entre estos dos extremos, se observan otras muchas líneas de acción, como la utilización más o menos intensa de software libre en sus productos (como es el caso del MacOS 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 es, probablemente, la aparición de dos ambiciosos proyectos con el objetivo de conseguir llevar el software libre al entorno de escritorio (desktop) de los usuarios no muy versados en la informática: KDE y GNOME. El objetivo final era, dicho de forma muy simplista, 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 Troll Tech, pero gratuito para su uso sobre Linux[2]), iniciaron la construcción de un conjunto de aplicaciones de escritorio que funcionasen de forma

integrada, y tuvieran 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 escritorios que sus usuarios podían elegir).

En gran medida como reacción a la dependencia que tenía KDE de la biblioteca propietaria Qt, en agosto de 1997 se anuncia el proyecto GNOME [icaza:_story_gnome], con objetivos y características muy similares a las de KDE, pero con el objetivo explícito de que todos sus componentes sean libres. En marzo de 1999 se liberó GNOME 1.0, que también se iría, con el tiempo, 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 sistemas de ventanas para sistemas 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 [stallman:why-free-software-better:98]. 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. La OSI utilizó para definir el software open source una definición derivada de la que utiliza el proyecto Debian para definir qué es software libre [debian:freesftwareguidelines] (que por otra parte refleja con bastante aproximación la idea de la FSF al respecto [fsf:definition]), 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 del software libre y del software de fuente abierta (o al menos las personas que se identifican como parte de una o de otra) pueden ser profundamente diferentes.

Década de los 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 ya listo para el escritorio. Sistemas como GNOME, KDE, OpenOffice.org y Firefox pueden ser utilizados por usuarios domésticos, y son suficientes para las necesidades de muchas empresas, 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 a empresas como Oracle, que reaccionan simplemente portando sus productos a GNU/Linux. En el otro podría situarse a IBM, que tiene la estrategia más decidida y ha realizado las mayores campañas de publicidad sobre 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 la comunidad, el crecimiento es enorme. Cada vez hay más desarrolladores, más proyectos de software libre activos, más usuarios... Cada vez más el software libre está pasando de ser algo marginal para convertirse en un competidor a 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 en sus diferentes aspectos: modelos de desarrollo, de negocio, mecanismos de coordinación, gestión de proyectos libres, motivación de desarrolladores, etc.

En estos años comienzan también a verse los primeros efectos de la deslocalización que permite el desarrollo de software libre: países considerados como 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 traducción de industria software) en proyectos como GNOME [lancashire:code-culture-cash]. Y es aún más interesante el papel que está teniendo Brasil, con 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, es de destacar cómo hay ciertos mercados (como los servicios de Internet o la ofimática) donde el software libre se ha convertido en una opción natural y es difícil de justificar no considerarla 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 un lado, las patentes de software (patentes de programación) están siendo consideradas cada vez en más países. Por otro, las nuevas leyes de protección de derechos de autor están dificultando o haciendo imposible el desarrollo de aplicaciones libres en algunos ámbitos, siendo el más conocido el de los visores de DVDs (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 va 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 nivel 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 medios de comunicación (es la primera vez que Extremadura ha sido portada del 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. Alrededor de 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 en esta enseñanza. Hay ordenadores que se venden con gnuLinEx preinstalado. En general, se está tratando de crear alrededor de esta experiencia todo un tejido docente y empresarial que le proporcione soporte. Y la experiencia se ha exportado. A principio del siglo XXI son varias las comunidades autónomas en España que han apostado por el software libre (de una u otra forma) para la enseñanza, y en general su importancia para las administraciones públicas es ampliamente reconocida.

Knoppix

Desde finales de los años 1990 hay distribuciones de GNU/Linux que se instalan fácilmente, pero probablemente Knoppix, cuya primera versión apareció durante 2002, ha llevado este concepto a su máxima expresión. 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 MS 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 MS-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 mercado de hasta el 80%. La situación empezó a cambiar cuando Microsoft incluyó Internet Explorer con su Windows 95, lo que supuso que poco a poco fuera perdiendo cuota de mercado. A principios de 1998 Netscape anunció que iba a distribuir gran parte del código de su Navigator como software libre, cosa que efectivamente hizo en marzo del mismo año, lanzando el proyecto Mozilla. Durante bastante tiempo estuvo rodeado de incertidumbre, e incluso pesimismo (por ejemplo cuando el líder del proyecto, 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 como 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 del Navigator.

Sugerencia: En [wilson:_netsc_navig] 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, teniendo en cuenta las cuotas de mercado que tuvo Internet Explorer durante 2002 ó 2003 (fue líder indiscutible relegando a Mozilla y 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 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, produciendo una buena cantidad de navegadores. Por otro lado, la combinación Firefox más OpenOffice.org permite usar software libre para las tareas más cotidianas incluso en un entorno MS-Windows (ambos funcionan no sólo sobre GNU/Linux, *BSD y

otros sistemas 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 libre en entornos de oficina sea simple: se puede empezar usando 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 pasando a GNU/Linux o 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 es compleja, está centrada en la acusación de que IBM contribuyó con código que era de SCO al núcleo de Linux. En mayo de 2007 el asunto aún 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), campañas de SCO amenazando 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 casos relacionados con la validez de la GPL que se han resuelto en Alemania durante 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 dedicarle 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 (un caso bien conocido es gnuLinEx, también basado ella), pero quizás sí ha sido, entre todas ellas, la que más recursos ha recibido. Por ejemplo, Canonical ha contratado a una gran cantidad de expertos en Debian (muchos de ellos participantes 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 lado, 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 la escena, son muchos los grupos y las empresas que 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 ha 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 son varias las que de forma regular convocan concursos públicos para la creación y mantenimiento de las nuevas versiones de sus distribuciones.

La creación de distribuciones particularizadas hace real una tendencia de la que se hablaba desde hace tiempo en el mundo del software libre: la adaptación de los programas a las

necesidades específicas de los usuarios, sin necesidad de que los productores originales tengan necesariamente que realizar esta adaptación.

Sugerencia: Algunas de las distribuciones autonómicas de GNU/Linux más conocidas son las siguientes: gnuLinEx, http://linex.org (Extremadura)

Guadalinex, http://guadalinex.org (Andalucía)

Lliurex, http://lliurex.net (Valencia)

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, de voluntarios con empresas

Desde prácticamente el principio del software libre ha habido empresas colaborando con voluntarios en el desarrollo de aplicaciones. Sin embargo, durante estos años donde 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 en tratar de que sean los voluntarios los que vengan a colaborar a sus propios proyectos libres. Ejemplos de la primera situación son la GNOME Foundation o el ya mencionado de 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 mostrado que en el campo de la producción de programas, es posible otra forma de hacer las cosas. Ha sido posible ver en la práctica cómo otorgando las libertades de distribución, modificación y uso es posible conseguir la sostenibilidad, bien mediante trabajo

voluntario o incluso mediante generación de negocio, que permita la supervivencia de 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. La Wikipedia está mostrando que en un área tan particular como la producción de enciclopedias puede recorrerse un camino muy interesante. Y cada vez son más los autores literarios, grupos musicales, incluso productoras de películas interesadas en modelos libres de producción y distribución.

Queda mucho camino por andar en todos estos dominios, 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 The Cathedral and the Bazaar empezaron a abrir el camino del estudio del software libre como tal, no fue hasta los años 2001 y siguientes cuando la comunidad académica comenzó a considerar al 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 los 2000 son ya varios los 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.

Sección 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, trataremos de 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, sin duda, su situación dentro de unos años.

Técnica FUD (fear, uncertainity, doubt o, en español, miedo, desconocimiento, duda). Son técnicas bastante habituales en el mundo de las tecnologías de la información, y que han sido utilizadas 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 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 tratando 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 generan en los clientes y 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 lo consideran de moda. Si no profundizan más allá, y 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 y organización que se aproxime al software libre.

Impedimentos legales. Sin duda este 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 suficiente espacio 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.

Capítulo 3. Licencias en Software libre

Resumen

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, version 2

Legalmente hablando, la situación de los programas libres respecto de los privativos no es muy diferente: también se distribuyen bajo licencia. Lo que que les diferencia es precisamente qué 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 a 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 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 recibe o se compra. 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 recibes un programa libre puedes redistribuirlo o no, pero si lo redistribuyes, sólo puedes hacerlo porque la licencia te 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 su obra lo que le apetezca, incluso distribuirla con términos y condiciones totalmente diferentes (o sea, 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 apeteciere, otra posterior bajo una licencia propietaria. En caso de existir más autores, y que la nueva versión contenga código cuya autoría les corresponda y que se vaya a publicar bajo otras condiciones, todos ellos han 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 la licencia del proyecto, 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 de 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, es preciso 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 que se presentaron en “Definición”.

Sección 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 otra, la mayoría de los usuarios prefieren referirse a una licencia ampliamente conocida que leerse y analizar licencias completas.

