Manual de la asignatura Ingeniería de Software de Fuentes Abiertas/Libre - versión 2022

Ricardo Medel y colaboradores

Índice general

Prefacio

Este texto es producto de un ejercicio de trabajo colaborativo utilizando herramientas digitales que son usadas regularmente en comunidades de software libre[1], realizado entre los años 2016 y 2018. Si bien su ambicioso título es *Manual de la asignatura Ingeniería de Software de Fuentes Abiertas/Libre*, más bien debe considerarse como una serie de apuntes sobre los diferentes temas abordados en esta asignatura electiva (de quinto nivel) de la carrera de Ingeniería en Sistemas de Infomación de la Facultad Regional Córdoba de la Universidad Tecnológica Nacional.

Estos apuntes, entonces, no deben tomarse como el texto definitivo de la asignatura, ya que ninguno de los capítulos cubre todos los aspectos de cada tema ni profundiza en ellos. Más bien, se recomienda utilizarlo como una base para comenzar una exploración más profunda de cada tema, a partir de las referencias y enlaces provistos.

Como todo trabajo colaborativo, este texto es la suma de los aportes de cada autor/a y la interacción entre autores/as, por lo que se puede notar cierta heterogeneidad en los estilos de escritura y los enfoques con los que se aborda cada tema. Hemos hecho ciertos intentos de homogeneización del estilo del texto, pero aún queda trabajo por hacer en ese sentido.

Durante los años en que se escribió y modificó este texto, muchas han sido las personas que hicieron sus aportes, no todas dejaron su registro[2], por lo que esta lista puede estar incompleta: Pablo Bajo, Luciano Bartoszensky, Federico Benito, Agustín Borello, Braian Chincho, Nahuel Ignacio Cuello, Alexis Donato, Manolo Fernández, Facundo Ferrero, Juan Filardo, Máximo Fiora, Mayco Garelli, Marcos López, Carlos Luna, Paolo Mattio, Ricardo Medel, Fernando Meichtri, Diego Moreno Moreyra, Jeremías Niño, Federico Prado, Fernanda Pucheta, Julieta Ríos, Mauricio Sánchez, Lucas Martín Segurado, Fabricio Simoncelli, Milena Vilardo, Gonzalo Ulla, Fernando Zamora.

[1] Herramientas utilizadas: el sistema de control de versiones Git, la plataforma de repositorios GitHub y el procesador de textos LaTeX.

[2] Si considerás que sos autor/a y no se te incluyó en esta lista, por favor, enviame un email a `rmedel@frc.utn.edu.ar` indicando que participaste en la creación de este manual.

A todas ellas y todos ellos, va el agradecimiento de la cátedra, por su aporte a este texto que nos permite tener un punto de partida en común para poder desarrollar cada uno de los temas que abordamos en esta asignatura única en su tipo, al menos por ahora, en todo el país.

Si las lectoras o los lectores encuentran errores o quieren aportar mejoras, son libres de hacerlo por su propia iniciativa, ya que este documento tiene licencia libre y su código fuente está accesible en un repositorio público[3].

Ricardo Medel
Córdoba, 8 de marzo de 2022

[3] https://github.com/RHMedel/Manual-ISFAL

Capítulo 1

Introducción al software libre

En este capítulo hacemos una revisión ligera de los principales hitos históricos del software libre, para lo cual necesitamos presentar algunos conceptos técnicos y legales, tales como la definición de software libre, las licencias de software o los sistemas operativos. Aquí algunos de estos conceptos son abordados someramente, pero volveremos sobre ellos con más detalles en los siguientes capítulos.

1.1. Historia del software libre

Si bien la historia del software libre es relativamente corta, sus principales eventos, sus protagonistas y, especialmente, sus efectos en la informática tal como la conocemos son abundantes, interesantes y muy ricos desde el punto de vista histórico. Gran cantidad de artículos, libros y hasta películas[1] sobre este tema han sido realizados. En esta sección nos proponemos realizar apenas una aproximación a su historia, proveyendo abundante bibliografía para que el o la lectora interesada pueda profundizar a su gusto.

