Software libre en educación

Jordi Adell
Iolanda Bernabé

Depto. de Educación
Universitat Jaume I
Castellón (España)
v. 2

1. Introducción

Este capítulo trata sobre el software libre en educación. Está dirigido a docentes en activo de todos los niveles, a gestores educativos y, especialmente, a estudiantes que se están preparando para una profesión relacionada con la educación. No presupone conocimientos previos más allá de los de un usuario informático normal: manejo del sistema operativo, de aplicaciones ofimáticas y de algunas aplicaciones Internet. Se asume, también, que el lector "ha oído hablar" del software libre y es posible que utilice alguna aplicación libre descargada de Internet.

Los autores se han marcado cuatro objetivos esenciales. El primero es introducir al lector en los conceptos clave del software libre, su definición, su origen y algunas de sus implicaciones. El segundo objetivo es incitar al lector a probar el software libre y a comprobar sus ventajas prácticas sobre el software privativo. El último apartado, las actividades, está dedicado a ello. El tercer objetivo es animar a reflexionar sobre la relación entre los valores que encarna el software libre y los fines de la educación pública. Pretendemos que los estudiantes desarrollen los conocimientos y las capacidades necesarias para integrarse adecuadamente en esta compleja y contradictoria sociedad de la información del siglo XXI, para ser ciudadanos libres, participativos y solidarios, para ser profesionales competentes, pero ¿es posible si en nuestras escuelas, institutos y universidades sólo conocen software privativo, basado en la idea de que investigar y comprender cómo funciona o compartirlo con los demás son actividades delictivas?

Finalmente, nos gustaría analizar si es posible utilizar en educación el modelo de desarrollo del software libre y las ideas sobre libertad y cooperación que lo sustentan. Los educadores creamos materiales formativos y actividades didácticas para nuestros alumnos, que podemos elaborar colaborativamente y distribuir en la Internet. Podemos aprender mucho de la filosofía y las prácticas de los programadores de software libre y de experiencias como la Wikipedia, la mayor enciclopedia del mundo, en la que cualquiera puede colaborar. En otro lugar (Adell, 2006) hemos propuesto algunas ideas sobre este tema.

Como puede verse, desde nuestra perspectiva, el software libre no es sólamente un

tipo de software o una forma de “licenciar” software, es un fenómeno social y cultural complejo, que tiene un indudable interés teórico y práctico para la educación.

2. Código fuente y código máquina

Antes de explicar qué es el software libre necesitamos conocer algunos aspectos clave del software, por ejemplo, cómo se produce y qué es “el código fuente” de los programas. Por software entenderemos tanto el sistema operativo como las aplicaciones que utilizamos los usuarios. Además del software, son muy importantes los formatos de fichero (la manera de organizar y codificar la información que producen las aplicaciones) y los protocolos de comunicación (la forma de comunicarse entre sí los programas a través de las redes). Pero de ellos hablaremos más adelante.

Un programa no es más que un conjunto de instrucciones que le dicen al ordenador qué tiene que hacer. Los programas los escriben los seres humanos utilizando lenguajes de programación. Pero antes de que el ordenador pueda ejecutar un programa es necesario traducir dichas instrucciones a su lenguaje, esto es a “código máquina”: largas series de ceros y unos. A fin de cuentas, un ordenador no es más que una gran cantidad de interruptores eléctricos, que pueden estar únicamente en uno de dos estados: dejar pasar la corriente o no dejarla pasar, uno o cero. Un programa le dice al ordenador qué interruptores poner a “cero” y cuáles poner a “uno” en cada momento, dependiendo de las acciones que ejecutemos en el teclado o con el ratón. El proceso de convertir un programa escrito en un lenguaje de programación a instrucciones inteligibles para el ordenador, se denomina “compilación” y lo hacen otros programas de ordenador especializados: los compiladores. Una vez el programa está compilado ya es posible ejecutarlo, a cambio, una vez “traducido” a código máquina, es casi imposible que un ser humano entienda algo de la larga serie de unos y ceros en que se ha convertido. Y este es el meollo del software libre.

Pero antes veamos un ejemplo (Hart, 2003). El primer programa que se suele escribir cuando se aprende a programar (una tradición informática reflejada en muchos manuales de programación) se denomina “Hello world” (“Hola mundo”) y consiste en hacer que el ordenador escriba las palabras “Hello world” en la pantalla. Sencillo, ¿no? En el siguiente cuadro presentamos algunos ejemplos de programas de este tipo escritos en diferentes lenguajes de programación (C++, Java y Python). En la última línea presentamos las palabras “Hello world” en código binario.

Lenguaje de programación	Código fuente

C++	`#include <iostream>`
	` int main()`
	`{`
	`  std::cout << "Hello World\n";`
	`  return 0;`
	`}`

Java	`class helloworld`
	`{`
	`  public static void main(String args[ ])`
	`  {`
	`  System.out.println("Helb World");`
	`  }`
	`}`

Python	`print "Hello World"`
Las palabras "Hello world" en código ASCII (binario)	`01001000011001010110110001101100011011110010000001010111011011110111001001110110001100100`

Tabla 1: Lenguajes de programación, código fuente y código máquina

Ahora, ¿se siente capaz de cambiar alguno de los programas de la Tabla 1 para que digan "Hola mundo" en lugar de "Hello world"? Parece bastante sencillo en cualquiera de ellos, mire en el código y verá la cadena alfanumérica "Hello world". Pero... ¿y en binario? ¿Sería capaz de traducirla? Y eso que no se trata de un programa, sólo de las palabras "Hello world" escritas en código ASCII. Esta es, en esencia, la diferencia entre código fuente, escrito en un lenguaje de programación de alto nivel, inteligible para los seres humanos, y el código binario, hecho a base de unos y ceros, inteligible para los ordenadores. Un programa normal de ordenador puede contener varios millones de líneas de ese tipo.

Cuando alguien compra un programa no-libre, el vendedor se limita a facilitarle el código máquina ejecutable, es decir, una tira ininteligible de ceros y unos. En cambio, cuando usa software libre, dispone también del código fuente en el que está escrito el programa. Por eso se dice también que el programa es de "código abierto", porque su código fuente se puede "ver" y "tocar". Un programador puede hacer cambios, arreglar errores, añadir nuevas funcionalidades y difundir sus mejoras para que otras personas se beneficien de su trabajo. No sólo es técnicamente posible, además es legal. En cambio, en el software privativo es muy difícil hacer cambios o siquiera entender cómo está hecho el programa, y además, de acuerdo con la licencia o contrato de compra, es un delito intentar "descompilar" el programa o hacer cualquier modificación. Nadie, excepto la empresa propietaria, puede hacer cambios en el programa. Por tanto, nadie puede comprobar qué hace realmente el software privativo ni aprender nada de él.

Ya tenemos algunos elementos para comprender la diferencia esencial entre el software libre y el software privativo. Pero alrededor del software libre hay muchas confusiones, algunas interesadas, otras producto de la ambigüedad del lenguaje, otras fruto de su historia y de las disensiones entre sus propios defensores. Necesitamos una buena definición. Desgraciadamente hay dos.

3. Las definiciones de software libre

El hecho de que en inglés, el idioma en el que se acuñó y difundió el término *software libre* (*free software*), una misma palabra (*free*) signifique tanto "libre" como "gratuito" y que gran parte del software libre sea efectivamente gratuito, ha favorecido las malas interpretaciones: mucha gente considera equivalente los términos *software libre* y *software gratuito*. Sin embargo, el rasgo esencial que define el software libre es la libertad, no el precio. Cuando se habla de software libre (*free software*) debemos pensar en "libertad de expresión" (*free speech*), no en "cerveza gratis" (*free beer*). El propietario de los derechos sobre el software libre garantiza a los usuarios, mediante una *licencia,* una serie de *libertades* que no otorga el propietario del software privativo, que se reserva numerosos derechos en base a la legislación sobre propiedad intelectual (por ejemplo, no permite el acceso al código fuente o no permite ninguna modificación y su subsecuente distribución). El usuario de software privativo en realidad paga por el derecho a *usar,* con numerosas limitaciones, el software. Pero pagar por él, no lo convierte en algo de su propiedad. Más adelante trataremos este punto con mayor detenimiento.

Una idea crucial para entender el revuelo que ha creado el software libre es que *software libre* es mucho más que *software.* Así, se ha asociado a un modelo de desarrollo del software (Raymond, 1999), como una comunidad de prácticas (Edwards, 2004; Tuomi, 2005), una "escena" (Lehman, 2004), una aproximación a cierto tipo de licencias (Perens, 1999), un modelo económico (Khalak, 2000; Lerner y Tirole, 2000), un sistema social y de valores (Lessig, 2004; Stallman, 1992), un movimiento social por una cuestión ética (Wynants y Cornelis, 2005) o un modelo híbrido, público-privado, de innovación (Lyn, 2005). Todas estas perspectivas, junto a estrategias dirigidas a despolitizar las ideas que subyacen al concepto original de *software libre* y a hacerlo "digerible" a la industria, han producido bastante confusión. En la Internet y en la bibliografía se suelen emplear cada vez más acrónimos como FLOSS (*Free, Libre, Open Source Software*) o FOSS (sin *Libre*) o FS/OS para no entrar en polémicas sobre si lo importante es la libertad o el código abierto.

El origen de las ideas que subyacen al software libre hay que buscarlo en las prácticas de los primeros programadores informáticos, la cultura "hacker" creada por los primeros programadores en los laboratorios de universidades y centros de investigación norteamericanos en los primeros tiempos de la informática, que asumían que compartir el conocimiento (y el código) libremente entre ellos era normal e incluso beneficioso para el avance del conocimiento. Pero las bases ideológicas del movimiento del software libre se asentaron en la década de los 80 gracias a la visión de una persona, Richard Stallman. Stallman, horrorizado por el camino que estaba tomando la naciente industria del software, con todas sus restricciones y prohibiciones

a los usuarios, se propuso crear un sistema operativo completo, tipo UNIX, completamente libre, que bautizó como GNU, así como a difundir los principios y las ventajas del software libre. Para conseguir sus objetivos creó la *Free Software Foundation* (FSF).

3.1. La definición de la *Free Software Foundation* (FSF): el software libre

La FSF propugna como elemento clave para la definición del software libre la libertad de la comunidad de usuarios para poder ejecutar, copiar, estudiar, mejorar y redistribuir el software. La palabra clave aquí es *libertad*. Libertad de usar el programa para cualquier propósito, de estudiar cómo funciona y adaptarlo a las diferentes necesidades, de distribuir copias, de poder mejorarlo y de hacer públicas las mejoras. La única restricción es que si redistribuimos el programa, tenemos que hacerlo reconociendo los mismos derechos en los usuarios de nuestras modificaciones. Como vemos, el acceso al código fuente es un requisito previo y necesario para ejercer la mayoría de estas libertades.