Sugerencia: Puede 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 [fsf:licences]. El punto de vista filosóficamente diferente de la Open Source Initiative se refleja en su listado[osi:licenses]. 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), notificar a Apple de las redistribuciones, o por la posibilidad de revocación unilateral. No obstante, la presión de esta clasificación ha hecho que Apple publique 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. Una de ellas 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 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 otra 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, ya que le permite hacer casi lo que quiera con él (en términos de condiciones de futuras redistribuciones), el segundo lo hace en la libertad de cualquiera que potencialmente pueda recibir algún día un trabajo derivado del programa (ya que obliga a que las sucesivas modificaciones y redistribuciones respeten los términos de la licencia original).

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.

Nota: El término copyleft aplicado a una licencia, usado sobre todo por la Free Software Foundation para definir sus licencias, tiene implicaciones similares a las del término “licencia robusta”, tal y como lo usamos en este texto.

Sección 2. Licencias permisivas

Las licencias permisivas, a veces también llamadas liberales o minimalistas, no imponen prácticamente ninguna condición sobre quien recibe el software, y sin embargo le dan permiso de uso, redistribución y modificación. Este enfoque, desde un punto de vista, 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 de que una vez recibido el programa por alguien, se sigan garantizando las mismas libertades cuando ese programa se redistribuye. De hecho, estas licencias típicamente 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 el punto que en muchas ocasiones se refieren las licencias permisivas como licencias 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 y la de los fuentes, aunque no obliga a ninguna de las dos en ningún caso. Asimismo se da permiso para realizar modificaciones y ser integrada con otros programas casi sin restricciones.

Nota: Una de las consecuencias prácticas de las licencias 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 licencia. 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 sockets, ya que 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: XWindow, 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 había sido pagado 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 pueda 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, ven en ella la máxima expresión de la libertad y argumentan que, a fin de cuentas, se puede hacer (casi) lo que se quiera con el software.

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

Las redistribuciones en fuente deben retener la nota de copyright y listar estas condiciones y la limitación de garantía,

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

Ni el nombre del propietario ni 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 (INCLUYENDO PÉRDIDA DE DATOS, DE BENEFICIOS O INTERRUPCIÓN DE NEGOCIO).

Otras licencias permisivas

A continuación se describen brevemente algunas licencias permisivas:

Licencia de X Window versión 11 (X11) [x_window_system]

Es la licencia usada para la distribución del sistema X Window, el sistema de ventanas más ampliamente usado en el mundo Unix, y también en entornos GNU/Linux. Es una licencia muy similar a la licencia BSD, que permite redistribución, uso y modificación prácticamente sin restricciones. A veces, esta licencia es llamada “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 [zope_public_licen]

Esta licencia (habitualmente llamada "ZPL") es usada para la distribución de Zope (un servidor de aplicaciones) y otros productos relacionados. Es una licencia similar a la BSD, con el interesante detalle de prohibir expresamente el uso de marcas registradas por Zope Corporation.

Licencia de Apache

Licencia bajo al 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 (en el dominio público, o del común). La principal consecuencia de esta declaración es que el autor renuncia a todos sus derechos sobre el programa, y por lo tanto puede modificarse, redistribuirse, usarse, etc. de cualquier manera. A efectos prácticos, es muy similar a que el programa esté bajo una licencia tipo BSD.

Sección 3. Licencias robustas

Son muchas las licencias robustas que podemos encontrarnos. A continuación se describen algunas de ellas.

La Licencia Pública General de GNU (GNU GPL)

La Licencia Pública General del proyecto GNU[foundation91:_gnu_gener_public_licen] (más conocida por su acrónimo en inglés GPL), que mostramos traducida en el Apéndice C, Licencia Pública GNU, es con diferencia la licencia 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 a veces se traduce 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 las fuentes. 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 enlazado o linkado) si éste tiene una licencia compatible. Esto ha sido llamado efecto viral (aunque muchos consideran esta denominación como despectiva) de la GPL, ya que código publicado una vez con esas condiciones nunca puede cambiar de condiciones.

Nota: Una licencia es incompatible con la GPL cuando restringe alguno de los derechos que la GPL garantiza, ya sea explícitamente contradiciendo alguna cláusula, ya 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 propaganda 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 ambas.

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 hecho privativo. Es más, ni ese programa ni modificaciones al mismo pueden ser publicadas con una licencia diferente a la propia GPL. Como ya se ha dicho, los partidarios de las licencias tipo BSD ven en esta cláusula un recorte de la libertad, mientras que sus seguidores ven en ello 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 tipo BSD lo hacen para los desarrolladores. Nótese, 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, ya que obliga a sus competidores a publicar sus modificaciones (mejoras, correcciones, etc.) en caso de que redistribuyan su software, mientras que con una licencia tipo BSD éste no tiene por qué ser el caso.

En cuanto a la naturaleza contraria al copyright, esto se debe a que la filosofía que hay detrás de esta licencia (y detrás de la Free Software Foundation) es que el software no debe tener propietarios [stallman:why-software-not-owners:98]. 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 su obra, las condiciones bajo las que publica su obra confieren a la misma tal carácter que podemos considerar que la propiedad del software corresponde a quien

lo tiene y no a quien lo ha creado.

Por supuesto, también incluye negaciones de garantía para proteger a los autores. Asimismo, y para proteger la buena fama de los autores originales, toda modificación de un fichero fuente debe incluir una nota con la fecha y autor de cada modificación.

La GPL contempla también a las patentes de software, exigiendo que si el código lleva algoritmos patentados (como dijimos, 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 para contemplar 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 versión tercera de la GPL [gplv3] pretende actualizarla 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 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 proporciona 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]. La licencia tampoco permite que el software obligue a la ejecución en entorno prefijado, como ocurre cuando se prohibe la utilización de núcleos no firmados en distribuciones cuya política de seguridad así 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 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

hardware y software así diseñadas, pudiendo modificarse el software para su utilización en otro hardware. Con respecto de 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 que no existiesen plataformas hardware o software en las que ejecutar dichas modificaciones del software, quedando en ese caso 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[foundation99:_gnu_lesser_public_licen] (comúnmente conocida por sus iniciales en inglés LGPL) es la otra licencia de la Free Software Foundation. Pensada en sus inicios para su uso en bibliotecas (la L en sus comienzos 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 la integración con cualquier otro software sin prácticamente limitaciones.

Como se puede ver, en un principio, esta licencia estaba orientada a las bibliotecas, de manera 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 la suite de ofimática OpenOffice.org están licenciadas, entre otras, también bajo la LGPL.

Nota: Igual que la GPL, la última versión publicada de la LGPL es la segunda, aunque ya hay borrador de la versión 3 [lgplv3]. Esta nueva versión es más corta que la anterior, dado que refiere todo su texto a GPLv3, destacando únicamente sus diferencias.

Otras licencias robustas

Otras licencias robustas que puede resultar interesante comentar son:

Licencia de Sleepycat [sleepycat-license]

Es la licencia bajo la que la empresa Sleepycat [sleepycat] distribuye sus programas (entre ellos el conocido Berkeley DB). Obliga a ciertas condiciones siempre que se redistribuye el programa o trabajos derivados del programa. 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 [ecos-license]

Es la licencia bajo la que se distribuye eCos [ecos], un sistema operativo de 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 [02:_affer_gener_public_licen]

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 el programa 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 el código fuente. La Affero GPL tiene una cláusula que obliga 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 fuente, cualquier usuario puede obtenerlo, y además esa redistribución está sometida a las condiciones de la licencia. La Free Software Foundation está considerando incluir provisiones similares en la versión 3 de su GNU GPL.

IBM Public License Version 1.0 [ibm_public_licen_version]

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 por obligar 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 [mozil_public_licen]

Ejemplo de licencia libre con origen en una empresa. Es una evolución de la primera licencia libre que tuvo el Netscape Navigator, y en su momento fue muy importante por ser la primera vez que una empresa muy conocida decidió distribuir un programa bajo su propia licencia libre.

Sección 4. Distribución bajo varias licencias

Hasta este punto se ha dado por supuesto que cada programa se distribuye bajo una única licencia en la que se especificaban 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 se difieren en la licencia. Como veremos, en la mayoría de los casos, esto se traduce en que dependiendo 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 GPL podían hacer uso de una versión de Qt GPL, mientras que si se quería 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 día KDE es considerado 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.

Sección 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 se le apliquen las mismas libertades y 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 futura del programa (changelog y TODO), autoría y condiciones de copia (AUTHORS y copyright o COPYING), así como las instrucciones de uso. Todas ellas, menos la autoría y las condiciones de copia, deberían ser libremente modificables según 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 estas mismas lo permiten.