Desde que en los años ‘50 la creciente miniaturización de los componentes electrónicos permitió la comercialización de computadoras y hasta comienzos de los años ‘70, aproximadamente, la mayoría de las compañías que producían computadoras digitales o periféricos tenían el hábito de dejar a disposición de los usuarios el código fuente (archivos legibles) de su software. Si algún usuario o grupo de usuarios realizaba mejoras a ese software, estas mejoras eran compartidas en las comunidades de usuarios de dichas computadoras e incluso las empresas adoptaban estas mejoras. Esto se debía principalmente a que las comunidades de usuarios

[1] *Revolution OS* es una película de 85 minutos sobre la historia del software libre desde sus comienzos hasta 2001. `http://revolution-os.com/`

eran pequeñas, ya que los costos tanto de las computadoras como de la infraestructura y mano de obra requerida para operar eran prohibitivos y por lo tanto se vendían apenas unas decenas de cada modelo, y también a que el software funcionaba solamente en el hardware para el que había sido escrito.

A comienzos de los '70 los avances de la década anterior en el desarrollo de sistemas operativos y compiladores permitieron el nacimiento de la industria del software como un fenómeno separado del hardware. De esta forma, el código fuente pasó a tener un valor comercial que antes no tenía y comenzaron las tensiones entre quienes comercializaban software y quienes lo consideraban una herramienta que debería estar al alcance de todos para su mayor desarrollo. Dos anécdotas, separadas casi por una década, marcaron esa tensión y una de ellas generaría un importante cambio en la cultura del desarrollo de software.

En febrero de 1976 Bill Gates, apenas meses después de fundar la empresa Microsoft, publicó una "Carta abierta a los aficionados"[2], en la que consideraba que lo que los aficionados al desarrollo de software llamaban *compartir* era en realidad, y en sus palabras, *robar* y eso impedía el desarrollo profesional de software.

A principios de la década del '80 Richard Stallman, mientras trabajaba en el Laboratorio de Inteligencia Artificial del Instituto Tecnológico de Massachusetts (MIT), quiso hacer unas modificaciones al software de una impresora Xerox y se encontró con que el código fuente no estaba disponible y aquellos investigadores que sí tenían acceso a dicho código habían firmado acuerdos de no revelarlo[3]. Este fue uno de los incidentes[4] que llevaron a Richard Stallman a dedicar su vida a la creación del movimiento del software libre [Williams, 2002], comenzando entre 1983 y 1985 con el proyecto GNU[5] para el desarrollo de un sistema operativo similar al entonces famoso Unix[6], la publicación de "El manifiesto de GNU"[7] y la creación de la Fundación Software Libre[8].

En la sección siguiente daremos una definición más precisa del Software Libre, pero por ahora será suficiente indicar que es aquel software que permite a los y las usuarias utilizarlo, modificarlo y distribuirlo sin restricciones.

Desde medidados de los años '80 el software libre y el movimiento sociopolítico que lo rodea no han parado de crecer. Sin embargo, a comienzos de los '90 el proyecto GNU había desarrollado muchas de las herramientas informáticas reque-

[2] "Open Letter to Hobbyists", por su título original en inglés [Gates, 1976].

[3] NDA, Non-Disclosure Agreement, en inglés.

[4] Otro posible evento fue el desacuerdo entre Stallman y Symbolics, Inc. sobre el acceso a las actualizaciones que Symbolics había realizado a su máquina Lisp, la cual estaba basada en código de libre acceso realizado por el MIT.

[5] `https://www.gnu.org/`

[6] GNU's Not Unix, es un acrónimo recursivo que significa "GNU No es Unix"

[7] `https://www.gnu.org/gnu/manifesto.es.html`

[8] FSF, Free Software Foundation en inglés. `https://www.fsf.org/`

ridas para un sistema operativo (editores de texto, compiladores, etc.) pero no tenía un componente clave: el núcleo (o *kernel*, en inglés). Fue en 1991 que, paralelamente al proyecto GNU y aprovechando la posibilidad de realizar desarrollo distribuido gracias a la creciente difusión de la Internet, un estudiante finlandés, Linus Torvalds, comenzó el desarrollo de un núcleo libre, con características similares a Unix, el cual se llamaría Linux. Esta pieza vino a completar todos los componentes requeridos para tener un sistema operativo libre, conocido como GNU/Linux.

Aproximadamente para la misma época, en 1994, se lanzaba la versión totalmente libre del sistema operativo BSD[9], también derivado de Unix. Este sistema y sus diferentes versiones han tenido cierto éxito, pero no puede compararse con la amplia difusión de las versiones de GNU/Linux, llamadas *distribuciones* o *distros*.