En concreto, la FSF se refiere a cuatro libertades que deben tener los usuarios del software para que pueda ser calificado como libre (Stallman, 1996):

- Libertad 0. Es la libertad de usar el programa para cualquier propósito.
- Libertad 1. La libertad de estudiar cómo funciona el programa y adaptarlo a las propias necesidades. Una condición previa para que se dé esta libertad es el acceso al código fuente.
- Libertad 2. La libertad de redistribuir copias.
- Libertad 3. La libertad de mejorar el programa y hacer públicas las mejoras, de modo que toda la comunidad se beneficie. Esta libertad también requiere el acceso al código fuente.

Por tanto, un programa puede definirse como software libre sólamente si los usuarios tienen todas estas libertades. Las cuatro libertades básicas de la FSF se concretan en la licencia GPL GNU (GNU *General Public License*). El tema de las diferentes licencias de software se trata en el apartado 4.

3.2. La definición de la *Open Source Initiative* (OSI): el software de *código abierto*

En 1998, algunos programadores y activistas del software libre, como Eric S. Raymond, Bruce Perens y Tim O'Reilly entre otros, crearon la *Open Source Initiative* (OSI). Eric Raymond había escrito poco antes el ensayo sobre ingeniería de software *La catedral y el bazar*

(Raymond, 1997), en el que describía el proceso de desarrollo de Linux como una manera nueva, diferente y muy eficiente de crear software. La OSI nació con el objetivo de crear y fomentar el uso de programas informáticos de *código abierto*. La razón por la que la OSI defiende el acceso al código fuente de los programas no es la libertad, sino la posibilidad de crear mejor software, adaptado a necesidades reales de los usuarios. La OSI se centra en destacar las ventajas pragmáticas de la utilización del software de código abierto, basadas en la constante exposición de la estructura y funcionamiento de los programas al escrutinio de la comunidad y en la capacidad de implicación y participación de sus usuarios en su mejora. El resultado final es que, según la OSI, el software de código abierto tiende a ser de mejor calidad que el privativo.

Para que un programa pueda ser considerado de código abierto, la OSI establece un decálogo de condiciones que ha de cumplir. En su versión 1.9 (OSI, 2006), y tal como las reproduce Mas (2005, pág. 33-35), son son las siguientes:

1. Libre distribución. No se puede impedir la venta o distribución del programa o parte de él. Así mismo, tampoco se puede exigir el pago de un canon o tasa a cambio de su distribución por parte de terceros.

2. Código fuente. El programa debe incluir su código fuente y no se puede restringir su redistribución.

3. Trabajos derivados. No debe impedirse realizar modificaciones o trabajos derivados del programa y debe permitirse que éstos sean distribuidos bajo los mismos términos que el software original.

4. Integridad del código de fuente original. Puede exigirse que una versión modificada del programa tenga un nombre y número de versión diferente que el programa original para poder proteger al autor original de la responsabilidad de estas versiones.

5. No discriminación contra personas o grupos. Las condiciones de uso del programa no pueden discriminar a una persona o un grupo de personas.

6. No discriminación contra usos. No se puede negar a ninguna persona el uso del programa para ningún fin como, por ejemplo, el comercial o el militar.

7. Distribución de la licencia. Los derechos del programa deben aplicarse a todos quienes se redistribuye el programa sin ninguna condición adicional.

8. La licencia no debe ser específica de un producto. Los derechos garantizados al usuario del programa no deben depender de que el programa forme parte de una distribución o paquete particular de software.

9. La licencia no debe restringir otro software. La licencia no debe poner restricciones en otros programas que se distribuyen junto con el software licenciado.

10. La licencia debe ser tecnológicamente neutra. No puede existir ninguna disposición en la licencia que obligue al uso de una tecnología concreta.

Pero, ¿cómo se obtienen y garantizan las libertades en el software? ¿Cómo saber qué derechos tenemos o no tenemos respecto a un programa que hemos conseguido en una tienda o bajado de Internet? ¿Qué es *la licencia*? Las condiciones de uso del software por parte del usuario las proporciona la licencia, un documento legal que acompaña al software y que poca gente lee.

4. Las licencias en el software privativo y libre

Se denomina “acuerdo de licencia de software” a un contrato entre un productor o propietario y un usuario de un programa de ordenador. Si el usuario es una persona, se le denomina “usuario final”. Un EULA o *End User License Agreement* (Acuerdo de Licencia de Usuario Final) es el contrato entre un propietario y un usuario final. El EULA especifica los límites de los permisos garantizados por el propietario a dicho usuario final, es decir, lo que podemos y no podemos hacer con el software adquirido.

4.1. Licencias privativas

La lectura atenta de un EULA de cualquier programa comprado en una tienda es un ejercicio muy conveniente para entender qué nos ofrece el software libre frente al software privativo. Físicamente, los EULA son esos documentos que van dentro de las cajas del software que venden en las tiendas y que nunca nadie suele leer (tienen la letra muy pequeña y un estilo francamente farragoso). También pueden aparecer durante la instalación del programa en el ordenador: es una de esas pantallas en las que sale un fragmento de un texto más largo, que se nos dice que “leamos atentamente” y que cliquemos sobre el botón “Acepto” si estamos de acuerdo. Veamos resumidamente qué dice un EULA típico.

Lo primero que sorprende es que no empieza con algo así como “La parte contratante de la primera parte será considerada como la parte contratante de la primera parte”. Lo que dice en conjunto es que, aunque tengamos esa impresión, en realidad no hemos “comprado” el software, sino que simplemente hemos adquirido el derecho a usarlo con numerosas restricciones, es decir, que el programa no es *nuestro*. En segundo lugar, ¡sorpresa! que al desprecintar la caja ya hemos aceptado todas sus condiciones... sin haberlas leído siquiera. Lo que viene a

continuación es, en esencia, una larga lista de prohibiciones para el usuario: no podemos instalar el software más que en un ordenador o en todo caso, sólamente podemos ejecutar el software en un único ordenador a la vez (dependiendo del tipo de software), no podemos hacer más de una copia de seguridad, no podemos descompilarlo o intentar de ninguna manera ver cómo está hecho por dentro, no podemos alquilarlo, venderlo o prestarlo más que una sola vez a otro usuario final y, en tal caso, tenemos que destruir toda la copia de seguridad, la garantía es limitada, etc. En resumen, no tenemos ningún otro derecho que el de usar el programa en un ordenador determinado, tal y como es, y nos sometemos a todas las obligaciones, condiciones y prohibiciones imaginables, incluyendo el hecho de que usándolo autorizamos a la empresa propietaria del software a que reúna datos sobre nosotros mismos y nuestro ordenador cada vez que iniciamos o actualizamos el programa, sin preguntar siquiera qué nos parece la idea. Si encontramos algún fallo, no podemos arreglarlo. En el pasado, algunas licencias prohibían incluso publicar cualquier análisis del funcionamiento del software sin permiso previo del propietario (es decir, “Ud. no puede ni siquiera hablar de algo que es mío, mío y sólo mío”). Increíble... y seguramente ilegal en cualquier país con una constitución democrática, que reconozca la libertad de expresión de sus ciudadanos.

Las licencias de software libre, en línea con las libertades de acceso, modificación y redistribución del código, otorgan permisos expresos a los usuarios que no suelen estar reconocidos en las leyes de propiedad intelectual, diseñadas para defender únicamente los derechos del propietario de la obra. Hay muchas licencias libres o de código abierto, pero los principales tipos se describen a continuación.

4.2. Licencias libres

Licencias BSD

La licencia BSD (*Berkeley Software Distribution*), creada por la Universidad de Berkeley, es una de las menos restrictivas. Permite la redistribución y las modificaciones del software y no impone restricciones a cómo puede ser utilizado el código (por ejemplo, dentro de productos comerciales). Por eso, hay código licenciado BSD en productos comerciales, como Windows o en la parte no-libre de Mac OS X. Como casi única restricción, incluye la prohibición de usar el nombre del propietario de los derechos para la promoción de productos desarrollados a partir del original.

La Licencia Pública General de GNU (GNU GPL)

La *GNU General Public License* es la licencia del proyecto GNU, propuesta por la *Free Software Foundation y* la más utilizada hoy en día en el software libre. Fue ideada por Richard Stallman para impedir que el software que se creara en el proyecto GNU pudiera convertirse en

privativo y negar las libertades que reconoce a los usuarios. Se denomina *copyleft* y, genialmente, utiliza las leyes de *copyright* para otorgar libertades a los usuarios, no para reservar derechos a los propietarios, y asegurar que éstos lo harán del mismo modo si modifican el código del programa y lo redistribuyen. El propio Stallman cuenta la anécdota de la asignación del nombre a este tipo de licencia "*all rights reversed*", en contraposición al "*all rights reserved*" del *copyright*, al relatar la historia del proyecto GNU (Stallman, 2002).

La Licencia Pública General Menor de GNU (GNU LGPL)

La *Lesser GNU Public License* o LGPL ha sido también creada para el proyecto GNU de la *Free Software Foundation*. Esta licencia, menos restrictiva, permite específicamente la integración con cualquier otro software sin prácticamente limitaciones.

Licencias para la documentación del software

Además de establecer las condiciones de uso de los programas, también se puede regular la utilización de su documentación. Para ello, Richard Stallman propuso una licencia de documentación libre de GNU. Es la licencia *copyleft* de la *Free Software Foundation* para documentos de carácter técnico o pedagógico relacionados con los programas informáticos.

Licencias *Creative Commons*

Otras licencias muy difundidas son las *Creative Common*s (Xalabarder, 2006), desarrolladas principalmente para licenciar recursos intelectuales de carácter literario, científico y/o artístico y que permiten a los autores mayor flexibilidad para definir las condiciones de uso de sus materiales que el *copyright* clásico, reservándose aquellos derechos que deseen (por ejemplo, permitiendo la reproducción, pero prohibiendo el uso comercial o la obra derivada). Las licencias *Creative Commons* han recibido críticas desde la FSF porque no aseguran la libertad de los usuarios.

Hay otros muchos tipos de licencias (X, Apache, Mozilla Public License, etc.), pero las citadas son sin duda los más populares. Una lista actualizada, con enlaces a explicaciones detalladas y copias *verbatim*, puede obtenerse en la Wikipedia, buscando "*List of software licenses*".

Pero el software libre no es sólamente un tema legal, es el fruto de una larga tradición de colectivos de "hackers" informáticos (en el sentido adecuado del término, no en el que los

medios de masas lo han utilizado inadecuadamente para referirse a “crackers”, personas que irrumpen sin autorización en ordenadores). Para entender las ideas que hay tras el software libre es necesario conocer algunos hechos relevantes de la historia de la informática que nos mostrarán que, en informática, no siempre ha sido todo como es hoy y que han contribuido a desarrollar el pensamiento de numerosos programadores.