Las instrucciones de uso suelen estar codificadas en formatos más complejos, ya que suelen ser documentos más largos y 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, el formato de páginas de manual de Unix, TexInfo, LaTeX o DocBook, sin perjuicio de distribuir también el resultado de transformar esos documentos fuente en formatos más aptos para visualizar o imprimir, 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 uso de un programa concreto (HOWTOs, COMOs o recetarios), a veces es documentación más amplia, abarcando varios programas y su integración, comparando soluciones, etc., ya sea en forma de tutorial o de referencia. A veces es una mera recopilación de preguntas frecuentes con sus respuestas (FAQs o PUFs). Ejemplo notable es el Proyecto de Documentación Linux [tldp]. En esta categoría podemos también incluir otros documentos

técnicos, no necesariamente sobre programas, ya sean las instrucciones para cablear una red local, construir una cocina solar, reparar un motor o 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 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 manifestado con tanta fuerza en el mundo del software libre como en el de la documentación libre.

Capítulo 4. El desarrollador y sus motivaciones

Resumen

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 si 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”

El desarrollo parcialmente anónimo y distribuido del software libre ha permitido que durante muchos años los recursos humanos con los que cuenta el software libre sean desconocidos. Consecuencia de este desconocimiento ha sido la mistificación al menos parcial del mundo del software libre y de la vida de los que están detrás de él, amparándose 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 a 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 los indirectos son difícilmente cuantificables.

Sección 1. ¿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, siendo los más significativos -aunque, por supuesto, no los únicos- [floss:survey:02] y [widi:survey:01].

Los desarrolladores de software libre son generalmente personas jóvenes. La media de edad está situada en torno a los 27 años. La varianza de la edad es muy grande, ya que el grupo predominante se encuentra en una horquilla que va desde los 21 a los 24 años, siendo la mediana -el valor que aparece con mayor frecuencia- los 23 años. Es interesante observar cómo la edad de incorporación al movimiento de software libre tiene sus máximos entre los 18 y 25 años -siendo especialmente pronunciada entre los 21 y 23 años-, lo que equivaldría a la edad universitaria. Esta evidencia contrasta con la afirmación de que el software libre es cosa principalmente de adolescentes, aunque su presencia es evidente (alrededor de un 20% de los desarrolladores tiene menos de 20 años). En definitiva, podemos ver cómo los desarrolladores suelen ser mayoritariamente veinteañeros (un 60%), mientras que los menores de 20 y los mayores de 30 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. Esto no es de extrañar, ya que como se ha podido ver en el capítulo de historia, el software libre -antes incluso de llevar esta denominación- ha estado íntimamente ligado a las instituciones educativas superiores. Aún hoy, el verdadero motor del uso y expansión del software libre siguen siendo las universidades y los grupos de usuarios estudiantiles. No es, por tanto, de extrañar que más de un 70% de los desarrolladores cuenten 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. Aún así, también tienen cabida -y no por ellos 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%, compitiendo con el propio margen de error de las encuestas. 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.

Sección 2. ¿Qué hacen los desarrolladores?

Profesionalmente los desarrolladores de software libre se definen como ingenieros software (33%), estudiantes (21%), programadores (11%), consultores (10%), profesores de universidad (7%), etc. En el lado contrario, podemos ver cómo 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 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 de 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 al 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, a la composición de los equipos de desarrollo. Es muy difícil encontrar una industria moderna donde el grado de heterogeneidad sea tan grande como el que se puede ver en el software libre, si es que existe.

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

Nota: El hecho de que la financiación del software libre, al contrario que pasa con el propietario, no pueda suceder 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.

Sección 3. 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 están mostrando en este capítulo es que al tratarse de encuestas en Internet abiertas a todo aquél que quiera participar, la participación depende mucho de los sitios donde se haya anunciado, así como de la forma en que se anunció. 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 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, siendo 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 [debian:developer-map], 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 siguiente tabla, 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, siendo Francia el ejemplo más claro en este sentido, ya que ha conseguido multiplicar por cinco su presencia. Considerando que los primeros pasos de Debian tuvieron lugar en el continente americano (en particular en Estados Unidos y Canadá), podemos ver que en los últimos cuatro años ha sufrido una europeización del proyecto. Suponemos que el siguiente paso será la ansiada mundialización con la incorporación de países sudamericanos, africanos y asiáticos (exceptuando Corea y Japón, ya bien representadas),

aunque los datos que manejamos (2 desarrolladores en Egipto, China e India, 1 en México, Turquía y 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 en el mundo del software libre 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 americana, efecto que queda mitigado por el hecho de que la población europea es mayor que la americana. Nos encontramos entonces ante una situación de guerra de cifras, ya que el número de desarrolladores per cápita favorece entonces a los norteamericanos, pero vuelve a ser favorable a los europeos si tenemos en cuenta en vez de las cifras de población absolutas solamente aquellas personas que cuentan con 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 centro-europeas (Benelux, Alemania y Chequia), seguidos de Australia, Canadá, Nueva Zelanda y Estados Unidos. La zona mediterránea, a pesar de ser importante en magnitudes absolutas (debido a la gran población que tienen Francia, Italia y España), sin embargo, se encuentra por debajo de la media.

Sección 4. Dedicación

El número de horas que los desarrolladores de software libre dedican al software libre es uno de los aspectos más desconocidos. Nótese, además, que se trata de 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 es conocido. 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 los datos 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 desarrolladores no cuenten el tiempo que dedican a leer el correo (o quizás sí), indicando sólo el tiempo que dedican a programar y a depurar. Por eso, todas las cifras que se muestren a continuación han de tomarse con el debido cuidado.

Los estudios que se han realizado hasta ahora muestran que en media cada desarrollador de software libre dedica alrededor de 11 horas semanales [hertel:motivationsurvey:03]. 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 [floss:survey:02] 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 hacía entre once y veinte horas semanales. 9,2% y 7,1% de los encuestados afirmaban respectivamente que el tiempo que dedicaban a desarrollar software libre era entre veinte y cuarenta horas semanales y más de cuarenta horas semanales.

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 poder 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 en conseguirlo.

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 como 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 ambas parecen seguir la ley de Pareto (ver “Estudios cuantitativos”).

Horas semanales	Porcentaje
0 - 2	22,50%
02-may	26,10%
05-oct	21,00%
oct-20	14.1%
20 - 40	9,20%
> 40	7,10%

Sección 5. 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 darse cuenta de 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%, ya que 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 [bezroukov:a-second-look:98] que viene a decir que los líderes de los proyectos de software libre tienen a buena cuenta no compartir toda la información de la 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 se 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.

Mientras la teoría clásica para explicar los motivos por los que los desarrolladores de software libre se dedican a aportar en proyectos de software libre 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 han de enfrentar sociólogos y psicólogos en los próximos tiempos.

Sección 6 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 y como se verá en el capítulo dedicado a la ingeniería del software. Como pudimos 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 radicaban en la forma en la que se preservan la autoría y la opinión del autor (más acentuada en textos que en programas).

En [floss:survey:02] se incluyó una pregunta en la que se instaba a los desarrolladores a indicar qué personas de una lista dada le eran conocidas, no necesariamente personalmente. Los resultados muestran que se pueden aglutinar a estas personas en tres grupos claramente diferenciados:

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

- Linus Torvalds: creador del núcleo Linux, el sistema operativo libre más utilizado. Coautor de Just For Fun: the story of an accidental revolutionary [torvalds:fun].
- Richard Stallman: ideólogo y fundador de la Fundación del Software libre y desarrollador en varios proyectos GNU. Autor de varios escritos muy importantes dentro del mundo del software libre ([stallman:why-free-software-better:98], [stallman:copyleft-pragmatic-idealism:88], [stallman:gnu-project:99], [stallman:gnu-manifesto:85])
- Miguel de Icaza: cofundador del proyecto GNOME y de Ximian Inc. Desarrollador de parte de GNOME y de MONO
- Eric Raymond: impulsor de la Open Source Initiative, autor de La Catedral y el Bazar [raymond:cathedral-bazaar]. Desarrollador principal de fetchmail.
- Bruce Perens: antiguo líder del proyecto Debian. Impulsor (converso) de la Open Source Initiative. Desarrollador de la herramienta e-fence
- Jamie Zawinsky: ex-desarrollador de Mozilla, famoso por una carta en 1999 en la que dejaba el proyecto Mozilla argumentando que el modelo utilizado no iba a dar frutos nunca [jwz:resignation-and-postmortem:99]
- Mathias Ettrich: fundador de KDE. Desarrollador de LyX y otros.

En el segundo grupo tenemos a 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.

- Jörg Schilling, creador de cdrecord entre otras aplicaciones
- Marco Presenti Gritti, desarrollador principal de Galeon
- Bryan Andrews, desarrollador de Apache Toolbox
- Guenther Bartsch, creador de xine
- Arpad Gereoffy, desarrollador de MPEGplayer

El tercer grupo lo componen los nombres de las últimas tres personas, que fueron inventados por el equipo de la encuesta para poder comprobar el margen de error de las respuestas.

De los resultados se pueden observar dos cosas. La primera es que el margen de error de las respuestas se puede considerar pequeño (menor a 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 fama.

Capítulo 5. Economía y negocio

Resumen

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. Se comienza 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 se exponen 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.