Hacia fines de la década de los '90 el movimiento de software libre había crecido, especialmente impulsado por la Internet, la cual permite a los y las desarrolladoras de software trabajar en forma distribuida desde prácticamente cualquier lugar del mundo y, asimismo, distribuir el software sin los costos asociados a los canales de comercialización tradicionales. Sin embargo, poco impacto se había logrado en el mundo de los negocios y las grandes empresas, y varias personas mostraron su disconformidad con las fuertes posiciones políticas de Richard Stallman. Fue así que en 1998, Bruce Perens, Eric S. Raymond y Jon "Maddog" Hall, entre otras personas, crearon la Iniciativa de Fuentes Abiertas (OSI, por sus siglas en inglés[10]). Su motivación es menos política y más pragmática, haciendo hincapié en las ventajas técnicas y económicas del software que pone el código fuente a disposición de todos los y las usuarias.

El mismo año de creación de la OSI, el navegador de internet Netscape, que unos años antes había sido el de mayor penetración de mercado (hasta 90 %), liberó su código como respuesta a la agresiva campaña de Microsoft para imponer su navegador Internet Explorer. La creación de la OSI y la liberación del código de Netscape tuvieron un significativo impacto comercial y durante el resto de la década, y hasta la explosión de la así llamada *burbuja puntocom*, fueron muchas las compañías basadas en software libre o software de fuentes abiertas que comenzaron a cotizar en bolsa con valuaciones millonarias.

En los últimos años, la difusión del software libre ha sido muy importante, excepto en las computadoras personales (ya sea de escritorio o portátiles) donde el sistema operativo mayoritario sigue siendo alguna versión de MS Windows. Donde el software libre es más fuerte es en los servidores que prestan servicios a través de Internet. Allí, el *stack* de software está generalmente constituido por diferentes distribuciones de GNU/Linux y servidores web libres, tales como Nginx o Apache.

[9] BSD por *Berkeley Software Distribution*.

[10] *Open Source Initiative*, `https://opensource.org/`

Así mismo, desde el año 2017 todas las supercomputadoras en el top500, el ranking de las computadoras más veloces del mundo, creado en 1995, corren bajo alguna versión de Linux.

En la actualidad, el software libre permite la creación rápida de *start ups* tecnológicas, ya que reduce los costos iniciales de desarrollo de las herramientas de software necesarias (evitando tanto el autodesarrollo como el pago de licencias por software privativo durante las etapas iniciales del negocio) y permite la adaptación a los requisitos particulares del nuevo emprendimiento [Kamau, 2016, Singh, 2021, Gerasimenko, 2016, Wiggers, 2021].

Completamos esta suerte de historia general del software libre y de fuentes abiertas con dos hitos anecdóticos que marcan sendos momentos en que el éxito en la difusión del software libre fue remarcado (en formas totalmente diferentes) por expresiones públicas de representantes de la empresa que supo encarnar todo lo opuesto al software libre, Microsoft.

- En junio de 2001 el por entonces gerente general de Microsoft, Steve Ballmer, dijo en un reportaje que "Linux es un cancer que contagia, en el sentido de propiedad intelectual, a todo lo que toca."[11]

- En una charla durante un evento en octubre de 2014 el por entonces nuevo gerente general de Microsoft, Satya Nadella, presentó la nueva posición de la empresa ante el software libre, resumiéndola con una filmina que decía "Microsoft ♥ Linux".[12]

1.2. Conceptos del software libre y de fuentes abiertas

Comenzamos formalizando la definición de *Software Libre* tal como la estableció Richard Stallman como parte del proyecto GNU: aquel software cuyos términos de uso (su licencia) le aseguran a los y las usuarias las siguiente cuatro libertades esenciales.

0. La libertad de utilizar el software con cualquier propósito.

1. La libertad de estudiar el programa y modificarlo para adaptarlo a las propias necesidades.

2. La libertad de distribuir copias del programa.

[11] La cita original es "Linux is a cancer that attaches itself in an intellectual property sense to everything it touches." La entrevista original ya no está disponible en el sitio del Chicago Sun-Times, pero un análisis (en inglés) de la noticia aún puede leerse en The Register[Greene, 2001].

[12] Grabación de la charla de Nadella: `https://youtu.be/54hHr8ye2kE`

3. La libertad de distribuir copias de las versiones modificadas del programa.

Cabe destacar que las libertades 1 y 3 requieren de acceso al *código fuente*, es decir, los archivos conteniendo el software en un formato legible por un ser humano (y por un compilador o intérprete). Por lo que se puede considerar que una característica fundamental del Software Libre es que sea de *Fuentes Abiertas*[13].