5. El origen del software libre

Una de las mejores historias del software libre en castellano es, sin duda, la incluida en González Barahona, Seoane y Robles (2003). En inglés, puede consultarse Bretthauer (2001), DiBona et al. (1999) o Wayner (2000). Para una biografía de Richard Stallman, el principal ideólogo del movimiento, véase Williams (2002). En este apartado resumiremos los hechos e ideas esenciales para comprender el origen y la evolución del software libre hasta nuestros días.

En realidad, los orígenes del software libre son los del propio software: en los años sesenta, cuando los ordenadores eran máquinas enormes y costosas, el software era libre. Era considerado un complemento necesario para que funcionara el ordenador y algo que escribían los programadores de universidades, centros de investigación, oficinas gubernamentales, etc., que compartían entre ellos sin ningún problema y que modificaban tranquilamente. A nadie se le ocurría decir que algo “era suyo”: lo hacían un poco entre todos. En ningún caso se trataba como un producto con valor de mercado, por el que hubiera que pagar, sino información que libremente compartían sus usuarios y que éstos mejoraban de manera acumulativa, beneficiando a todos. La cultura “hacker” de los laboratorios de informática universitarios estadounidenses (Stanford, Berkeley, Carnegie Mellon, MIT, etc.), creada en los 60 y los 70 paralelamente del movimiento contracultural californiano, es el caldo de cultivo del software libre.

No fue hasta principio de los años setenta cuando IBM comenzó a “vender” separadamente sus máquinas y sus aplicaciones. Para protegerlas de posibles modificaciones, se comenzó a restringir la distribución del código fuente. De este modo, si los usuarios únicamente disponían del código máquina, las modificaciones eran prácticamente imposibles. La política de IBM se generalizó a medida que crecía la industria informática. A pesar de ello siguió existiendo el software libre. El caso más destacado es el desarrollo del sistema operativo Unix, que rápidamente se extendió por universidades y centros de investigación. Aunque la relación entre empresas y universidades en el desarrollo de Unix se vio comprometida por pleitos y querellas cuando las empresas se dieron cuenta que había beneficios a ganar. Este episodio, protagonizado por AT&T y la Universidad de Berkeley, dio lugar a la aparición de distintas versiones de Unix, unas libres, otras de pago, y a una fragmentación del mercado y una inseguridad jurídica de la que todo el mundo salió perdiendo.

La popularización de los ordenadores personales en los ochenta acabó cambiando las reglas de juego de la industria informática, especialmente tras la decisión de IBM de construir un ordenador con piezas comunes, que se podían adquirir en cualquier tienda, y encargar el sistema operativo a una pequeña empresa de software llamada Microsoft. IBM rompió el modelo de negocio informático: el elemento clave del sistema ya no fue el hardware, sino el software. Muchas empresas fabricaban PCs "compatibles" con las especificaciones de IBM (los "clónicos"), pero sólo una poseía los secretos y la propiedad del sistema operativo. Esta decisión, junto a otros factores, provocó un vuelco en la industria de tales consecuencias que a la larga echó fuera del juego a la propia IBM: todo el mundo podía construir PCs, incluso la industria de países con mano de obra barata, y cayeron los precios permitiendo el acceso al PC a muchas más personas. La decisión de IBM puso el sector en manos de los fabricantes de software y, como se constató más tarde, creó las condiciones para la aparición de un monopolio basado en el sistema operativo del ordenador y en las aplicaciones que mejor se ejecutasen en este entorno.

El año 1984, Richard Stallman, un brillante programador de ordenadores del Laboratorio de Inteligencia Artificial del MIT (Instituto de Tecnología de Massachussets) abandonó su trabajo, harto de la política de las empresas de software. Se cuenta que la gota que colmó el vaso fue la negativa de una empresa a proporcionarle acceso al código fuente de un programa que controlaba una impresora. Stallman quería repararlo porque funcionaba mal. En esa época ocurrieron dos hechos, menos anecdóticos, que condujeron a Stallman a la decisión de abandonar el MIT y crear software libre. El sistema operativo que había contribuido a crear con sus colegas se convirtió en trabajo inútil tras la compra de un nuevo ordenador por parte del laboratorio. La nueva máquina usaría un sistema privativo. Por la misma época, la mayoría de los "hackers" que trabajaban con Stallman abandonaron la universidad para formar una empresa privada. El paraíso "hacker" en el que vivía Stallman se estaba desmoronando ante sus ojos (Williams, 2002). A principios de 1984, Stallman dejó su trabajo en el MIT a fin de que la institución no interfiriera en sus planes: crear un sistema operativo completo, tipo UNIX. Lo llamó GNU (un acrónimo recursivo, esto es, que se refiere a sí mismo, algo muy del gusto de los programadores de inteligencia artificial, y que significa *GNU No es Unix*) y fundó la *Free Software Foundation* (FSF) para reunir los fondos necesarios. El trabajo sería realizado por voluntarios y el sistema sería libre: se podría compartir, modificar y distribuir libremente las mejoras. Eso si, toda aplicación que se realizara utilizando las herramientas o el código fuente GNU debería otorgar a sus usuarios los mismos derechos. Es decir, Stallman, usando las leyes de *copyright,* creó una licencia que aseguraba que su trabajo y el de otros muchos voluntarios no podría ser utilizado nunca para "cerrar las puertas" a los usuarios finales, que nadie podría apropiárselo y negar esos mismos derechos a sus usuarios. Ese mismo año, Stallman escribió la licencia GPL (GNU *General Public License*) para asegurar dichas libertades, *The GNU Manifesto* (Stallman, 1984a) y *Why Software Should Not Have Owners (*Stallman*,* 1984b*)*, ensayos seminales que explicaban

sus puntos de vista sobre el software y su intención de crear un sistema tipo UNIX completamente libre.

A principio de la década de los noventa la FSF tenía desarrolladas casi todas las piezas necesarias para que su sistema operativo fuera utilizable... excepto el *kernel* (Stallman, 1999). El *kernel* es el núcleo del sistema operativo de un ordenador. Es la pieza que hace posible la comunicación entre el resto del software y el hardware, una pieza esencial. Pero en esa época ocurren dos hechos que cambiarán radicalmente la historia del software libre. El primero es la popularización de la Internet, anteriormente confinada a las universidades y centros de investigación. Los programadores ya disponen de un medio para comunicarse, coordinarse y colaborar de un modo mucho más rápido, barato y eficiente que mandando cintas de ordenador por correo. El segundo es la aparición en escena de un joven estudiante finlandés de 21 años llamado Linus Torvalds, que, para aprender cómo funcionaba, había decidido crear un *kernel* para ordenadores personales por sí mismo. La idea, sorprendentemente cuaja y -con la ayuda de numerosos voluntarios, que se comunicaban por Internet- se desarrolla un *kernel*, llamado Linux. De la unión entre las herramientas creadas por el proyecto GNU y Linux, nació el sistema GNU/Linux con licencia GPL. Era rápido, eficiente y, lo que es más importante, se podía modificar y compartir.

La aparición de la *Open Source Initiative* (OSI) en 1998 marca un punto de inflexión en la historia del software libre. Un grupo de programadores, liderados por Eric Raymond, Bruce Perens, Tim O'Reilly (un editor de libros técnicos), hartos del lenguaje mesiánico y revolucionario de Stallman, que asustaba a los empresarios y a los medios de comunicación, de la confusión entre “libre” y “gratuito” y, sobre todo, de la personalidad de Stallman, decidieron adoptar el término “código abierto” para referirse a lo mismo que Stallman llamaba “software libre”. Raymond diseñó una cuidadosa estrategia de comunicación que triunfó cuando Netscape, una empresa puntera en tecnología web, pero con graves problemas debido a la decisión de Microsoft de regalar preinstalado su navegador web, señaló su ensayo “La catedral y el bazar” como la inspiración de su decisión de liberar el código de Netscape. Raymond y Perens crearon la OSI para certificar que las licencias del software se ajustaban a la definición de “software de código abierto” elaborada por Perens en base a la licencia del proyecto Debian, una de las más famosas distribuciones GNU/Linux. A nivel de ideas, la OSI defiende la superioridad técnica, no moral, del software libre, producto del nuevo modelo de desarrollo creado por Linus Torvalds y la comunidad Linux y que Raymond había descrito en la metáfora de “La catedral y el bazar”. La “catedral” es Stallman y el proyecto GNU, incapaz de crear durante una serie de años un *kernel* utilizable, encerrado en su torre de marfil. El “bazar” es Linux, una comunidad de iguales, gestionada de manera abierta, descentralizada y poco jerárquica, a la que cualquiera puede contribuir con su código, sus propuestas, informes de errores, escribiendo documentación, localizando (traduciendo) software, etc. Numerosas empresas han prestado mucha atención a esta nueva

manera de producir software y a los modelos de negocio ligados al código abierto. Los medios de comunicación encumbraron rápidamente a la nueva generación de “hackers”, bastante menos ideologizados y mucho más mediáticos que los barbudos radicales de la FSF. Surgieron nuevas empresas dedicadas a comercializar “distribuciones” GNU/Linux y diversos fabricantes de software propietario crearon versiones de sus productos para dicho sistema y varios fabricantes de hardware soportaron Linux en sus ordenadores (sobre todo en servidores) a la vista de cómo crecía su uso, especialmente en servidores. Red Hat, Oracle, IBM, Sun, Intel, Apple y otras muchas empresas trabajan *con* software de código abierto. Prestan y cobran por servicios, elaboran y venden distribuciones o compaginan en sus productos código abierto y código privativo, revertiendo en la comunidad sus desarrollos sobre el código abierto. Mientras tanto Raymond “dispara” contra la pieza clave que sustenta el movimiento del software libre, la licencia GPL, afirmando en una entrevista en una de las revistas de Tim O'Reilly que “ya no la necesitamos más” (Biancuzzi, 2005). En los últimos tiempos, la OSI parece bastante adormecida. Raymond dejó la presidencia y el nuevo ejecutivo duró un mes en el cargo. Mientras tanto, el impacto de la OSI y las ideas de Raymond y sus colegas en los medios de comunicación y en las empresas tecnológicas ha sido enorme.