Sección 1. Financiación de proyectos

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 altruistamente, 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 esta sección nos vamos a centrar en los proyectos donde 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. De esta manera, 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 quien decide (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 se comentan los tipos de financiación externa más habituales. Mientras el lector se dedique a ellos, conviene, no obstante, no olvidar que esta es

sólo una de formas como los proyectos que construyen software libre consiguen recursos. Hay otras, y entre ellas, la más importante: el trabajo de muchos desarrolladores voluntarios (como se discute en el Capítulo 4, El desarrollador y sus motivaciones)

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 (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 suelen 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 ciertas 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 pueden destacarse las siguientes

* Científica. Este 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 o no con la informática), 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 mejoren. En estos casos, es bastante habitual distribuirlos como software libre. En este caso, los recursos que consiguió el grupo que realiza la investigación se dedicaron en parte a la producción de este software, por lo que se puede decir que fue desarrollado con financiación pública.

* 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 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 directamente adaptar 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, en estos casos 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).

* Social. El software libre es una herramienta de gran interés en al 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 el software libre (normalmente proyectos de desarrollo de nuevas aplicaciones o de adaptación de otras 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 fin. 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 el uso 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 Ada9X 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 fue Ada95), cuyo uso trataba de promover en aquella época. Uno de las causas que se habían identificado en cuanto a la adopción de la primera versión de Ada (Ada83) por las empresas de software era 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 Ada95, asegurándose de que hubiera un compilador de forma prácticamente simultánea con la publicación del nuevo estándar del lenguaje.

Para lograrlo, el proyecto Ada9X contrató un proyecto con un equipo de la Universidad de Nueva York (NYU), por un importe aproximado de 1 millón de USD, para la realización de una "prueba de concepto" de compilador de Ada95. Con esos fondos, y aprovechando de la existencia de GCC (el compilador de C de GNU, del que se aprovechó la mayor parte del dorsal), el equipo de NYU construyó efectivamente el primer compilador de Ada95, que liberó bajo la GNU GPL. El compilador tuvo tanto éxito que a la terminación del proyecto parte de sus constructores fundaron una empresa (Ada Core Tecnologies) que desde entonces se ha convertido en líder en el mercado de compiladores y herramientas de ayuda a la construcción de programas en Ada.

Es notable cómo se puede ver en este proyecto la combinación de elementos de investigación (este proyecto avanzó en el conocimiento sobre la construcción de frontales y de sistemas de tiempo de ejecución para compiladores de lenguajes tipo Ada) y de promoción de estándares (que era el objetivo más claro de su financiador).

Financiación privada sin ánimo de lucro

Este es un tipo de financiación con muchas características similares a las del caso anterior, donde la financiación la 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 considera 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 donde se está estudiando esa enfermedad).

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

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.

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

Financiación por 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, una empresa, para uso interno, necesita de cierta funcionalidad en un programa dado. O bien necesita que ciertos errores en un programa sean corregidos. En este tipo de 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 el proceso de realizarlo, 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 que se realizaran estas mejoras, Corel contrató a Macadamian, que contribuyó con sus mejoras al proyecto Wine. Con ello, tanto Corel como Wine salieron beneficiados.

Financiación con beneficios relacionados

En este caso, 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 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:

* 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 el financiar el proyecto le dará acceso antes que a sus competidores a desarrolladores clave, o simplemente conseguirá con ello 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, al ser libre el software desarrollado, pueden aparecer competidores que vendan aparatos del mismo tipo, usando esos desarrollos, pero sin colaborar a 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 permitía influir para conseguir que los desarrollos que se realicen con más prioridad sean los que más le interesen.

* CDs con programas. Probablemente el modelo de este tipo más conocido 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 CDs con una cierta distribución de GNU/Linux, y por tanto, financiar su desarrollo puede ser un buen negocio para quien los vende.

Es importante darse cuenta de 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 aporte 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 sobre 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.

Nota: 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, colaborar a 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.

Nota: 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, lo 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 nada). A pesar de ello, Red Hat se beneficia de su contribución a GNOME.

Financiación como inversión interna

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

Este tipo de financiación da lugar a varios modelos de negocio. Cuando se analicen éstos ("Modelos de negocio basados en software libre") 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 en algunos casos, 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 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.

Nota: Ximian (antes Helix Code) es un caso bien conocido de desarrollo de aplicaciones libres en entorno empresarial. Muy ligada desde sus orígenes al proyecto GNOME, Ximian ha producido sistemas software como Evolution (un gestor de información personal que tiene una funcionalidad relativamente similar a la ofrecida por MS 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 sus 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.

Nota: Cisco Enterprise Print System (CEPS) [cisco_enter_print_system] 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 estar seguro de los motivos que tuvo Cisco para hacer esto, pero es posible que tuviera que ver con la búsqueda de contribuciones externas (informes de error, nuevos controladores, parches, etc.). En cualquier caso lo que está claro es que, al no tener Cisco ningún plan para comercializar el producto, y no estar muy claro su mercado potencial, no tenía mucho que perder con esta decisión.

Otros modos de financiación

Hay otros modos de financiación difíciles de clasificar entre los anteriores. Entre ellos pueden destacarse, a modo de ejemplo, 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 lo desea pueda entrar en contacto con un desarrollador que pueda 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. Una un desarrollador y un cliente se ponen de acuerdo, tenemos una situación similar a la ya descrita como “financiación por quien necesita las mejoras” (“Financiación por quien necesita mejoras”).

Nota: SourceXchange fue un ejemplo de mercado para poner en contacto a desarrolladores con sus clientes potenciales. Para ofrecer un proyecto, un cliente escribía una RFP (Request for Proposal, o Petición de Propuesta), especificando qué desarrollo se necesita y cuántos recursos se está 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. Cuando un desarrollador y 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, que efectivamente estas tenían sentido, que aconsejaba sobre cómo llevar adelante el proyecto, etc. SourceXchange (propiedad de la empresa CollabNet) se encargaba de ofrecer el sitio, garantizar la competencia de los revisores, asegurarse del pago en caso de que los proyectos se completasen, y ofrecer herramientas para el seguimiento del proyecto (servicios que facturaba al cliente). El primer proyecto mediado por SourceXchange se terminó 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 orientado 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 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 termina 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 podrían repartirse entre muchas (incluyendo individuos), que además sólo tendrían que pagar si finalmente el proyecto termina con éxito. Un mecanismo muy similar a este se propone, con mucho más detalle, en [rasch01:_wall_street_perfor_protoc].

Nota: El sistema de bonos descrito está basado en el Street Performer Protocol (protocolo del artista callejero) [kelsey98:_street_perfor_protoc] kelsey99:_street_perfor_protoc_digit_copyr], 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 puede 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 [harrison02:_ration_street_perfor_protoc]) 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 también una empresa puede hacer esto). En este tipo de organizaciones pueden darse combinaciones variadas de trabajo voluntario con trabajo remunerado. Un ejemplo de este tipo de organización 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, realizando donaciones voluntarias a modo de financiación para el desarrollador.

Sección 2. Modelos de negocio basados en software libre

Además de los mecanismos de financiación de los proyectos, ya tratados, otro aspecto muy relacionada con la economía que merece la pena tratar es el de los modelos de negocio. Ya al hablar de estos mecanismos de financiación 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 alrededor del 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 los basados en la venta de licencias de uso de programas producidos, pues en el mundo del software libre ese es un mecanismo de financiación muy difícil de explotar. Sin embargo, sí se pueden utilizar los basados en el servicio a terceros, con la ventaja de que sin ser necesariamente el productor de un programa puede darse 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 forma 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 menor precio (algo que no puede prohibir la licencia del programa, si éste es libre). En el ejemplo anterior, podría tratar de vender 10 copias a 100 euros cada una, con lo que el producto le saldría gratis (y además dificultaría mucho que el productor original vendiese otra copia a 1.000 euros, si el producto pude obtenerse 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.

Aún 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 sus competidores propietarios, pero superior (y normalmente claramente superior) al coste de copia (incluso cuando se pueden bajar libremente de Internet). Por supuesto en estos casos entran a jugar otros factores, como la imagen de marca o la comodidad para el consumidor. Pero no es este 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 de hacer, y probablemente se obtendrá menos beneficio, pero que puede haber modelos basados justamente en eso.

Mejor conocimiento

La empresa que utiliza este modelo de negocio trata de rentabilizar su conocimiento de un producto (o conjunto de productos) libres. 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 estará especialmente bien situada si es la productora, o participa activamente en el proyecto lo produce.

Esta es una de las razones por la 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.

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 donde aconsejan a sus clientes cómo utilizar mejor el producto en cuestión (especialmente si es complejo, o si su correcto funcionamiento es crítico para el cliente en cuestión).

Relación con los proyectos de desarrollo

Este tipo de empresas tiene 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 a sus empleados que 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 conocen el producto en cuestión. Además, de esta forma podrán seguir muy de cerca el desarrollo, tratando de asegurarse, por ejemplo, de que mejoras demandadas por sus clientes pasan a formar parte de producto desarrollado por el proyecto.