Como dijimos previamente, la creación en 1998 de la *Open Source Initiative* tenía como objetivo hacer más amigable al software libre con el ámbito de los negocios. Además de cambiar el confuso (en inglés) nombre de Software Libre al más neutro Fuentes Abiertas, se estableció una definición alternativa, aunque conceptualmente similar. Para la OSI, un software es de Fuentes Abiertas si cumple con los siguientes 10 criterios.

1. **Redistribución libre**: La licencia no restringirá el derecho de vender o regalar el software como parte de un paquete de software. No se podrá requerir el pago de tasas por las ventas.

2. **Código fuente**: El programa debe incluir el código fuente y debe permitir la distribución de dicho código y de su forma compilada. Si el producto no se distribuye con el código fuente, debe haber una forma clara de obtenerlo por no más que un costo razonable de reproducción, preferentemente descargándolo de la Internet en forma gratuita. El código fuente debe ser la forma preferida en que un/a programador/a modifique el programa. Código ofuscado deliberadamente no es aceptable. Formas intermedias, como el resultado de un preprocesador o un traductor no son permitidas.

3. **Trabajos derivados**: La licencia debe permitir modificaciones y trabajos derivados, y permitir que sean distribudos bajo los mismos términos que la licencia del software original.

4. **Integridad del código fuente del/la autor/a**: La licencia puede restringir la distribución de modificaciones del código fuente solo si permite la distribución de ”parchesçon el código fuente que modifiquen el programa al momento de construcción (*build*) La licencia debe permitir explícitamente la distribución del software construido a partir del código modificado. La licencia puede requerir que los trabajos derivados tengan un nombre o número de versión diferentes del software original.

5. **No discriminación contra personas o grupos**: La licencia no debe discriminar a ninguna persona o grupo de personas.

[13] *Open Source*, en inglés.

6. **No discriminación contra actividades**: La licencia no debe restringir a nadie de hacer uso del programa en una actividad específica.

7. **Distribución de licencia**: Los derechos vinculados a un programa deben aplicarse a todas las partes a las que les es redistribuido sin la necesidad de licencias adicionales para dichas partes.

8. **La licencia no debe ser específica para un producto**: Los derechos vinculados a un programa no deben depender de que tal programa sea parte de una distribución particular de software. Si el programa es extraído de dicha distribución y utilizado o distribuido respetando los términos de su licencia, todas las partes a quienes el programa es redistribuido deben tener los mismos derechos que han sido otorgados en conjunto con la distribución original.

9. **La licencia no debe restringir otro software**: La licencia no debe establecer restricciones sobre otro software que sea distribuido junto con el software sobre el cual se aplica dicha licencia.

10. **La licencia debe ser tecnológicamente neutral**: Ninguna parte de la licencia debe basarse en una tecnología o estilo de interfaz específicos.

Aunque las definiciones en principio lucen muy diferentes, realmente no establecen un concepto diferente para el software libre o de fuentes abiertas. La principal diferencia entre la Free Software Foundation, promotora del término y definición del software libre, y la Open Source Inititative, promotora del software de fuentes abiertas, está en sus motivaciones fundamentales.

La FSF argumenta que el software es conocimiento y debe poderse difundir sin trabas. Ocultar el conocimiento es una actitud antisocial y moralmente reprensible. Además, la posibilidad de estudiar y modificar programas es una forma de libertad de expresión.

Por su parte la OSI tiene una motivación más pragmática, argumentando que el software de fuentes abiertas tiene ventajas técnicas y económicas que solo se logran compartiendo el código fuente.

En ese aspecto, existe alguna confusión respecto a que el software libre es igual a software gratuito. En principio, como se mencionó arriba, la primera confusión viene del término *free software*, ya que *free* en inglés significa tanto "libre" como "gratuito"[14]. Es por eso que, jocosamente, Richard Stallman aclara que el software es *free as in Freedom, not as in free beer*[15].

[14]Nótese que en español a veces también se utilizan ambos términos en forma similar, por ejemplo, cuando se indica que en una fiesta hay "barra libre".

[15]El software es libre como en libertad, no como en cervezas gratis.

Por otro lado, es cierto que en general se puede acceder al software libre en forma gratuita, ya que al requerirse el acceso al código fuente y su posibilidad de distribuirlo (por las libertades 1 y 3), el costo del software, o mejor dicho de su código binario, tiende a cero: la empresa o comunidad desarrolladora lo puede vender pero sus primeros clientes lo pueden distribuir en forma legal (ya sea por menor costo o gratuitamente), por lo que los clientes siguientes no tendrán la necesidad de adquirirlo al precio establecido por sus desarrolladores. Como veremos más adelante, esto no impide que se puedan hacer negocios basados en software libre. Simplemente, no tiene sentido cobrar por utilizarlo, tal cual es el modelo de negocios tradicional en la industria del software.