En la actualidad GNU/Linux es un sistema operativo en franca expansión. Lidera segmentos de mercado como los servidores Internet. Los “expertos” utilizan GNU/Linux por su estabilidad, seguridad, eficacia y eficiencia, escalabilidad, ritmo muy alto de innovaciones y mejoras, soporte de la comunidad de usuarios, cantidad de aplicaciones disponibles, facilidad de integración en cualquier tipo de entorno, coste total de propiedad, etc. Todas estas ventajas son un efecto, sobre todo, del hecho de que los programadores pueden acceder al código, mejorarlo y distribuir los cambios. En la actualidad GNU/Linux se está extendiendo entre los sistemas de escritorio, esto es, en los ordenadores personales que utilizamos los “usuarios finales” para realizar nuestro trabajo cotidiano: escribir, calcular, navegar por Internet, etc. La mejora constante en la facilidad de uso de los entornos de ventanas (los dos más utilizados son GNOME y KDE), el desarrollo de distribuciones fácilmente instalables e incluso ejecutables desde un CD o DVD y la aparición de la segunda versión de una *suite* ofimática potente y fácil de usar como OpenOffice.org (la primera es del 2002) ha convencido a muchas personas de que el software libre es una opción técnicamente viable y económicamente ventajosa.

El sector educativo “mira” atentamente el software libre como una alternativa real al software privativo, aunque sin decidirse totalmente. Riina Vuorikari, en un Informe Especial para la *European Schoolne*t (Vuorikari, 2004) ha destacado el escaso partido que las autoridades educativas nacionales europeas han sacado del potencial del software libre. Las razones, afirma, son diversas. Por una parte, la falta de conocimiento sobre el desarrollo del software libre dificulta su extensión en el sector educativo. Por otra, el hecho de que el debate sobre su potencial esté conducido por activistas y “lobbistas”, que exageran sus ventajas e inconvenientes, ha dejado

poco espacio para análisis equilibrados y estudios rigurosos. Incluso algo tan sencillo de evaluar para las autoridades educativas como el "coste total de propiedad" del software libre y del privativo no ha sido estudiado hasta hace muy poco tiempo (BECTA, 2006), para comprobar que el software libre es más barato considerando todos los conceptos: adquisición, mantenimiento, instalación, soporte, formación, etc. A pesar de la indiferencia denunciada por Vuorikari en el año 2004, el paso ya lo han dado algunas administraciones educativas. Por ejemplo, en España las autoridades educativas de Extremadura, Andalucía, Valencia, Madrid, Castilla-La Mancha, y Galicia han desarrollado sus propias distribuciones, gnuLinEx, Guadalinex, Lliurex, Max, MoLinux y Trisquel, respectivamente, están dotando a todos sus centros educativos de ordenadores con GNU/Linux preinstalado y creando servicios y software libre para cubrir sus necesidades. En países como Brasil (Kim, 2005) o Venezuela ha habido iniciativas legislativas a favor del software libre. Pero... ¿cuáles son las razones que justifican estas decisiones? ¿Por qué debemos usar software libre en educación?

6. El software libre en educación

Las razones por las que los partidarios del software libre y de código abierto defienden su superioridad sobre el privativo se pueden dividir en dos tipos esenciales, que se corresponden con los discursos de las dos principales corrientes de pensamiento que hemos descrito anteriormente. Mientras la OSI (*Open Source* Initiative), esto es, Eric Raymond, Tim O'Reilly, Bruce Perens y otros conocidos desarrolladores como Linus Torvalds defienden el código abierto por razones pragmáticas, como su mayor calidad, su menor coste, mayor seguridad, estabilidad, eficiencia, integración, etc., la FSF (*Free Software Foundation*), esto es, Richard Stallman y otros conocidos desarrolladores y activistas, defienden su superioridad ética, social y política. En realidad, ambos conjuntos de argumentos son complementarios.

Algunos autores, conforme se han ido extendiendo las ideas sobre el software libre, han apuntado también argumentos que tienen en cuenta el contexto escolar, es decir, por qué es más adecuado o ventajoso usar software libre en educación, además de las razones genéricas que se aplican a cualquier usuario informático. Entre ellos merecen mención aparte los relacionados con la enseñanza de la informática y la ingeniería del software a nivel universitario, que tiene en el software libre un fenomenal recurso didáctico. Pero vayamos por partes.

6.1. Ventajas pragmáticas

Las ventajas de índole práctica del software libre derivan de su modo de producción: redes distribuidas de iguales que colaboran voluntariamente por una amplia gama de motivaciones (Hars y Ou, 2001; III-UM and Berlecom Research, 2002). Raymond (2004) lo explica

de manera simple: “los programadores de código abierto han aprendido que el secreto es enemigo de la calidad. La manera más efectiva de conseguir fiabilidad en el software es publicar el código para que lo revisen otros programadores y no-programadores expertos en el área de aplicación del software”. Raymond lo resume en una frase, la Ley de Linus (Torvalds): “si suficientes globos oculares miran el código, los errores serán evidentes”. Puede parecer increíble, pero en el desarrollo de Linux han participado, en mayor o menos medida, más de 750.000 programadores de todo el mundo (Raymond, 2004). No hay empresa que pueda competir con eso. La Internet y GNU/Linux son dos ejemplos notables de que el modelo de código abierto y libre colaboración entre programadores funciona.

Por tanto, para el movimiento del código abierto, el software desarrollado siguiendo el modo de producción colaborativa entre iguales es de mayor calidad, ofrece mayor seguridad, más estabilidad a lo largo del tiempo (el código no desaparece si una empresa cierra o es comprada por otra), los tiempos de desarrollo son menores y los proyectos interesantes crecen a una enorme velocidad (al ser la programación una tarea altamente paralelizable, especialmente la parte más tediosa: la detección de errores) y tiene un precio sencillamente inigualable. Es más, el concepto de software como servicio y no como producto favorece a la industria local en lugar de contribuir a la creación de monopolios y beneficia a más programadores que el modelo privativo. El software de código abierto es más fácilmente “localizable”, un tema de la mayor importancia en educación, sobre todo para lenguas minoritarias o minorizadas (Mas, 2003), es más accesible y desarrollar software libre la mejor manera de usar los fondos públicos para potenciar la industria de l software local, en lugar de pagar *royalties* por el software privativo, que van a parar a grandes empresas extranjeras.

Uno de los informes más completos sobre las ventajas *cuantitativamente demostrables* del software libre es el de Wheleer (2005) en el que se recogen y valoran estudios empíricos que llevan a afirmar a su autor que “en muchos casos, utilizar software de código abierto/software libre es un enfoque razonable o incluso superior a utilizar su competidor privativo de acuerdo con diversas medidas”. Wheleer (2005) analiza aspectos como la cuota por segmentos de mercado, la fiabilidad, el rendimiento, la escalabilidad, la seguridad y el coste total de propiedad, concluyendo que:

> “El software OSS/FS tiene una cuota de mercado significativa en muchos mercados, es a menudo el software más fiable y, en muchos casos, tiene el mejor rendimiento. El software OSS/FS escala, tanto en tamaño del problema como del proyecto. El software OSS/FS a menudo es mucho más seguro, quizá debido a la posibilidad de que todo el mundo lo revise. El coste total de propiedad es a menudo mucho menor que el software privativo, especialmente cuando el número de plataformas aumenta. Todas estas afirmaciones no son meras opiniones; estos efectos pueden ser demostrados cuantitativamente, utilizando diversas medidas. Este argumento no considera otros temas difíciles de medir, tales como la libertad frente al control de una única fuente, libertad de la gestión de licencias (con el riesgo añadido de auditorías y juicios)” (Wheleer, 2005, pág. 124).

Pero no todo son alabanzas. Michelle Levesque (2004) ha criticado algunas actitudes de la cultura

del software libre: la falta de interés de los programadores en el diseño de interfaces sencillas e intuitivas, la poca documentación existente en ocasiones, el exceso de funcionalidades, el hecho de programar únicamente para usuarios avanzados y despreciar lo que se puede aprender del software privativo (por ejemplo, la facilidad de uso). Es posible que en el pasado las críticas de Levesque tuvieran más sentido que ahora. En los últimos años el software libre ha experimentado un salto cuántico y los sistemas de escritorio, como Ubuntu (una distribución GNU/Linux muy completa e intuitiva), OpenOffice.org (suite ofimática), Firefox (navegador web), Thunderbird (correo electrónico), GIMP (dibujo) y, en general, todo el software listado en la Tabla 2, son sumamente intuitivos y fáciles de manejar, rápidos, estables y seguros. Para convencerse no hacen falta muchos estudios: basta probarlos.

6.2. Razones políticas, éticas y sociales

Pero hay más razones, además de las prácticas o técnicas para escoger software libre frente a privativo. La mayor parte de las personas cree que el software *vive* en el interior de los ordenadores y que su relación con él empieza cuando encienden su ordenador y termina cuando lo apagan. Es una idea tranquilizadora: nos da una falsa sensación de control, de *estar al mando*. Basta apagar el ordenador para que el software se *duerma* hasta que lo volvamos a necesitar. ¿Qué mal puede hacernos? Lo único que les parece importante es si funciona bien o no, si cumple su misión. Reflexionemos un momento sobre la importancia del software en nuestra sociedad. Aunque no hayamos usado nunca un ordenador, nuestra vida entera depende del software: ¿cómo se calcula nuestro salario?, ¿cómo lo cobramos?, ¿dónde está nuestro dinero en el banco?, ¿qué pasa con los cajeros automáticos cuando no funciona la red informática que los une al banco?, ¿y la información sobre nuestra salud?, ¿cómo se gestiona la red eléctrica de nuestra ciudad?, ¿y la de gas o agua?, ¿qué pasa cuando en el supermercado un producto no tiene el código de barras?, ¿quién controla el encendido y los frenos ABS de nuestro coche?, ¿cómo calculan las autoridades los impuestos que debemos pagar?, ¿dónde están anotadas las calificaciones de nuestras asignaturas en la universidad?

Efectivamente: en ordenadores... y todos ellos funcionan con software. Dicho software lo produce alguien. Parte de él es privativo, es decir, nadie, excepto la empresa que lo produce, puede ver cómo está hecho por dentro. Y el objetivo de la empresa es ganar dinero. Por eso no es extraño que intenten fidelizar a sus clientes, por ejemplo, guardando la información en ficheros informáticos con formatos secretos, protegidos por patentes, que ningún otro programa pueda leer, ocultando información sobre sus sistemas para que la competencia no pueda desarrollar productos que funcionen bien o patentando ideas (o algoritmos) para que nadie pueda utilizarlas, frenando la innovación, impidiendo la competencia en el mercado y borrando del mapa a las empresas pequeñas que no tienen dinero para pleitear durante años. También sabemos que periódicamente tendremos que renovar nuestros ordenadores porque el nuevo software no

funcionará en los que tenemos, aunque estén en perfecto estado, y el software que utilizábamos dejaría de actualizarse y de tener soporte de sus fabricantes. Es más, sabemos que nuestros ordenadores, al inicializarse, si están conectados a Internet, envían información sobre nosotros y nuestro software al fabricante del sistema operativo... pero no sabemos muy bien qué información envían ni qué hace con ella dicha empresa. Un gobierno, por motivos de seguridad, no puede usar software que no sabe perfectamente qué hace: podría estar enviado información delicada o comprometedora a otros gobiernos. O una empresa podría estar siendo espiada por sus competidoras, ayudadas por el creador del sistema operativo.