Analizándolo desde un punto de vista más general, esta es una situación donde 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.

Ejemplos

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

> LinuxCare [linuxcare]. 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 Linux ejecutando sobre máquinas virtuales z/VM en grandes ordenadores de IBM. Su modelo de negocio ha cambiado también a mejor conocimiento con limitaciones, al ofrecer como parte fundamental de sus servicios una aplicación no libre, Levanta.
>
> Alcove [alcov]. Fundada en 1997 en Francia, proporciona fundamentalmente servicios de consultoría, consultaría estratégica, soporte y desarrollo para software libre. Desde su fundación, Alcove ha mantenido en plantilla a desarrolladores de varios proyectos libres, y ha tratado de rentabilizarlo en términos de imagen. También ha tratado de 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 Alcove-Labs [alcov_labs]).

Mejor conocimiento con limitaciones

Estos modelos son similares a los expuestos en el apartado anterior, pero tratando de limitar la competencia a 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 esa 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 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 es demandado por sus propios clientes.

Ejemplos

Son muchos los casos donde se usa este modelo de negocio, ya que es común considerarlo como menos arriesgado que el de conocimiento puro. Sin embargo, las empresas que lo han usado han tenido evoluciones variadas. Algunas de ellas son:

> Caldera [calder]. 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 Linux hasta sus demandas contra IBM y Red Hat en 2003, y su 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 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 demás distribuciones de GNU/Linux incluían la parte libre de Caldera OpenLinux, no encontraban en ellas la parte propietaria.
>
> Ximian [ximian]. 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 MS Exchange, muy utilizados en grandes organizaciones. De esta manera trataba de competir ventajosamente con otras empresas que proporcionen servicios basados en GNOME, quizás con los productos desarrollados por la propia Ximian, que no podrían interaccionar tan fácilmente con Exchange. Salvo por este producto, el modelo de Ximian ha sido de mejor conocimiento, y también basado en ser la fuente de un programa (ver más adelante). En cualquier caso, este componente fue liberado en 2005.

Fuente de un producto libre

Este modelo es similar al basado en el mejor conocimiento, pero especializándolo de forma 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 mantenimiento de la aplicación en cuestión, lo que puede ser muy interesante de cara a convencer a posibles clientes de sus capacidades. Igualmente proporciona muy buena imagen de cara a la comunidad del software libre en general, ya que reciben de la empresa un nuevo producto libre que pasa a formar parte del acervo común.

Ejemplo

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

> Ximian. Ya se 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 principales productos, como Evolution o RedCarpet 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 los ha desarrollado 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 [corporation]. 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 es 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. El modelo de negocio de la empresa desde entonces fue producir Zope y productos relacionados con él, y ofrecer servicios de adaptación y mantenimiento para todos ellos. Zope Corporation ha sabido, además, crear una dinámica comunidad de desarrolladores de software libre alrededor de sus productos, y colaborar activamente con ellos.

Fuente de un producto con limitaciones

Este modelo es similares al anterior, pero tomando 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 nada en una licencia libre obliga a distribuir el programa a todo el que lo quiera (esto es algo que quien tiene el software puede hacer o no), lo que hace el productor es distribuirlo durante un tiempo sólo a sus clientes, que le pagan por ello (normalmente en forma de contrato de mantenimiento). Al cabo de un tiempo, el productor 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

Algunos casos de empresas que utilizan este modelo de negocio son lo siguientes:

artofcode LLC [artofcod]. Desde el año 2000, artofcode comercializa Ghostscript en tres versiones (anteriormente lo hacía 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 [ada_core_techn]. 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., y 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 sigue estando basado 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.

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 quiere integrar el producto con uno propietario (algo que la GPL no permite).

Ejemplo

Sleepycat Software [sleepycat]. 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 datos muy popular y 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 permitan empotrarlos sin tener que distribuir el 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.

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 suficiente margen 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 menor 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, tratando de rentabilizar lo más posible esa imagen de marca (desde formación hasta programas de certificación para terceras partes).

Ejemplo

Red Hat [red_hat]. La distribución 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, 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.

Sección 3. Otras clasificaciones de modelos de negocio

Hay otras clasificaciones de modelos de negocio clásicas en la literatura sobre software libre. A modo de ejemplo ofrecemos a continuación una de las más clásicas.

Clasificación de Hecker

La clasificación que se ofrece en [hecker:setting-shop-business:98] 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 ellos 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 él.

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 para él, 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). Modelo similar a 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, y al cabo de él 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 cómo esta clasificación es bastante diferente de la que

hemos ofrecido nosotros, pero aún así algunas de sus categorías coinciden casi exactamente con algunas de las nuestras.

Sección 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 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 personas con las que se relacionan. Por eso cualquier iniciativa dedicada a romper una situación de facto donde 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 dominante. Sólo los cambios tecnológicos producen, durante un tiempo, la inestabilidad suficiente como para que nadie domine claramente. Pero el 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, cuando hablamos de software, lo preocupante es lo que ocurre cuando un producto llega a ser 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 alrededor de él.

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 usa, su formato de datos se convierte en estándar de facto, y las presiones para usarlo (y la aplicación por tanto) son formidables.

Cadenas de distribución. Normalmente uno de los problemas para empezar a usar un programa es obtener una copia de él. Y normalmente es 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 pueda comprarlos. El producto dominante, sin embargo, lo tiene fácil: el primer interesado en tenerlo va a ser al 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 para él, libros que lo describan, entrevistas a sus usuarios, etc.

Inversión en formación. Una vez se han invertido tiempo y recursos en aprender cómo funciona una herramienta, se está muy motivado para no cambiar. 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. El recibir una máquina con software ya instalado desde luego es 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 va a estar dispuesto a preinstalar será solamente el más utilizado.

El mundo del software

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 la evolución del producto, 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 los motivos que se han comentado en el apartado anterior). Ante esto, poco podrán hacer, salvo tratar de 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 mediatizada por las mejoras que le haga a su producto. En el caso general, esta es una situación donde 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. Estos se han convertido en un mercado cautivo.

La situación del 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, serán muchas las 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 motivado para hacer evolucionar el producto precisamente en la línea que sus usuarios quieran. Naturalmente, tendrán la ventaja de un mejor conocimiento del programa, pero eso es todo. Tienen que competir por cada cliente*.

La aparición de productos dominantes se traduce en el mundo del software libre, por lo tanto, en mayor competencia entre empresas. Y con ello los usuarios retoman el control: las empresas en competencia no pueden más 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 de web. Pero hay muchas empresas que están detrás de Apache, desde algunas muy grandes (como IBM) a 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 la situación donde 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 ser echados del mercado. Pero además no pueden separarse demasiado de lo que es "GNU/Linux estándar", pues eso es 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 a decenas de empresas compitiendo y haciendo evolucionar el sistema. Y de nuevo, todas ellas esta detrás de satisfacer 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 ha encargado durante mucho tiempo de coordinar su desarrollo. Hay muchas empresas que hacen mejoras al 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 ha percibido una parte de los usuarios) ha habido espacio para un fork (división) 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).

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. Estas 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, si mencionaremos una que es hasta cierto punto específica del mercado del software, y que ya está siendo experimentada 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 Linux, y para ninguna otra. Eso implicará que sus clientes tendrán

que usar esa distribución de Linux u olvidarse del soporte por parte del fabricante (lo 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, estamos de nuevo ante una situación de empresa monopolística.

Nota: Hasta cierto punto, en el mercado de distribuciones de GNU/Linux se están empezando 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 tenga 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 dicho. El primero es que estas posiciones monopolísticas no serán fáciles de conseguir, y en cualquier caso lo serán por mecanismos en general “no informáticos” (a diferencia de la situación de producto dominante, que como ya vimos es relativamente normal y se llega a ella 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 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 convenientes.

Capítulo 6. Ingeniería del software libre en Software libre

Resumen

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, barata y de mayor calidad: el software libre se diferencia del software tradicional en aspectos más fundamentales, empezando por razones filosóficas y motivaciones, siguiendo por nuevas pautas económicas y de mercado y finalizando con una forma de generar software diferente. La ingeniería del software no puede ser ajena a este suceso y desde hace poco más de un lustro se ha venido 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 en lo que hemos venido a denominar la ingeniería del software libre.

Sección 1. La catedral y el bazar

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 lo hay en el mundo del software libre [5], pero aún 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.

En 1997 Eric S. Raymond publicó el primer artículo ampliamente difundido, La catedral y el bazar [raymond:cathedral-bazaar], en el que se trataba de describir algunas características de los modelos de desarrollo de software libre, haciendo especial énfasis en lo que diferencia 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.

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 roles, destacando 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, dando lugar a unos procesos claramente detallados en los que idealmente cada participante en la actividad tiene un rol 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 que también entran en él 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 e implementación del software. Las tareas que desempeñan estas personas, así como sus roles están perfectamente definidos y alguien que quisiera entrar a formar parte del equipo de desarrollo necesitaría que se le asignara una tarea y un rol según las necesidades del proyecto. Por otra parte, las entregas de este tipo de programas se encuentran espaciadas en el tiempo siguiendo una planificación bastante estricta. Esto supone tener pocas entregas del software y ciclos entre las entregas largos y que constan de varias etapas.