Por completitud mencionaremos el *freeware* o *shareware* que, aunque ya no sean tan comunes esos términos, definen a software que es gratuito o que se comparte gratuitamente. en algunos casos con funciones reducidas (se de pagar para acceder a más funciones) y otros con la posibilidad de realizar pagos voluntarios a sus autores. La principal diferencia con software libre es que este tipo de software puede no ofrecer algunas de las libertades requeridas para ser definido como tal. Generalmente no se permite ni el acceso al código fuente, ni su modificación, ni la distribución de dichas modificaciones.

1.3. El software libre en Latinoamérica y el Caribe

Latinoamérica y el Caribe son regiones consumidoras de tecnología, ya sea importada de otras partes del mundo o producida localmente por sucursales de compañías extranjeras. A partir de los procesos de privatización y desregularización de las telecomunicaciones llevados adelante en los años ’80 y ’90, los servicios también están dominados por gigantes globales. La mayoría del software propietario líder en el mercado ha sido traducido a español y portugués en busca de un mercado creciente.

El software libre, si bien no con la fuerza que tiene en países centrales, también tiene su cuota de desarrollo en esta región. Mencionaremos algunos hitos y organizaciones que así lo muestran. No incluimos en este somero listado las diferentes organizaciones y eventos que surgieron a lo largo del tiempo y luego desaparecieron o se discontinuaron, ya que eso queda para un relevamiento histórico más completo.

El programador brasileño Marcelo Tosatti desde noviembre de 2001, cuando tenía solo 18 años, se convirtió en el mantenedor y co-mantenedor de ciertas versiones del kernel Linux.

El software de escritorio Gnome fue iniciado por los programadores mexicanos Miguel de Icaza y Federico Mena, forma parte oficial del proyecto GNU y es uno

de los escritorios más utilizados por distribuciones GNU/Linux.

El evento FLISoL (Festival Latinoamericano de Instalación de Software Libre)[16] se realiza desde el año 2005 simultáneamente en diferentes ciudades de países latinoamericanos. Comenzó como un encuentro presencial para instalar distribuciones de sistemas operativos en computadoras que los usuarios acercaran al evento, pero actualmente es un evento que incluye otras actividades, como charlas, talleres, presentaciones y ferias.

El Proyecto Software Libre de Brasil (*Software Livre Brasil* en portugués) es una red de personas e instituciones creada a principios de los años 2000 con el objetivo de promover el uso y desarrollo de software libre como una alternativa de libertad económica, tecnológica y de expresión en ese país[17]. En su momento de mayor auge pudo promover el desarrollo y uso de software libre en los gobiernos de varios estados de Brasil y del gobierno federal, así como la organización del Foro Internacional de Software Libre, que durante 18 años se llevó a cabo en Porto Alegre.

[16] `https://flisol.info/`

[17] `http://softwarelivre.org/portal/quem-somos`

Capítulo 2

Licencias de software

2.1. ¿Qué es una licencia de software?

Una licencia de software es un documento (generalmente en formato digital) a través del cual la empresa desarrolladora o dueña del software otorga al receptor o usuario ciertos derechos sobre el software desarrollado.

En general todo software, independientemente de su forma de distribución, ya sea como pieza separada de software o como servicio (a través de Internet), tiene una licencia (también conocida como EULA por las siglas en inglés de *End-User License Agreement* o Acuerdo de Licencia con el Usuario Final) o un documento de "condiciones de uso".

De lo anterior debe quedar claro que el software generalmente no se vende (ni su código fuente, ni su código binario) sino que se comercializa su derecho a usarlo por un tiempo determinado o a eternidad. Este concepto es importante, porque es la base del modelo de negocios mayoritario en la industria del software, tal como veremos en un capítulo posterior.

Si la licencia o EULA otorga al usuario las 4 libertades de la definición de software libre (ver capítulo 1) se dice que es una "licencia libre". Si la licencia es compatible con las 10 cláusulas que definen al software de fuentes abiertas (según la definición de la OSI que dimos en el capítulo 1) se dice que es una "licencia de software de fuentes abiertas". En caso contrario será una "licencia no libre" o, como muchas veces se las menciona, una "licencia privativa" (porque se priva al usuario de esas libertades o derechos).

A su vez, las licencias libres se clasifican en los siguientes dos tipos, según si permiten al usuario modificar la licencia para la redistribución del software.