No hace falta seguir: es evidente que las tecnologías de la información y el software que hace que funcionen o no de determinada manera son demasiado importantes en nuestras vidas para que no sepamos qué hacen realmente o para que se comporte de manera que nos aten de por vida a intereses comerciales. El software conforma la estructura de la comunicación y la información en una sociedad post-industrial cuyo mayor factor de producción es el conocimiento, define cómo podemos trabajar, comunicarnos, divertirnos o relacionarnos con nuestros vecinos o parientes. Algunos autores han sugerido que las nuevas tecnologías están conformando nuevos tipos de procesos cognitivos en los jóvenes, una nueva manera de procesar la información determinada por su extensa práctica con nuevos tipos de medios, lenguajes y modelos de comunicación (el hipertexto y el multimedia, la interactividad de los videojuegos, la instantaneidad de los teléfonos móviles y la Internet, etc.). El cambio es de tal magnitud que se habla de “nativos e inmigrantes digitales” (Prensky, 2001).

Por otra parte, el software libre promueve la cooperación entre las personas donde el software privativo la convierte en un delito. Y la cooperación es un valor fundamental de nuestra sociedad al que la escuela debe prestar especial atención.

6.3. El software libre en educación

Richard Stallman (2003) ha escrito un texto sobre las razones por las que las escuelas deberían utilizar exclusivamente software libre. El software libre, recuerda Stallman, permite que los usuarios controlen lo que hacen sus ordenadores y cooperen entre ellos. Las dos razones son también válidas para la educación Pero hay razones netamente “educativas”.

1. La primera es que el software libre se puede copiar y redistribuir a precio de coste. La Administración educativa puede dotar de software a todos sus centros docentes a muy bajo precio y dedicar los recursos ahorrados a otros temas necesarios para la educación: más ordenadores, formación del profesorado, desarrollo de software libre educativo, etc. En los países menos desarrollados, el software libre puede ayudar a dotar de infraestructura tecnológica a sus escuelas y a paliar la “brecha digital” con el mundo

desarrollado. Los vendedores de software privativo, que saben de la importancia de la educación para sus futuras ventas, pueden ofrecer software a muy bajo coste o gratuito a las escuelas. Pero se trata en realidad de una estrategia comercial para captar futuros clientes y para formarlo en sus productos a costa del erario público. Es una simple trampa.

2. La escuela ha de enseñar a los estudiantes valores y estilos de vida que beneficien a toda la sociedad. La escuela ha de promover el uso de software libre por la misma razón que promueve el reciclaje: porque nos beneficia a todos. Si los estudiantes usan el software libre y aprenden que es mejor que el privativo, cuando sean adultos seguirán usando el software libre. Eso permitirá a la sociedad liberarse de los abusos y del control de las multinacionales que controlan el software privativo.

3. El software libre favorece que los estudiantes aprendan cómo funcionan los ordenadores y el propio software. Los futuros programadores se inician en la programación durante la adolescencia. Es una etapa clave en la que necesitan buenos modelos y ejemplos para modificar, copiar y "jugar" con ellos. Necesitan desafíos. El software libre, al permitir el acceso al código fuente del programa, les facilita enormemente el aprendizaje. El software privativo es una "caja negra" que no aporta nada para satisfacer su curiosidad y sus ansias de saber. El mensaje que les envía el software privativo es "el conocimiento es una mercancía, lo que quieres saber es un secreto comercial, aprender está prohibido por la ley". El software privativo mantiene a la gente alejada del conocimiento, sacraliza la tecnología y contribuye interesadamente a la ignorancia tecnológica que tan buenos resultados económicos les proporciona a las empresas que lo comercializan.

4. Pero, aunque muchos adolescentes no sientan curiosidad por cómo están hechos los programas de ordenador, hay valores generales que persigue la educación que están en claro conflicto con el mensaje que transmite el software privativo. Las escuelas deben enseñar hechos, conceptos, principios y procedimientos, pero también valores. La misión de la escuela es enseñar a las personas a ser buenos ciudadanos, a cooperar con los demás, a ser solidarios. Esta es la base de la sociedad. En informática, cooperar significa, entre otras cosas, compartir software, poder hacer copias a todos los compañeros de clase, llevarse a casa el software que se usa en la escuela. Y todo eso, con el software privativo es un delito.

5. Finalmente, enseñar a los estudiantes a usar software libre y a participar en la comunidad de usuarios/desarrolladores de software libre es una lección cívica llevada a la práctica. También enseña a los estudiantes que el ideal es el modelo de servicio público y la solidaridad, no el modelo del beneficio a cualquier precio de los magnates. Todos los niveles pueden y deben usar software libre (Stallman, 2003).

Amatriain (2004, pág. 5) resume perfectamente la coincidencia en valores del software libre y la educación: “los valores que una institución educativa tendría que promover están muy relacionados con aquellos que promueve el software libre: libertad de pensamiento y expresión, igualdad de oportunidades, esfuerzo y beneficio colectivo en lugar del beneficio individual, etc. De hecho, la libertad puede que sea el valor más importante relacionado con la educación: la educación sin libertad se convierte en adoctrinamiento”.

6.4. El software libre en la enseñanza de la informática

El caso de la enseñanza de la informática a nivel universitario es especial. En primer lugar, el software libre permite ver y analizar cómo están diseñados y funcionan programas de ordenador de primerísimo nivel. En segundo lugar, algunas de las mejores herramientas software son libres y los estudiantes pueden utilizarlas sin coste alguno. Pero más allá de estudiar y usar software de código abierto, los estudiantes pueden participar activamente en proyectos reales de desarrollo (Shockey y Cabrera, 2005). Los proyectos proporcionan un contexto más amplio que las típicas tareas académicas en pequeño grupo y les permiten comprender las relaciones entre desarrolladores y comunidad de usuarios, practicar habilidades comunicativas, trabajar en equipo con materiales, ideas y líneas de trabajo establecidas, explorar posibilidades y soluciones nuevas, etc. Es decir, los proyectos libres (y la facilidad para contribuir a ellos) proporcionan un contexto real de trabajo y un valioso entorno de programadores profesionales y altamente cualificados. Otros autores (por ejemplo, Farber, 2002) han sugerido utilizar el proceso de desarrollo de software libre como modelo para diseñar procesos de enseñanza/aprendizaje formales. Es decir, intentar reproducir el modelo de un entorno distribuido de construcción colaborativa de artefactos en el aula presencial. Sin embargo, como principio de dicho modelo, Faber utiliza una serie de consejos de Raymond (1999) a quienes aspiran a desarrollar software de código abierto, extraídos de su ensayo *La catedral y el bazar* sobre el desarrollo de Linux y sus propias experiencias como desarrollador. La correspondencia entre los consejos de Raymond (de los que Faber elije el subconjunto más “aprovechable” pedagógicamente) y los aspectos mínimos necesarios de un modelo educativo es, cuando menos, tenue. Los desarrolladores de software libre, las comunidades que se forman de manera más o menos espontánea alrededor de proyectos de software libre ejemplifican, sin duda alguna, procesos interesantes desde el punto de vista educativo. “Es hora de que las instituciones de educación superior tomen en consideración este importante y nuevo método de producción y aprendizaje seriamente (el código abierto), y adopten muchos de sus métodos” (Staring, Titlestad y Gailis, 2005).

Bryan Pfaffenberger (2000) ha elaborado un argumento en defensa del uso del software libre en la alfabetización informática que merece análisis. Su tesis es que el software libre en general -y Linux en particular- son más adecuados para preparar a los estudiantes para un

mundo en rápido cambio tecnológico que el software propietario. La alfabetización informática suele estar centrada en productos comerciales y adopta el enfoque “Qué tecla hay que apretar”. Es decir, está basada en procedimientos y destrezas concretos y de corto alcance. La justificación de este enfoque no es pedagógica, sino pragmática... y errónea. Se afirma que el entorno de sistema y aplicaciones privativas dominantes es lo que los estudiantes encontrarán en el mundo del trabajo, cuando acaben sus estudios. Es lo que los empresarios demandan y, por tanto, es lo que hay que enseñarles. Sin embargo, el enfoque “Qué tecla hay que apretar” olvida el rápido desarrollo de las tecnologías de la información: posiblemente la versión que “dominan” a la perfección los estudiantes ya no exista cuando busquen empleo. Lo que los empresarios necesitan no es alguien que domine versiones viejas del software, sino alguien capaz de aprender de manera rápida cualquier aplicación informática, alguien que posea también conocimientos y competencias generales. Esta crítica, quizá no sea tanto al software privativo como a los múltiples “paquetes formativos” y programas de certificación desarrollados por las empresas y adoptados por las instituciones educativas. El software libre, según Pfaffenberger, por su apertura y flexibilidad, facilitaría la formación basada en competencias genéricas, transferibles a otras situaciones y entornos, y el desarrollo de la capacidad de seguir aprendiendo por su cuenta a lo largo de toda la vida de los estudiantes. A nuestro juicio, se trata más de una cuestión de enfoque didáctico que de la naturaleza del software: se puede formar de la misma manera estrecha estilo “Qué tecla hay que apretar” con software libre, aunque Pfaffenberger acierta plenamente en su crítica indirecta a los programas de certificación, manuales o completos “paquetes formativos” diseñados por la empresas para enseñar a manejar su software. Los fines que persiguen las empresas seguramente no son los mismos que los fines de los centros educativos. La alfabetización tecnológica es bastante más que saber manejar una *suite* ofimática.

6.5. El software libre y la innovación en tecnología educativa

Graham Atwell (2005) ha puesto de manifiesto un hecho diferencial del software libre en la educación que no podemos dejar de señalar: su maridaje con la innovación educativa. Las razones son diversas En primer lugar, en los proyectos de software libre el coste inicial es muy bajo: suelen ser personales o de un pequeño grupo de entusiastas. En segundo lugar, se puede “construir” sobre el trabajo de otros proyectos y explorar sus aplicaciones educativas (por ejemplo, integrando herramientas que originalmente no fueron diseñadas con propósito educativo, como blogs y wikis). Si el proyecto cuaja, porque la gente lo encuentra de interés, es fácil abrirlo a la colaboración. Un ejemplo de este proceso es Moodle, una plataforma de enseñanza basada en presupuestos socio-constructivistas del aprendizaje que ha sobrepasado en funcionalidades e implantación a sus alternativas privativas y que se ha hecho tremendamente popular en el último año. Iniciado por una sola persona, Martin Dougiamas, que, descontento por cómo estaba diseñado y funcionaba el software privativo equivalente de su universidad, “se hizo” una

plataforma (realmente modesta en sus inicios) para sus clases. Hoy, la comunidad Moodle está formada por decenas de desarrolladores, miles de usuarios, sus instalaciones se cuentan por millares y varios millones de estudiantes y profesores utilizan Moodle en sus clases presenciales, semi-presenciales o a distancia.