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 están 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 telemáticas.

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 usable, 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 la publicación prontía (release early en

inglés) y 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. Ha de tratárseles 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 existe 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 desarrolladores puede darse el caso de que alguno de ellos encuentre un error y lo solucione, 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 al que descubra un error sea la que finalmente lo entienda y 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 va 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 (en la terminología anglosajona se conoce como release often) y posibilita una gran modularidad [narduzzo:modularity:03] a la vez que maximiza el efecto propagandístico que tiene publicar una nueva versión del software.

Nota: La gestión de nuevas versiones depende lógicamente del tamaño del proyecto, ya que la problemática a la que hay que enfrentarse no es igual cuando el equipo de desarrollo tiene dos componentes a cuando son cientos. Mientras en los proyectos pequeños en general este proceso es más bien informal, la gestión de entregas en los proyectos grandes suele seguir un proceso definido no exento de cierta complejidad. Existe un artículo titulado Release Management Within Open Source Projects [ehrenkrantz03:release] que describe detalladamente la secuencia que se sigue en el servidor web Apache, el núcleo del sistema operativo 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.

Sección 2. 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 con las que ha de contar 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 al proyecto. 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, cuenta con una estructura jerárquica basada en la delegación de responsabilidades por parte de Linus Torvalds, el dictador benevolente. De esta forma, 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 ejemplo claro de cómo la alta modularidad existente en un proyecto de software libre ha propiciado una forma de organización y toma de decisiones específica [narduzzo:modularity:03].

Nota: Existen voces que dicen que la forma de organizarse dentro de los proyectos de software libre se asemeja a la del equipo quirúrgico propuesta por Harlan Mills (de IBM) a principios de la década de los setenta y popularizada por Brooks en su mítico libro The Mythical Man-Month [brooks:mmm:75]. Aunque se pueden dar casos donde 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 propone Mills/Brooks. En cualquier caso, 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 Fundación Apache nos encontramos con una meritocracia, ya que cuenta con un comité de directores formado por personas que han contribuido de manera notable al proyecto. En realidad, no se trata de una rigurosa meritocracia 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 Apache (que se encarga de gestionar varios proyectos de software libre, entre ellos 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 es el caso de FreeBSD o GNOME.

Otro caso interesante de organización formal es el GCC Steering Commitee. 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 en el proyecto egcs (que durante un tiempo fue paralelo al proyecto GCC, pero 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 tome el rol de líder del proyecto. Estudios recientes [barahona03:_unmoun] demuestran que en general los proyectos suelen tener un liderazgo que cambia de manos con frecuencia, pudiéndose 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 deliberadamente 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. Es por ello que

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.

Sección 3. Procesos en el software libre

Aunque el software libre no está necesariamente asociado con un proceso de desarrollo software específico, existe una 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, fundador de proyecto y usuario suelen coincidir en una misma persona. Más adelante, y si el proyecto crece, la captura de requisitos suele tener lugar a través 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 la clasificará según importancia, dependencia, etc. y se podrá monitorizar si ha sido resuelta o no). El uso de esta herramienta para la planificación es más bien reciente, por lo que se puede observar cómo en el mundo del software libre existe una cierta evolución desde la carencia absoluta a 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 y 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 de proyectos, 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 hándicap 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 por que a ojos de los desarrolladores 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 apetecidos desde el punto de vista subjetivo del programador. Históricamente, raro es el caso en el que se han incluidas pruebas unitarias conjuntamente con el código, aún cuando facilitarían la modificación o 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 [reis:mozilla:02]. 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 -refiriéndonos con ello al mantenimiento de versiones antiguas-, es una tarea cuya existencia depende del proyecto. En proyectos donde se requiere una gran estabilidad, como núcleos del sistema operativo, etc., se mantienen versiones antiguas del proyecto, 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 y aconsejan utilizar la versión más moderna con la esperanza de que haya desaparecido por el hecho de que el software ha evolucionado.

Sección 4. Crítica a La catedral y el bazar

La Catedral y el Bazar adolecen 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 se está contando básicamente una experiencia puntual -el caso de Linux- y que se pretenden generalizar las conclusiones para todos los proyectos de software libre. En este sentido, se puede ver en [krishnamurthy:cave:02] que la existencia de una comunidad tan amplia como 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 siguiendo el modelo de desarrollo como el 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 [bezroukov:a-second-look:98] 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 agregar desarrolladores a un proyecto de software retrasado lo retrasa aún más [brooks:mmm:75] no es válida en el mundo del software libre. En [jones:brooks:00] 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.

Sección 5. 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) llegándose 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 fuera publicado a principios de 2000 [ghosh:orbiten-survey:00]. 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 [pareto:political-economy]: el 80% del código corresponde al 20% más activo de los desarrolladores, mientras que el 80% de desarrolladores restante tiene una aportación de un 20% del total. Muchos estudios posteriores han confirmado y ampliado a otras formas de participación diferentes a la aportación de código fuente (mensajes a lista de correo, notificación de errores, incluso el número de descargas como se pueden ver en [hunt:on-the-pareto-distribution:02]) 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 [ghosh:cooking-pot-markets:98], en el que se introduce la economía del regalo en Internet. En [wikipedia:_pareto] se puede obtener más información sobre el principio de Paret o y su generalización en la distribución de Pareto. Asimismo, son interesantes la curva de Lorenz [wikipedia:_lorenz], que representa de manera gráfica la participación en el proyecto de los desarrolladores, y el coeficiente de Gini [wikipedia:_gini], que se calcula a partir de la curva de Lorenz resultando 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 [koch:results-software-engineering:00] 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 lista 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 de 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, aún 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 [mockus00case]. 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 semi-automática de los datos que los proyectos ofrecen públicamente en la red. Al igual que en [koch:results-software-engineering:00], con este artículo no se proporcionó ninguna herramienta ni proceso automático que permitiera ser reutilizada en el futuro por otros equipos de investigación.

En [wheeler:redhat:00] y [wheeler:redhat:01] 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 (véase por ejemplo [gonzalez-barahona01:_count_potat] y [barahona:anatomy-distributions:03]). 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 última versión estable hasta el momento, 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 [barahona:anatomy-distributions:03]. Cabe mencionar que el tamaño medio de los paquetes permanece prácticamente constante a lo largo de los últimos cinco años, por lo que la tendencia natural de los paquetes 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 [boehm:software-engineering-economics:81] 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 esta sección 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 ([robles:evolution:03] o [german:automating-measurement:03]) 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 -poniendo especial hincapié en que sea genérica-, utilizarla para otros proyectos de software libre implica un esfuerzo 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 [libresoft] se pueden seguir con mayor detenimiento estas iniciativas.

Capítulo 7. Entornos y tecnologías de desarrollo

Resumen

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 se tratan las más comunes, y se comentará sobre su impacto en la gestión y evolución de los proyectos.

Sección 1. Caracterización de entornos, herramientas y sistemas

Antes de explicar herramientas concretas, vamos a definir las características y propiedades generales que tienen en función del trabajo a realizar y el modo de organizarse 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 libres. Esto no ha sido así siempre. 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 BSD libres. Hoy 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 medio de máquinas virtuales interpuestas (Java, Python, PHP, etc). En cualquier caso, el entorno y la máquina virtual han de estar suficientemente difundidas y ser lo bastante económicas como para poder reunir suficientes codesarrolladores que dispongan de esas herramientas.

En segundo lugar, también para atraer el mayor número 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 ritmo de trabajo de cada colaborador, sin retrasar a nadie.

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

Sección 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 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[gnustyle]. Otros lenguajes que se le acercan bastante son C++, también soportado por gcc por defecto, y Java, con cierta semejanza y popular por permitir desarrollar para máquinas virtuales disponibles en gran variedad de plataformas. Generalmente no se tienen en cuenta razones de ingeniería software: en SourceForge (ver “SourceForge”), en el año 2004, por cada 160 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 fuentes. El programador libre usará normalmente la versión de GNU[gmake] mucho más que la incompatible de BSD[pmake]. Con ellas puede especificar árboles de dependencias entre ficheros y reglas para generar los ficheros dependientes a partir de aquellos 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 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 mismo cuando 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 complicado y muy dependiente de entornos tipo Unix. Otras alternativas supuestamente mejores, como jam[jam], aap[aap] o ant [ant] son poco usadas (esta última está ganado popularidad, sobretodo en el mundo Java).

Dada la heterogeneidad de sistemas existentes incluso en el mundo Unix, también se usan herramientas dedicadas a ayudarnos a que nuestros programas sean portátiles. Las herramientas de GNU autoconf [autoconf], automake [automake] y libtool [libtool] 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 [gettext] y a las facilidades de internacionalización de la biblioteca estándar de C [glibc] para realizar programas que se pueden 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.