- Permisivas: son las licencias libres que permiten al usuario redistribuir el software con otra licencia diferente, ya sea libre o privativa. Las licencias

MIT, BSD y Apache son ejemplos de este tipo de licencias.

- Robustas: obliga al usuario a utilizar la misma licencia para la redistribución del software libre. Esto asegura (u obliga) a que el software desarrollado como libre siempre se mantenga libre, independientemente de quién lo modifica y redistribuye posteriormente. El ejemplo más conocido de este tipo de licencia es la familia de licencias GPL (*GNU General Public License*).

2.2. Conceptos legales (y no tanto)

Para la ley argentina el software es una obra protegida por la propiedad intelectual. La ley original era la 11.723, que fue promulgada en 1933 y luego modificada para adaptarse a los tiempos. En 1998 se promulgó la ley 25.036, que modifica a la anterior y establece en su artículo primero[1] que "las obras científicas, literarias y artísticas comprenden los escritos de toda naturaleza y extensión, **entre ellos los programas de computación fuente y objeto**".

En general la ley reconoce al/la autor/a de la obra (en nuestro caso los/as desarrolladores/as del software) y a los/as titulares del derecho de propiedad intelectual. La autoría es un derecho moral y es eterno, es decir, nunca se puede cambiar quién es el o la autora de una obra, independientemente de quién tenga el derecho a la propiedad. En cambio el derecho a la propiedad, o también llamado "derecho a copia" o *copyright* en inglés, es un derecho patrimonial (el poseedor de este derecho puede pedir resarcimiento económico por el uso de su obra) y tiene limitaciones temporales que dependen del tipo de obra y la legislación de cada país.

El artículo 4 inciso (d) de la nueva ley de propiedad intelectual argentina establece que "las personas físicas o jurídicas cuyos dependientes contratados para elaborar un programa de computación hubiesen producido un programa de computación en el desempeño de sus funciones laborales, salvo estipulación en contrario" son titulares del derecho a la propiedad intelectual del software. Los/as desarrolladores/as contratados retienen el derecho moral de autoría del software, pero quien tiene el derecho patrimonial para obtener rédito económico de dicho software es la empresa que los contrató.

A partir del término inglés *copyright*, que establece al dueño de los derechos sobre una obra, Richard Stallman propuso, como otro juego de palabras, el concepto de *copyleft* para indicar que el autor de la obra otorga los derechos de dicha obra

[1] El texto actual de la ley 11.723, modificado por la ley 25.036 y otros decretos, está disponible en `http://servicios.infoleg.gob.ar/infolegInternet/anexos/40000-44999/42755/texact.htm`

a los usuarios[2]. En ese caso, las licencias libres serían consideradas como *copyleft*, ya que otorgan los derechos de uso, modificación y distribución a los usuarios, mientras que las licencias privativas serían *copyright*, pues los propietarios retienen la mayoría de esos derechos.

Recientemente aparecieron conceptos derivados, tales como *copyfarleft* y *copyfair*, que establecen ciertas reglas más estrictas para evitar que empresas que no colaboraron en el desarrollo de la obra puedan lucrar con su uso. En el caso de *copyfarleft* solo empresas de propiedad de sus trabajadores (por ejemplo, cooperativas) pueden beneficiarse económicamente de la obra. Por su parte, usar *copyfair* exige a las empresas que no colaboraron en el desarrollo el pago de una tasa si quieren beneficiarse económicamente del producto. Un ejemplo de este tipo de licencia es la PPL (*Peer Production License*)[3].

2.3. Compatibilidad de licencias

Es fácil pensar en un programa informático y su correspondiente licencia. Sin embargo, sabemos que actualmente los programas se constuyen por medio de la combinación de diferentes paquetes de software, librerías[4], módulos nuevos y viejos, modificados o no. Cada parte de un programa, al provenir de diferentes fuentes, puede tener diferentes licencias, por lo cual es importante entender bajo qué términos se puede licenciar el software desarrollado de esta forma.

Las licencias pueden tener cláusulas contradictorias entre sí, que hacen imposible combinar el código para crear nuevos paquetes de software. Por ejemplo si una licencia de un paquete de software dice que las versiones modificadas deben mencionar a todos los/as desarrolladores/as y la licencia de otro paquete dice que las versiones modificadas no pueden contener atribución de autoría adicional, entonces crear un software que utilice ambos paquetes modificados es posible técnicamente pero no se puede asociar una licencia que satisfaga a ambas licencias originales.