Una tercera razón reside en el efecto de unir en una comunidad en pos de un objetivo común a informáticos y especialistas en otros campos. La comunidad Moodle está formada por informáticos profesionales, profesores de informática, educadores de diferentes niveles educativos, especialistas en tecnología educativa y en *e-learning*, etc. El proceso por el que se proponen, discuten, diseñan, desarrollan, prueban, modifican, vuelven a probar, rediseñan, perfeccionan y adoptan nuevas funcionalidades es un modelo típico de desarrollo de software de código abierto. En el proceso, tanto los programadores como los educadores proponen, argumentan, programan, prueban, critican, etc. y, mientras tanto, aprenden unos de otros. La comunidad de usuarios/desarrolladores es, sin duda alguna, lo que ha convertido a Moodle en un sistema puntero desde el punto de vista didáctico y tecnológico, líder mundial en número de instalaciones, que van desde desde universidades gigantescas (la *Open University*, por ejemplo, con cerca de 120.000 estudiantes distribuidos por todo el mundo), pasando por numerosas universidades presenciales de tamaño medio o pequeñas (como la de los autores), hasta escuelas rurales minúsculas en países de los cinco continentes. Moodle ha sido traducido por voluntarios a más de 70 lenguas, incluyendo algunas sumamente minoritarias, para las que la probabilidad de que una gran empresa de software “localice” y traduzca a su lengua un producto comercial de estas características es exactamente “ninguna.” La razón: no hay dinero a ganar. Muchos proyectos de código abierto poseen este tipo de comunidades mixtas en las que desarrolladores informáticos y especialistas en el área de aplicación unen sus conocimientos para crear un producto adaptado a las necesidades reales de los usuarios. Estas comunidades sirven como espacios naturales de intercambio de ideas, de debate y reflexión, de formación mutua en el “otro” campo del conocimiento y en el “propio”. Son un lugar excelente para aprender.

7. Los formatos libres

Un tema del que no somos demasiado conscientes cuando usamos ordenadors es el efecto de utilizar y difundir ficheros en formatos privativos, cerrados y sujetos a patentes, por más estándares *de facto* que creamos que son. Cuando enviamos un fichero por correo electrónico a un amigo o compañero de trabajo tenemos dos opciones. Si se lo enviamos en un formato privativo le estamos obligando a que utilice -y si no lo tiene, a que compre o consiga de otra forma- la aplicación privativa y única que lee dicho formato. De este modo contribuimos a crear estándares *de facto,* lo cual implica extender los monopolios de software y la dependencia absoluta de un único fabricante para acceder a *nuestra* documentación.

La segunda posibilidad es usar un formato libre, a ser posible estandarizado, que puedan utilizar distintos programas. De esta manera permitimos a nuestro compañero o amigo usar el software que prefiera, libre o privativo, para abrir el fichero y acceder a la información. Si, como docentes, pedimos a nuestros estudiantes que utilicen formatos privativos en sus trabajos y asignaciones les estamos obligando a comprar determinado software o a usarlo indebidamente.

El problema es que, hasta hace poco, en áreas muy importantes, como la ofimática, no existían formatos de interoperabilidad estándar. Afortunadamente, hoy el formato OpenDocument (ODF), creado por el consorcio OASIS es ya el estándar ISO/IEC 26300 y podemos utilizar aplicaciones diversas para trabajar con documentos de texto, hojas de cálculo, gráficos y presentaciones e intercambiar documentos en formato estandarizado sin demasiados problemas.

Un área en la que los formatos son sumamente importantes es la Internet. Si nuestras páginas web, por ejemplo, sólo se pueden ver adecuadamente con cierto navegador, estamos obligando a las personas que quieran verlas a usar dicho software. Si usamos extensiones propietarias, aunque sean gratuitas, estamos obligando a los demás a instalar software privativo y cerrado para ver nuestros contenidos multimedia. La solución es simple: utilizar siempre los estándares aprobados por la W3C (*World Wide Web Consortium*). Por ejemplo, usar HTML estándar con el máximo nivel de accesibilidad, no usar extensiones privativas, ni incluir ficheros en formatos privativos. Así conseguiremos que todo el mundo pueda ver bien nuestras páginas web sin tener que usar el software que nosotros decidamos. También de este modo evitaremos el triunfo de las estrategias *Embrace, extend and extinguish* de algunas empresas, que consisten en comenzar utilizando un estándar abierto, añadirle extensiones privativas para crear un formato expandido, “compatible” con el estándar oficial, en nombre de una supuesta innovación, para luego convertirlo en el estándar *de facto* y acabar con la competencia. Otras estrategias conocidas son integrar el software en el sistema operativo de tal manera que sea imposible sustituirlo por alternativas libres o no facilitar información vital sobre el sistema a la competencia para que su software funcione peor que el propio). Si esto ocurriera, si un fabricante convirtiera en imprescindible su software para usar la red, la Internet estaría completamente en sus manos (Wikipedia, 2006). Lo mismo que hemos argumentado en relación a los formatos se puede aplicar a los protocolos de comunicación en la red. El caos actual de la mensajería instantánea, con sistemas incompatibles entre sí, ligados a fabricantes, es una demostración palpable de por qué la sociedad necesita estándares abiertos, no los creados por una empresa para obtener una ventaja competitiva sobre sus rivales. Lo que es bueno para una empresa, es malo para el conjunto de los usuarios. Queremos una única Internet, en la que quepamos todos, no tantas redes como empresas de software. Y no la queremos dominada por un monopolio transnacional del software o por un país.

8. Para terminar...

Las tecnologías de la información y la comunicación, los ordenadores, la Internet... son cada día más importantes en nuestras vidas. Toda nuestra economía utiliza intensivamente dichas tecnologías. El ocio y el tiempo libre, nuestros hogares, nuestro aprendizaje, la comunicación con otras personas, etc., muchas actividades de nuestra vida cotidiana dependen en mayor medida de lo que parece de las nuevas tecnologías y de los programas que las hacen funcionar. Nuestra sociedad está siendo conformada por dichas tecnologías y el software es lo que las hace funcionar. Lo que nos enseña el movimiento por el software libre es que no podemos dejar dichas tecnologías al albur de intereses comerciales, que es necesario el control social en un momento en el que la legalidad está siendo moldeada por la presión, y el dinero, de grupos de interés que buscan asegurar su posición privilegiada (por ejemplo, mediante las patentes de software), siquiera para que sea un terreno de juego justo para los propios intereses comerciales, cuanto más, pues, para un uso democrático y social de la tecnología. El software libre nos proporciona un marco de reflexión sobre las contradicciones que afloran en el advenimiento de la sociedad de la información entre los intereses privados y el bien común, sobre los valores que deben presidir el desarrollo y uso democrático de las tecnologías de la información y es un ejemplo de pautas de acción para conseguir que triunfen dichos valores en otros ámbitos (por ejemplo, en la educación como servicio público). Además, los defensores del código abierto han explicado por qué las redes descentralizadas de iguales son superiores a los modelos centralizados y jerárquicos en el diseño y desarrollo de artefactos complejos, como las aplicaciones informáticas, es decir, por qué el software libre sencillamente es mejor que el privativo. Los negocios basados en el software de código abierto han demostrado que es posible obtener beneficios económicos con el software libre y que potenciando los servicios asociados al software libre se apoya a la industria local y nacional.

Hemos resumido algunas de las razones por las que es conveniente utilizar software libre en educación. Pero, más allá del software libre como producto, si nos fijamos en cómo se desarrolla y en las prácticas y valores de las comunidades de programadores libres, podemos imaginar perfectamente su aplicación a la creación de contenidos y materiales didácticos libres, a la producción de software educativo adaptado al curriculum o a la existencia de comunidades de práctica, formadas por una mezcla de docentes experimentados y novatos, para promover el desarrollo profesional. El éxito de la Wikipedia hace pensar si los docentes no podríamos crear enciclopedias escolares libres y colecciones de actividades didácticas diseñadas por profesores y de eficacia contrastada en el aula. Las barreras son más culturales y estructurales que tecnológicas. El lugar en el que hay que empezar a cambiar de mentalidad es, sin duda, en la formación inicial del profesorado.

Hay elementos muy arraigados de la cultura docente actual (*que impiden este tipo de inciativas)*, el aislamiento físico y psicológico, la colegiabilidad burocrática, la saturación de tareas, la ausencia de apoyo institucional a un modelo de docente no solo "consumidor" de materiales predigeridos sino "creador" de actividades y contenidos, la falta de autonomía curricular, la ausencia de formación en nuevas tecnologías (...). El lugar en el que hay que empezar a trabajar el conjunto de cambios necesarios, sin duda, es en la formación inicial del profesorado. Sin embargo, pese a todas las dificultades, los desarrolladores de software libre y los docentes más innovadores ya nos muestran el camino (Adell, 2006, pág. 10).

El software libre tiene también enemigos. Además de aquellos "naturales", a quienes resta poder, influencia e ingresos, el peor enemigo del software libre son nuestras propias inercias, nuestra escasa disposición a aprender cosas nuevas. Algunos docentes, y futuros docentes, aceptando intelectualmente las ventajas técnicas y sociales del software libre, no están dispuestos a realizar el pequeño esfuerzo que supone aprender a utilizar aplicaciones libres. ¿Cómo podemos pedirles a nuestros alumnos que aprendan si nosotros no estamos dispuestos a hacerlo? Formando a nuestros estudiantes con software libre les enseñamos que el conocimiento es fruto de la libertad, que la ciencia se basa en la cooperación y en la transparencia, les enseñamos a compartir y colaborar con sus compañeros, les ayudamos a ser libres e independientes de ataduras tecnológicas artificiales y les capacitamos mejor para seguir aprendiendo a lo largo de su vida. Pero todo esto únicamente lo podemos enseñar de manera efectiva dándo ejemplo. Una sociedad libre necesita una escuela libre y una escuela libre necesita software libre.

Actividades

El objetivo final de la secuencia de actividades que a continuación proponemos es *el cambio progresivo y sin traumas del software privativo al software libre.*

Cada fase requiere que el usuario se sienta confortable con el entorno informático, sistema y aplicaciones, de la fase anterior y, por tanto, es imposible ofrecer una estimación del tiempo necesario siquiera aproximado: unos usuarios estarán preparados antes que otros, en función de su experiencia previa o de lo mucho o poco que utilicen las aplicaciones libres de su ordenador. Así, a un usuario ocasional, que utiliza esporádicamente el ordenador, puede costarle un poco cambiar a aplicaciones libres. Un usuario habitual, que trabaje diariamente varias horas delante de su ordenador y haya desarrollado competencias genéricas en el uso del software, el paso de una fase a otra puede ser cuestión de días u horas. En algún caso, no notará la diferencia entre la aplicación libre y la privativa a la que sustituye. En otros, es posible que necesite un corto periodo de adaptación.