Sección 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 fuentes 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 [gnuemacs], obra estrella de Richard Stallman, escrita y extensible en Emacs Lisp, para el que existen multitud de contribuciones.

Eclipse [eclipse] puede considerarse actualmente como el IDE referencia en el mundo del software libre, con el inconveniente de funcionar 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 [kdevelop] para KDE, Anjuta [anjuta] para GNOME, Netbeans [netbeans] de Sun para Java, y Code::blocks [codeblocks] para aplicaciones C++.

Sección 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 las fuentes 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 70, el correo electrónico fue una extensión del protocolo de transferencia de ficheros de ARPANET.

En el mundo Unix, a mediados de los 70, 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 news (noticias) de USENET, un sistema de foros temático estructurado jerárquicamente y distribuido por inundación a los sitios suscritos a una jerarquía, jugaron un papel fundamental en el desarrollo libre de software, enviándose programas completos en fuente a los grupos de la jerarquía comp.sources.

En paralelo se desarrollaron las listas de correo, entre los que cabe mencionar los gestores de listas de BITNET (1981). Hoy día existe la tendencia a preferir las listas de correo a los grupos de noticias 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 hay 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]. La administración de las listas tradicionalmente se ha hecho por correo electrónico, por medio de mensajes especiales con contraseña. Eso 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 día[mailman] no puede ser administrado por correo electrónico y tiene que hacerse necesariamente vía Web. Las listas de correo juegan un papel crucial en el software libre, llegando en muchos casos[7] a ser el único método de contribución.

Hoy día, con la popularidad del Web, muchos foros son puros foros Web o weblogs, sin otra interfaz que la que se ofrece a través del navegador. Estos pueden ser sean genéricos, como los populares SlashDot[slashdot] o BarraPunto[barrapunto], 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 (ver "Sitios de soporte al desarrollo"). También hay interfaces Web a grupos de noticias y listas tradicionales.

También se ha hecho popular un mecanismo de colaboración basado en Wikis, sobretodo cuando se trata de elaborar un documento conjunto, que puede ser la especificación de un programa, módulo o sistema. Hablaremos de él en la "Wikis"

Finalmente hay que mencionar los mecanismos de interacción donde los desarrolladores conversen 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, ya sea regularmente o en congresos virtuales con fechas acotadas. La herramienta más usada es el IRC (Internet Relay Chat[ircrfc]), que normalmente comunica gente por medio de canales temáticos establecidos por intermedio de una serie de servidores colaboradores. No es común que se utilicen herramientas multimedia (sonido, imagen), probablemente porque puede requerir conexiones de calidad no disponibles para todos, además de problemas de disponibilidad de software libre y la dificultad de registrar y editar los resultados de las conversaciones, con el propósito de documentar.

Sección 5. Gestión de fuentes

A todo proyecto de desarrollo de programas le conviene tener archivada la historia del mismo. Por ejemplo, porque alguna modificación pudo producir un error oculto que se descubre tardíamente y hay que recuperar el original, al menos para analizar el problema. Si el proyecto lo desarrollan varias personas, es necesario también registrar el autor de cada cambio, con las razones del mismo explicadas. Si del proyecto van haciéndose entregas versionadas, es necesario saber exactamente que 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 costo 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 meta-datos necesarios.

Pero también queremos que un sistema de estos 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 concurrentemente, 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 bloqueó los ficheros está ocupado en otras cosas, o incluso se olvidó 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.

CVS

CVS (Concurrent Version Sytem) es un sistema de gestión de fuentes optimista diseñado a finales de los 80, que es usado por abrumadora mayoría en los proyectos libres[cvshome][fogel:cvs][ceder:cvs]. Utiliza un repositorio central al que se accede según un sistema cliente/servidor. El administrador del sitio decide quienes tienen acceso al repositorio o a qué partes del repositorio, aunque normalmente, una vez que un desarrollador ha sido admitido en el círculo de confianza, tiene acceso a todos los ficheros. Además puede permitirse 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 reportarlos, 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

Nos 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 directorios del módulo y ciertos meta-datos (contenidos en subdirectorios llamados CVS), que nos 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 resulta que 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

Otros sistemas de gestión de fuentes

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

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

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

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

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.

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 a la rama estable, es preciso conocer en detalle que correcciones se han hecho ya, para no hacerlas varias veces.

CVS depende de un servidor centralizado y, aunque puede hacerse trabajo estando desconectado, necesitamos conexión con el mismo para generar versiones, compararlas y mezclarlas.

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

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

Y sin embargo existen otros sistemas libres que solucionan varios de estos problemas. Podemos destacar el ya designado sucesor de CVS: subversion[subversion][ora:subversion], 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 espectativas, ya que por un lado depende de la accesibilidad y buen funcionamiento del servidor, y por otro depende de los administradores de ese servidor el 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 puede mezclar o no con la oficial. Así funcionan GNU arch[arch] o bazaar[bazaar], así como sistema propietario BitKeeper[bitkeeper], 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 por ser propietario, con una licencia que permitía a los proyectos libres obtenerlo gratuitamente siempre que todas las operaciones de compromiso de cambios con sus meta-datos se registren 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], 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 grandemente 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]).

Sección 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 haber ya herramientas libres, como OpenOffice. 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 (“Licencia de documentación libre de GNU”), exigen formatos transparentes, sobre todo por facilitar el trabajo a los que realicen documentos derivados.

Las herramientas WYSIWYG (what you see is what you get) generalmente no contienen más información que la estricta de visualización, haciendo 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, siendo 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.

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[sgml] y XML[xml], donde 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 ir a DocBook[walsh:docbook].

DocBook es una aplicación de SGML originalmente desarrollada para documentación técnica informática, y hoy con una variante XML. DocBook es hoy el estándar de formato de documentación libre para muchos proyectos (Linux Documentation Project, KDE, GNOME, Mandriva Linux, etc.) y un objetivo a 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, siendo la más popular el modo psgml de emacs. También es pesado de procesar y los procesadores libres aún generan una calidad de documentos poco atractiva.

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[ward:wiki], 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, 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, o las celdas de 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 (a través de un NombreWiki, casi siempre dos palabras juntas con la primera letra en mayúsculas). Casi ningún Wiki admite hiperenlaces dentro de la misma página.

Al contrario que en CVS, cualquiera puede escribir en un Wiki, aunque se aconseja que se identifique el autor con un registro previo. Cuando se visita un Wiki veremos que todas las páginas tienen un botón para permitir su edición. Si se pulsa, el navegador nos mostrará en un formulario el 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 quien las hizo y cuando. También se pueden comparar con facilidad. También 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 hacen. Para ello puede suscribirse a los cambios, recibiendo notificaciones de los mismos por correo electrónico. A veces el que 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, bien alguien que asuma el rol de editor, puede emplearlos para reformar el texto original, posiblemente moviendo frases de los comentarios a los sitios oportunos.

Sugerencia: La mejor forma de entender un Wiki es accediendo a uno y experimentando en una página destinada a ello, denominada habitualmente SandBox.

Sección 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 ella sienta que sus informes o soluciones son tenidos en cuenta. Por ello es necesario un mecanismo sencillo de informar de errores, de modo que los desarrolladores reciban información suficiente, de forma sistemática, y con todos los detalles necesarios, ya sea aportados por el colaborador, como la explicación de lo que pasa, nivel de importancia y posible solución, ya por algún mecanismo automático que determine, por ejemplo, la versión del

programa y del entorno en que funciona. Así mismo es necesario que los errores se guarden en una base de datos que pueda ser consultada, para ver si un error ya ha sido reportado, si ha sido corregido, su nivel de importancia, etc.