Si bien este problema no es exclusivo del software libre, existen más trabajos para simplificar el análisis de la incompatibilidad o compatibilidad de licencias en este tipo de software que en el privativo. Un ejemplo de tablas de compatilibidad de licencias de software libre y de fuentes abiertas son las matrices realizadas por la

[2]En inglés *right* significa tanto "derecho" en el sentido legal como "derecha" en ideología política, mientras que *left* significa tanto "izquierda" en el sentido político como "dejar" o "ceder" en el sentido de ceder derechos.

[3]PPL (Peer Production License): `https://wiki.p2pfoundation.net/Peer_Production_License`

[4]Entendemos que el término "librería" para referirse a una colección de funciones de software es una mala traducción del término en inglés *library*. Pero dado lo común de su uso, lo adoptamos en este texto.

Oficina de Coordinación de Software Libre de *Xunta de Galicia*[5], que mostramos en la Figura 2.1. Para entender la diferencia entre las dos matrices se debe tener en cuenta que un componente de software es "estático" cuando se combina con el resto del paquete en tiempo de compilación, copiándose dentro del ejecutable o binario de la aplicación, y es "dinámico" cuando es invocado en tiempo de ejecución, sin ser parte del código binario de la aplicación.

	MIT	BSD4	BSD3	ASL1	ASL2	LGPL2.1	LGPL2.1+	LGPL3+	MPL	CDDL	CPL/EPL	EUPL	GPL2	GPL2+	GPL3	AGPL3
MIT	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí
BSD4	Sí	Sí	Sí	Sí	Sí	No	No	No	Sí	Sí	Sí	Sí	No	No	No	No
BSD3	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí
ASL1	No	No	No	Sí	Sí	No	No	No	Sí	Sí	Sí	Sí	No	No	No	No
ASL2	No	No	No	No	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	No	No	Sí	Sí
LGPL2.1	No	No	No	No	No	Sí	No	No	No	No	No	Sí	Sí	Sí	Sí	Sí
LGPL2.1+	No	No	No	No	No	Sí	Sí	Sí	No	No	No	Sí	Sí	Sí	Sí	Sí
LGPL3+	No	No	No	No	No	No	No	Sí	No	No	No	No	No	Sí	Sí	Sí
MPL	No	No	No	No	No	No	No	No	Sí	No	No	No	No	No	No	No
CDDL	No	No	No	No	No	No	No	No	No	Sí	No	No	No	No	No	No
CPL/EPL	No	No	No	No	No	No	No	No	No	No	Sí	Sí	No	No	No	No
EUPL	No	No	No	No	No	Sí	Sí	No	Sí	Sí	Sí	Sí	Sí	Sí	No	No
GPL2	No	No	No	No	No	No	No	No	No	No	No	No	Sí	Sí	No	No
GPL2+	No	No	No	No	No	No	No	No	No	No	No	No	No	Sí	Sí	Sí
GPL3	No	No	No	No	No	No	No	No	No	No	No	No	No	No	Sí	Sí
AGPL3	No	No	No	No	No	No	No	No	No	No	No	No	No	No	Sí	Sí

(a) Matriz de compatibilidad de licencias de paquetes de software estáticos.

	MIT	BSD4	BSD3	ASL1	ASL2	LGPL2.1	LGPL2.1+	LGPL3+	MPL	CDDL	CPL/EPL	EUPL	GPL2	GPL2+	GPL3	AGPL3
MIT	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí
BSD4	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	No	No	No	No
BSD3	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí
ASL1	Sí	Sí	Sí	Sí	Sí	No	No	Sí	Sí	Sí	Sí	Sí	No	No	No	No
ASL2	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí
LGPL2.1	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí
LGPL2.1+	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí
LGPL3	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	No	Sí	Sí	Sí
MPL	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	No	No	No	No
CDDL	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	No	No	No	No
CPL/EPL	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	No	No	No	No
EUPL	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	No	No
GPL2	No	No	No	No	No	No	No	No	No	No	No	No	Sí	Sí	No	No
GPL2+	No	No	No	No	No	No	No	No	No	No	No	No	Sí	Sí	Sí	Sí
GPL3	No	No	No	No	No	No	No	No	No	No	No	No	Sí	No	Sí	Sí
AGPL3	No	No	No	No	No	No	No	No	No	No	No	No	Sí	No	No	Sí

(b) Matriz de compatibilidad de licencias de paquetes de software dinámicos.