En cada fase deberemos aprender algunas cosas nuevas y experimentar con nuevas aplicaciones, similares aunque a veces ligeramente diferentes, a las que estamos acostumbrados. Hay mucha ayuda en la Internet: guías, tutoriales paso a paso, manuales, grupos de usuarios que ofrecen apoyo, etc. También es posible que, por la naturaleza de los requerimientos informáticos del usuario, no llegue el momento en que utilice solamente software libre. Algunas aplicaciones especializadas todavía no tienen equivalentes libres de su mismo nivel. Pero... sólo es cuestión de tiempo. Mientras tanto podemos experimentar con un entorno mixto de aplicaciones libres y propietarias. Si al final utiliza cuantas aplicaciones libres le sea posible en su trabajo, el objetivo de esta propuesta de actividades se habrá conseguido con creces. No se rinda a la primera dificultad.

Otro consejo: si tiene un amigo o compañero de trabajo usuario avanzado de GNU/Linux, le será de gran ayuda. Coméntele sus intenciones y recibirá toda la ayuda de la que sea capaz su amigo. Los usuarios de GNU/Linux suelen ser también activos proselitistas y abanderados de dicho sistema operativo y del software libre en general y le ayudarán encantado, por la intensa satisfacción de “liberar” a una persona del software privativo, de los virus y de la lógica mercantilista y creadora de dependencia que implica el software privativo.

1. Primer contacto con el software libre

La siguiente tabla (Tabla 2) le ayudará en esta fase del proceso. Además, seguramente, Ud. ya debe utilizar aplicaciones libres. Muchas de las que se pueden bajar de Internet y usar gratuitamente, son libres. Descargue, instale y utilice algunas aplicaciones libres básicas. Verá como funcionan extraordinariamente bien y le ofrecen posibilidades que las propietarias no tienen. No se frustre si los menús no están en el mismo sitio que en otro programa: enseguida se familiarizará con estos pequeños cambios. Siga los siguientes pasos:

1. Comience por las más sencillas y parecidas a las que usa actualmente: por ejemplo, el navegador Mozilla Firefox y el programa de correo Mozilla Thunderbird.

2. Siga por la suite ofimática OpenOffice.org.

3. Explore otras aplicaciones libres disponibles para sus sistema operativo actual (vea en la Tabla 2 una selección de lo mejor del software libre para su sistema operativo actual).

4. Si es docente, pruebe también algunas aplicaciones educativas libres: JClic, etc.

5. En las ultimas filas de la Tabla 2 hay direcciones de sitios web que recopilan software libre. Hay realmente muchísimo. Explore y pruebe aquellas aplicaciones que le parezcan más interesantes.

6. Comente y comparta con sus compañeros y amigos sus hallazgos de software libre: el software libre es para eso, para compartir y colaborar.

Tipo:	Aplicación libre:	Puede descargarse en:
Navegación Web	Mozilla Firefox	http://www.mozilla.org/products/firefox/
Mensajería instantánea	Gaim	http://gaim.sourceforge.net/downloads.php
Correo electrónico	Mozilla Thunderbird	http://www.mozilla.org/products/thunderbird/
Agregador RSS	RSSOwl	http://www.rssowl.org/download
Internet TV / Video Podcasting	Democracy Player	http://www.getdemocracy.com/
Compartir ficheros P2P	Azureus	http://azureus.sourceforge.net/download.php
Video players	VLC MPlayer	http://www.videolan.org/vlc/ http://www.mplayerhq.hu/design7/dload.html
Conversión video Creación DVD	Media Coder	http://mediacoder.sourceforge.net/download.htm
Ofimática (suite completa) Edición de textos	OpenOffice.org AbiWord	http://www.openoffice.org/ http://www.abisource.com/download/
Podcasting	Juice	http://juicereceiver.sourceforge.net/index.php
DVD Ripping	Handbrake	http://handbrake.m0k.org/download.php
Sonido	Audacit	http://audacity.sourceforge.net/download/windows
Gráficos	GIMPShot Paint.net Inkscape	http://www.gimpshop.net/ http://www.getpaint.net/download.html http://www.inkscape.org/download.php
Transferencia de ficheros	Filezilla	http://sourceforge.net/project/showfiles.php?group_id=21558
IRC	X-Chat 2	http://silverex.info/download/
Gráficos 3D y modelado	Blender Jahshaka	http://www.blender3d.org/cms/Blender.31.0.html http://www.jahshaka.org/component/option,com_docman/task,cat_view/gid,16/Itemid,49/
Astronomía	Celestia	http://www.shatters.net/celestia/download.html
Antivirus	ClaimWin	http://www.clamwin.com/
Actividades didácticas	JClic	http://clic.xtec.net/es/jclic/
Mapas conceptuales	CmapTools	http://cmap.ihmc.us/
Editor de páginas Web	NVU	http://www.nvu.com/download.php
Editor de paquetes SCORM e IMS Learning Desig	Reload	http://www.reload.ac.uk/tools.html
Recopilaciones de software libre (y alguno gratuito pero no libre) para Windows	CDLibre Paraisoft Alternativas libres WinSLow	http://www.cdlibre.org/ http://www.paraisoft.com/ http://alts.homelinux.net/ http://winslow.aditel.org/
Aplicaciones instalables en una memorias Flash USB (para llevar)	Framakey Portable USB Software: A Melange	http://www.framakey.org/ http://meprisant2.blogspot.com/2006/01/portable-usb-sofftware-melange.html

7. Tabla 2: Software libre recomendado para Windows

2. Probar GNU/Linux: en vivo y en directo

Ya es el momento de probar GNU/Linux. Puede hacerlo con una distribución *Live*. Un sistema tipo *Live* le permite arrancar su ordenador desde un CD o DVD en lugar del disco duro, utilizar todas las aplicaciones que contiene y guardar sus documentos en otros medios (una memoria flash, por ejemplo). Al terminar la sesión los contenidos de su disco duro estarán intactos: no habrá borrado ni escrito nada en él. Esta etapa le permitirá familiarizarse con el sistema operativo GNU/Linux sin ningún riesgo: podrá volver a su sistema habitual después de finalizar la sesión y expulsar el CD o DVD. Para conseguir un sistema GNU/Linux *Live* puede usar la distribución actualmente más popular de GNU/Linux, Ubuntu, (puede descargarla de <http://www.ubuntu.com/>) y grabar en su ordenador un CD o DVD de arranque. Si no tiene una buena conexión a Intentet, puede utilizar los CDs o DVDs que regalan con las revistas de Linux o pedirlo por correo (es gratuito).

3. Arranque dual

En esta fase, instalará un sistema GNU/Linux en su ordenador, junto al sistema que actualmente tiene. En primer lugar, haga una copa de los contenidos de su disco duro. Esta fase es un tanto delicada, como toda instalación de software, y corre el riesgo, si se equivoca, de borrar completamente su disco duro. Una copia de seguridad es siempre una buena idea. Siga las instrucciones de instalación de la distribución GNU/Linux que haya probado y si conoce a algún usuario experimentado de Linux, pídale ayuda. Configure el arranque para que pueda escoger qué sistema utilizará cada vez que inicie el ordenador. En esta etapa comprobará que prácticamente cualquier cosa que hacía con el software privativo, puede hacerla con software libre. Al principio puede que se sienta un poco “torpe”, como si fuera un usuario novato de nuevo. La sensación desaparecerá a medida que use más el sistema y las aplicaciones. Piense que utilizamos el sistema un 10% del tiempo y aplicaciones concretas el 90% restante. Las aplicaciones libres son muy parecidas en su funcionamiento a las propietarias, por tanto, la mayor parte del tiempo se sentirá cómodo.

4. Al fin libres...

Si después de varios meses descubre que ya no utiliza para ninguna tarea el anterior sistema operativo y el software privativo, considere la posibilidad de “hacer sitio”, siempre necesario en su ordenador y borrarlo definitivamente. También puede dejarlo dónde está: al fin y al cabo ha pagado su buen dinero por utilizarlo (con numerosas restricciones) cuando compró el ordenador. Pero cuando adquiera otro ordenador, insista al vendedor que quiere uno “libre” con GNU/Linux y que no le cobre por el otro sistema que Ud. ya no utiliza.

En este punto comprobará el nivel de dominación del mercado al que ha llegado cierta empresa de software y la estrecha connivencia entre los fabricantes de hardware y los de software y la poca libertad que tiene en realidad el usuario. Si tiene un momento, piense en el argumento de la libertad de elección de los usuarios, esgrimido por las empresas de software privativo para frenar las iniciativas públicas de difusión del software libre.

Bibliografía de profundización

González Barahona, J., Seoane, J., Robles, G. (2003). *Introducción al software libre.* Universitat Oberta de Catalunya (UOC). Formación de Postgrado. Disponible en: <http://www.uoc.edu/masters/esp/img/693.pdf>.

Más, J. (2005). *Software Libre. Técnicamente viable, económicamente sostenible y socialmente justo*. Infonomia.com. Accesible en <http://www.softcatala.org/~jmas/swl/llibrejmas.pdf>. Consultado el 31 de julio de 2006.

Raymond, E. S. (1997). *La catedral y el bazar*. Disponible en: <http://es.tldp.org/Otros/catedral-bazar/cathedral-es-paper-00.html>.

Stallman, R.M. (1997). *El derecho a lee*r. Disponible en: <http://www.gnu.org/philosophy/right-to-read.es.html> y (1996) *La definición de software libre*. Disponible en <http://www.gnu.org/philosophy/free-sw.es.html>. Consultados el 31 de julio de 2006.

Véase también las últimas noticias en las páginas web de la *Free Software Fundation* (FSF) <http://www.fsf.org>, la Fundación Software Libre América Latina <http://www.fsfla.org/> y GNU España <http://www.es.gnu.org>.

Bibliografia citada

ADELL, J. (2005). Del software libre al conocimiento libre. Andalucía educativa, 51, Octubre de 2005, págs., 7-10, http://www.juntadeandalucia.es/educacion/portal/com/bin/Contenidos/IEFP/ANDALUCIA_EDUCATIVA/ANDALUCIA_EDUCATIVA/1133272276307_opinion.pdf (4/8/2006).