Existen varios de estos sistemas, de filosofía diferente. Unos son vía Web, otros vía correo electrónico, por medio 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 reportados sobre el mismo. 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 fuentes, normal, menor o sugerencia) y etiquetas sobre la categoría (por ejemplo, seguridad). Tras ello, si confirmamos la petición, automáticamente averigua la versión del paquete y de aquellos de los que depende, así como la versión y arquitectura del núcleo. Obviamente la dirección de correo electrónico la sabe, por lo que un informe similar al siguiente es enviado al sitio correcto:

```
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 vía 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 requiren también 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 les puede obligar a nada. No obstante sí se pueden definir tareas y esperar a que alguien se de de alta en el sistema y las asuma, declarando un plazo para ello. Se tenga o no control sobre lo que pueden hacer determinadas personas, siempre es conveniente tener controladas las tareas que hay que hacer, qué dependencias tienen, su nivel de importancia y quiénes están trabajando en ellas. Muchos de los proyectos importantes gestionan estos temas con BugZilla[bugzilla:guide] 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 acostumbrado. 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 seguimiento de esos errores puede ser conveniente federar distintos sistemas, como hace Malone [malone].

Sección 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 ("SourceForge") 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 demanda de más recursos, y de más tiempo de administrador. Ya el mservicio 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 debe 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 ejecución de las pruebas de regresión se hagan sistemáticamente, de forma automática, por ejemplo todas las noches, si ha habido cambio en los fuentes. 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 [moz:tinderbox] 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.

Sección 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 y concentrarse en su proyecto.

SourceForge

Uno de los primeros en establecerse este tipo de servicios y el más popular es SourceForge [sourceforge], gestionado por 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 fue 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 entrar, si ya se es. Para colaborar en el sitio, conviene hacerse miembro, siendo imprescindible hacerlo se quiere crear un proyecto nuevo o participar en uno ya existente. Para ser espectador no es necesario, y como tal podemos ver cuales son los proyectos que experimentan un desarrollo más activo o son descargados con más frecuencia; 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, estado (alfa, beta, producción), descriptores (lenguaje, sistema operativo, temática, tipo de usuarios, idioma, licencia), errores y temas pendientes o subsanados, detalles 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 (por ejemplo, no nos pueden responder).

Cualquier usuario autenticado puede solicitar dar de alta un proyecto, que es admitido por los administradores del sitio si cumple las políticas del mismo, y 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 los proyectos con baja o nula actividad apenas son visibles, ya que se muestran ordenados por ella

en las búsquedas. 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:

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 a través de 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 criterio de un administrador.

Un servicio de noticias, donde los administradores anuncian 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. Similar a un rastreador, permite 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 la tarea.

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 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, 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, un usuario recibirá por correo electrónico notificaciones lo que quiere vigilar.

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. Así mismo 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 demostrar 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 ellas, entre los que destacan Savannah[savannah], dedicado al proyecto GNU y a otros programas con licencia tipo copyleft, o BerliOS[berlios], 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[fsfesavannah].

Otro ejemplo de sistema de gestión de proyectos de software libre es Launchpad[launchpad], utilizado por Ubuntu para el desarrollo de cada versión de la distribución. Launchpad no es un repositorio de código fuente, sino que su propósito es 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. Launchpad no es software libre, y su código no está derivado del de SourceForge.

Otros sitios y programas

Naturalmente se han desarrollado y siguen desarrollándose sistemas de colaboración y algunas empresas hacen negocio del mantenimiento y servicio de esos sitios. Por ejemplo, el proyecto Tigris[tigris], que sólo mantiene proyectos de ingeniería de software libre, usa un portal de colaboración (SourceCast) mantenido por una empresa servicios (CollabNet), que también

mantiene sitios de proyectos individuales, como OpenOffice. Nuevos sitios vienen adoptando nuevo software libre. como GForce[gforge], utilizado por el proyecto Debian[alioth]. Pueden verse una comparación exhaustiva de muchos sitios en [fosphost].

Capítulo 8. Otros recursos libres

Resumen

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. Hay diferencias cuyas implicaciones han hecho que no se desarrollen con la misma fuerza que los programas. La principal es que basta 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 a la puesta en producción de un hardware descrito en un lenguaje apropiado.

Sección 1. Recursos libres más importantes

De la documentación de programas y otros documentos técnicos ya hablamos en la "Documentación de programas". Hablaremos aquí más 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.

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 pueden obtener ingresos a partir de contratos de investigación obtenidos gracias al prestigio obtenido.

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 especialistas.

Por ello han surgido numerosas iniciativas de revistas en la red, entre las que destacan la veterana First Monday[firstmonday] o el proyecto Public Library Of Science (PLOS[plos]). En [doaj] 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 (ver “Licencias de Creative Commons”).

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[kelty:freescience].

Leyes y estándares

Hay documentos cuyo carácter es normativo, que definen cómo deben hacerse las cosas, ya sea para facilitar la convivencia entre las personas, ya 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, ya sea entre máquinas remotas o 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 pueda intentar justificarse esto para cubrir parcialmente los gastos, la libre difusión del texto de los estándares ha resultado mucho más productiva. Este es el caso de las recomendaciones del W3C[13] y sobre todo 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.

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 medios similares. En [bcp9] y en [ietf:tao] 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 [w3c:lic]. 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 existen 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 sean productos que interoperen.

Lo mismo que es conveniente tener una definición de software libre o abierto, también es necesaria una definición de estándares abiertos. Bruce Perens[perens:openstandards] ha propuesto una basada en los principios siguientes:

Disponibilidad. Si es posible, proporcionar incluso una implementación libre de referencia.

Maximizar las opciones del usuario final.

Sin tasas sobre la implementación (no así sobre la certificación, aunque aconseja la disponibilidad de herramientas libres de autocertificación).

Sin discriminación de implementador.

Permiso de extensión o restricción (no certificable).

Evitar prácticas predatorias por fabricantes dominantes. Toda extensión propietaria debe tener una implementación libre de referencia.

Enciclopedias

En 1999 Richard Stallman lanza la idea de una enciclopedia libre [free-encyclopedia] como un mecanismo para evitar la apropiación del conocimiento y proporcionar acceso universal a documentación formativa. Estaría formada por artículos contribuidos por la comunidad, sin un control centralizado, donde distintos actores asumirían distintos roles, entre los que se aconseja, pero no se obliga, el de revisor. Esta enciclopedia no contendría solamente texto, sino elementos multimedia y software educativo libre.

Han surgido varias iniciativas para realizar esta visión. Por ejemplo, la Nupedia[nupedia] ha tratado sin éxito de construir una enciclopedia de calidad, quizá por requerir un formato relativamente difícil de aprender (TEI), aunque seguramente en mayor medida por el requisito de que todos los artículos necesiten un editor, revisores científicos y de estilo, etc.

Heredera de la nupedia, pero con mucho más éxito que ésta ha sido la Wikipedia[wikipedia]. Wikipedia es una enciclopedia libre plurilingüe basada en la tecnología wiki. Wikipedia se escribe de forma colaborativa por voluntarios, permitiendo 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, eliminando los obstáculos que imponia la 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 de sistema de texto estructurado extraordinariamente simple, como se vió en “Wikis”. En febrero de 2007, el número de artículos en inglés en la wikipedia supera la cifra de 1.500.000, y en español se han alcanzado los 200.000 artículos.

Wikipedia es un proyecto de la fundación sin ánimo de lucro Wikimedia, la cual mantiene además los siguientes proyectos con la misma fórmula que la Wikipedia:¿Decir algo de la Fundación?

Wikcionario [wikcionario]. Se trata de un proyecto colaborativo para producir un diccionario multilingüe gratuito, con significados, etimologías y pronunciaciones, en aquellas lenguas en las que sea necesario.

Wikilibros [wikilibros]. 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 [wikiquote]. Es una recopilación de frases célebres en todos los idiomas, incluyendo las fuentes cuando éstas se conocen.

Wikisource [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 [wikispecies]. Es un repertorio abierto de especies animales, vegetales, hongos, bacterias y todas las formas de vida conocidas.

Wikinoticias [wikinoticias]. Es una fuente de noticias de contenido libre en los usuarios son los redactores de las noticias.

Commons [commons]. Repositorio libre de imágenes y multimedia.

Wikiversidad [wikiversidad]. Plataforma educativa online libre y gratuita, basada en proyectos de aprendizaje a cualquier nivel educativo.

Meta-Wiki [metawiki]. Sitio web de apoyo a los proyectos de la fundación Wikimedia.

Cabe destacar también la Concise Encyclopedia of Mathematics[wolfram:math] con un concepto de libertad más limitado (sólo se puede consultar en la red) y un modelo de desarrollo que necesariamente pasa todas las contribuciones por un comité editorial.

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 en 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:

Cumplir su misión como agente difusor del conocimiento.

No cuesta mucho hacer disponibles materiales existentes a todo el mundo.

Estos materiales no sustituyen a la enseñanza presencial.

Son propaganda que puede atraer alumnos y que contribuyen al prestigio de la Universidad.

Permiten crear una comunidad de docentes que revisen los materiales y los mejoren.

La iniciativa más notable en este sentido es la del MIT [mit:opencourseware] que prevé poner accesibles más de 2000 cursos de forma coherente, uniforme y bien catalogados.

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 ODP (Open Directory Project [dmoz]), operado por Netscape y mantenido por 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[odp:socialcontract], inspirado en el de la distribución Debian[debian:socialcontract], que facilita la colaboración exterior, asegurando que siempre será libre, con políticas públicas, autorregulado por la comunidad, 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 su fácil ejecución.

Hardware

La libertad en el hardware tiene dos aspectos. El primero es la necesidad de que las interfaces y 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 para poder 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 Open Cores

[opencores], para circuitos integrados.

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. ¿Que razones puede tener un artista para conceder libertades de copia, modificación y redistribución? Por una lado dar a conocer al artista y favorecer la difusión de su obra, lo que puede permitirle obtener ingresos por otras actividades, como conciertos y obras de encargo. Por otro lado 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 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 les hace funcionar inmediatamente en un ordenador, mientras que otros, como la escultura, no se pueden. No existen muchas iniciativas en arte y literatura libres, y éstas son muy diversas. Podemos mencionar las novelas del colectivo Wu Ming[wuming].