AMATRIAIN, X. (2004). Free software in education: a guide for its justification and implementation, http://www.create.ucsb.edu/~xavier/FreeSoftware/FreeSoftwareEducation/FreeSoftwareEducation.html (7/8/2006). Hay traducción catalana del mismo autor en http://www.create.ucsb.edu/~xavier/FreeSoftware/ProgramariLliureEducacio.pdf.

ATWELL, G. (2005). What is the Significance of Open Source for the Education and Training Community? Preoceedings of the First International Conference on Open Source Systems, Genova, 11th-15th July 2005, http://oss2005.case.unibz.it/Papers/OES/EK4.pdf (2/8/2006).

BECTA (2006). Open Source Software in Schools: A study of the spectrum of use and related ICT infrastructure cost, http://www.becta.org.uk/corporate/publications/documents/BEC5606_Full_report18.pdf (3/8/2006).

BINACUZZI, F. (2005). ERSR: “We Don't Need the GPL Anymore”. O'Reilly ONLamp.com, http://www.onlamp.com/pub/a/onlamp/2005/06/30/esr_interview.html (2/8/2006).

BRETTHAUER, D. (2001). Open Source Software: A History. Information Technology and Libraries (ITAL), 21(1), http://www.lita.org/ala/lita/litapublications/ital/2101bretthauer.htm (1/8/2006).

DIBONA, C., COOPER, D. y STONE, M. (eds.).(2004). Open Sources 2.0: The Continuing Evolution. Sebastopol, CA: O'Reilly & Associates.

DIBONA, C., OCKMAN, S. y STONE, M. (eds.). (1999). Open Sources. Voices from the Open Source Revolution. O'Reilly & Assoc., Sebastopol: CA. http://www.oreilly.com/catalog/opensources/book/toc.html (21/7/2006).

EDWARDS, K. (2004). Epistemic Communities, Situated Learning and Open Source Software Development, http://opensource.mit.edu/papers/kasperedwards-ec.pdf (15/7/2006).

EISENBERG, J.D. (2005). OASIS OpenDocument Essentials. Using OASIS OpenDocument XML, http://books.evc-cit.info/ (2/8/2006).

FABER, B.D. (2002). Educational Models and Open Source: Resisting the propietary University. ACM Special Interest Group for Design of Communications, Proceedings of the 20th Annual International Conference on Computer Documentation, 31-8, http://portal.acm.org/ft_gateway.cfm?id=584961&type=pdf&coll=GUIDE&dl=GUIDE&CFID=15151515&CFTOKEN=6184618 (24/7/2006).

GONZÁLEZ BARAHONA, J., SEOANE PASCUAL, J., ROBLES, G. (2003). Introducción al software libre. Universitat Oberta de Catalunya (UOC). Formación de Postgrado, http://www.uoc.edu/masters/esp/img/693.pdf.

HARS, A. y OU, S. (2001). Working for Free? -Motivations of Paticipating in Open Source Projects. Proceedings of the 42th Hawaii International Conference on System Sciences. http://csdl.computer.org/comp/proceedings/hicss/2001/0981/07/09817014.pdf (2/8/2006).

HART, T.D. (2003). Open Source in Education. Documento inédito, http://portfolio.umaine.edu/~hartt/OS%20in%20Education.pdf (2/8/2006).

III-UM (International Institute of Infonomics, University of Maastrich) y Berlecom Research, GmbH. (2002). FLOSS Final Report, http://www.infonomics.nl/FLOSS/report/ (2/8/2006).

KHALAK, A. (2000). Economic Model for Impact of Open Source Software, http://opensource.mit.edu/papers/osseconomics.pdf (15/7/2006).

KIM, E. (2005). F/OSS Adoption in Brazil: the Growth of a National Strategy. En Karaganis, J. y Latham (2005). The Politics of Open Source Adoption. Social Science Research Council, http://www.sscr.org/wiki/POSA, (7/8/2006).

LEHMAN, F. (2004). FLOSS Developers as a Social Formation, http://www.firstmonday.org/issues/issue9_11/lehman/ (15/7/2006).

LERNER, J. y TIROLE, J. (2000),. The Simple Economics of Open Source, http://www.nber.org/papers/w7600 (15/7/2006).

LESSIG, L. (2004). Free Culture: how big media uses technology and the law to lock down culture and control creativity. New York: The Penguin Press, http://www.free-culture.cc/. Hay y traducción al castellano en http://www.elastico.net/archives/001222.html (2/8/2006).

LEVESQUE, M. (2004). Fundamental Issues with Open Source Software Development. First Monday, 9(4) (April 2004), http://firstmonday.org/issue9_4/levesque/index.html (8/8/2006).

LYN, Y. (2005). Hybrid innovation: how does the collaboration between the FLOSS community and corporations happen? Knowledge, Technology and Policy, http://opensource.mit.edu/papers/lin4_hybrid.pdf (31/7/2006).

MAS, J. (2003). El software libre y las lenguas minoritarias: una oportunidad impagable, Digithum UOC. 5, http://www.uoc.edu/humfil/articles/esp/mas0303/mas0303.html (8/8/2006).

MAS, J. (2005). *S*oftware Libre. Técnicamente viable, económicamente sostenible y socialmente justo. Madrid, Infonomia.com, http://www.softcatala.org/~jmas/swl/llibrejmas.pdf (31/7/2006).

MOGLEN, E. (2003). Freeing the Mind: Free Software and the Death of Propietary Culture. Keynote address at the University of Maine Law School's Fourth Annual Technology and Law Conference, Portland, Maine, June 29, 2003. http://moglen.law.columbia.edu/publications/maine-speech.html (3/8/2006).

OSI (Open Source Initiative) (2006). The Open Source Definition. v. 1.9, http://www.opensource.org/docs/definition.php (31/7/2006).

PERENS, B. (1999). The Open Source Definition. En DiBona, C., Okman, S., y Stone, M. (eds.). Open Sources: Voices from the Open Source Revolution. Sebastopol, CA: O'Reilly & Associates.

PFAFFENBERGER, B. (2000). Linux in Higher Education: Open Source, Open Minds, Social Justice, Linux Journal, http://www2.linuxjournal.com/article/5071 (3/8/2006).

PRENSKY, M. (2001). Digital Natives, Digital Immigrants, http://www.marcprensky.com/writing/Prensky%20-%20Digital%20Natives,%20Digital%20Immigrants%20-%20Part1.pdf (8/8/2006).

RAYMOND, E. S. (1997). La catedral y el bazar. http://es.tldp.org/Otros/catedral-bazar/cathedral-es-paper-00.html (31/7/2006). Hay edición impresa, junto a otros ensayos, titulada genéricamente The Cathedral and the Bazaar: Musing on Linux and Open Source by an Accidental Revolutionary. Sebastopol, CA., O'Reilly and Assoc. 1999.

RAYMOND, E.S. (1999). The Cathedral and the Bazaar: Musing on Linux and open Source by an Accidental Revolutionary. Sebastopol, CA., O'Reilly and Assoc. 1999.

RAYMOND, E.S. (2004). Open Minds, Open Source. Analog, june-july 2004, http://www.catb.org/~esr/writings/analog.html (2/8/2006).

SHOCKEY, K. y CABRERA, P.J. (2005). Using Open Source to Enhance Learning. ITEH 6th Annual International Conference, july 7-9, 2005, Juan Dolio, Dominican Republic, http://fie.engrng.pitt.edu/ithet2005/papers/2042.pdf (2/8/2006).

STALLMAN, R.M. (1984a). The GNU Manifesto, http://www.gnu.org/gnu/manifesto.html (1/8/2006).

STALLMAN, R.M. (1984b). Why Software Should Not Have Owners, http://www.gnu.org/philosophy/why-free.html (1/8/2006).

STALLMAN, R.M. (1992). Why Software Should Be Free, http://www.gnu.org/philosophy/shouldbefree.html (15/7/2006)..

STALLMAN, R.M. (1996). Free Software Definition. En Stallman, R. (2002). Free Software, Free Society: Selected Essays of Richard M. Stallman. Boston, MA., GNU Press, págs. 41-43.

STALLMAN, R.M. (1997). El derecho a leer, http://www.gnu.org/philosophy/right-to-read.es.html (31/7/2006).

STALLMAN, R.M. (1999). The GNU Operating System and the Free Software Movement. En DIBONA et al., (1999).

STALLMAN, R.M. (2003). Por qué las escuelas deben usar exclusivamente software libre, http://www.gnu.org/philosophy/schools.es.html (8/8/2006).

STARING, K., TITLESTAD,O.H Y GAILIS, J. (2005). Educational Transformation through Open Source Approaches. Proceedings of IRIS'28, Kristiansand, Norway, 2005, http://www.hia.no/iris28/Docs/IRIS2028-1106.pdf (2/8/2006).

TUOMI, I. (2005). The Future of Open Source: Trends and Prospects. En Wynants, M. y Cornelis, J. (eds.). *How Open is the Future? Economic, Social and Cultural Scenarios Inspired by Free and Open Source Software*. Brussels, VUB Brussels University Press. http://crosstalks.vub.ac.be/publications/Howopenisthefuture/howopenfuture_CROSSTALKSBOOK1.pdf (2/8/2006).

VUORIKARI, R. (2004). Insigth Special Report: Why Europe Needs Free and Open Source Software and Content in Schools, http://www.eun.org/insight-pdf/special_reports/Why_Europe_needs_foss_Insight_2004.pdf (2/8/2006).

WAYNER, P. (2000). Free for All. How Linux and the Free Software Movement Undercut the High-Tech Titans. Boston, HarperCollins, http://www.wayner.org/books/ffa/ (21/7/2006).

WHELEER, D.A. (2005). Why Open Source Software / Free Software (OSS/FS, FLOSS, or FOSS)? Look at the Numbers!, http://www.dwheleer.com/oss_fs_why.html (21/7/2005).

WIKIPEDIA (2006). Entradas “Criticism of Microsoft”, “Fear, uncertainty and doubt”, “Embrace, extend and extinguish”y “Open Document Format”, http://en.wikipedia.org (2/8/2006).

WILLIAMS, S. (2002). Free as in Freedom. Richard Stallman's Crusade for Free Software. Sebastopol, CA., O'Reilly and Assoc, http://www.oreilly.com/openbook/freedom/ (15/7/2005).

WYNANTS, M. y Cornelis, J. (2005). Preface. En Marleen Wynants & Jan Cornelis (Eds) How Open is the Future? Economic, Social & Cultural Scenarios inspired by Free & Open-Source Software. Bruselas: VUB University Press, http://crosstalks.vub.ac.be/publications/Howopenisthefuture/howopenfuture_CROSSTALKSBOOK1.pdf (21/7/2006).

XALABARDER, R. (2006). Las licencias Creative Commons: ¿una alternativa al *copyright*? UOC Papers, 2, http://www.uoc.edu/uocpapers/dt/esp/xalabarder.html (31/7/2006).