Figura 2.1: Matrices de compatibilidad realizadas por el Centro Nacional de Referencia de Aplicación de las Tecnologías de la Información y Comunicación (CENATIC) en 2019

[5]Matriz de compatibilidad de licencias de software libre: `https://www.mancomun.gal/es/documento/matriz-de-compatibilidad-de-licencias-de-software-libre/`

Capítulo 3

Gestión de proyectos de software libre

3.1. Comunidades de software libre

Dadas las características del software libre, raramente una única empresa desarrolla un paquete de software, como es habitual con el software privativo. El software libre es desarrollado en el seno de comunidades de software libre.

Una comunidad de software libre es un grupo de personas que cooperan entre sí para desarrollar, mantener o difundir uno o varios paquetes de software libre. Las personas en una comunidad pueden compartir una localidad o pueden estar distribuidas por el mundo. Una comunidad puede estar estrictamente organizada a través de reglamentos escritos y seguidos a rajatabla, o ser un grupo *ad-hoc* sin demasiadas reglas. El liderazgo puede ser llevado adelante por una empresa, una organización, un grupo de personas o incluso una única persona.

A través del uso de Internet las comunidades de software libre se distribuyen alrededor del mundo y les permite estar compuestas por personas de distinto género, nacionalidad, conocimientos previos, formación académica, culturas, idiomas y motivaciones. Esta diversidad enriquece el desarrollo del software y ha permitido crear complejos productos informáticos, tales como sistemas operativos o compiladores, pero también hace compleja la gestión de los grupos de desarrollo.

Esta complejidad fue primero notada y analizada por Eric S. Raymond en una serie de artículos que luego se transformaron en su clásico libro “La catedral y el bazar” [Raymond, 2001], originalmente escrito en 1997, como un ensayo sobre la metodología de desarrollo del software de código abierto. En él analiza dos modelos de producción de software bien diferenciados. Por un lado, el modelo de desarrollo más hermético, planificado y vertical característico del software propie-

tario, al que compara con la construcción de una catedral, donde cada pieza es cuidadosamente ensamblada por trabajadores bajo la guía de el/la o los/las arquitectos/as según un plan previamente trazado, sin que haya lugar a lanzar versiones de prueba antes de que haya llegado el momento de inaugurar la obra. Por el otro lado el modelo de desarrollo del kernel Linux, asimiliado por Raymond a un bazar persa[1] por su dinámica horizontal, caótica y bulliciosa, sin una planificación ni guía vertical estricta, formada por personas con diferentes objetivos y enfoques. Sorprendentemente, de ese caos aparente surge un sistema coherente, estable y útil, ya sea un bazar o el kernel de un sistema operativo.

Raymond detectó ciertas características de liderazgo de Linus Torvalds que son fundamentales en el éxito de Linux y de otras comunidades de software libre. Una de ellas es lanzar versiones de prueba enseguida y a menudo. Bajo el lema "libere pronto, libere a menudo y escuche a los usuarios", el lanzamiento continuo de versiones mejoradas del sistema permite minimizar la redundancia de esfuerzos mediante la difusión rápida de correcciones ya realizadas, además de proveer un estímulo y recompensa constante a los usuarios (especialmente a los más ansiosos) que realizarán una constante revisión en cada versión, permitiendo detectar los errores de forma temprana y minimizando el costo de cada error.

Otra característica del liderazgo de Torvalds es delegar cuanto sea posible. Cualquier líder de una comunidad de software libre cuyo desarrollo dependa tan sólo de su cerebro va a estar siempre en desventaja frente al que sepa cómo crear un ambiente abierto y en evolución, en el cual tanto el desarrollo como la búsqueda de errores y las mejoras se confíen a cientos de personas.

Tratar a los usuarios como colaboradores permite mejorar con rapidez y depurar eficazmente un programa. Esto a su vez permite hacer crecer la comunidad. Un número mayor de usuarios encuentra más errores debido a que añade muchas más formas diferentes de forzar el programa. Dada una base lo suficientemente amplia de probadores y colaboradores, casi todos los problemas se identificarán con rapidez y su solución será obvia para alguien.

Para trabajar y competir con eficacia, quienes quieran desarrollar un proyecto en colaboración deben aprender a reclutar y motivar a la gente en base a intereses comunes y a conectar la individualidad de los/as colaboradores/as para llevarles a culminar objetivos difíciles solo alcanzables mediante una colaboración sostenida. Esto no quiere decir que la visión y la capacidad individual no importan. Los proyectos más exitosos de software libre son los que comienzan a partir de una idea y perserverancia de una persona pero que al mismo tiempo promueve la construcción de un grupo de trabajo con intereses comunes.

[1] Si se quiere argentinizar la metáfora podríamos pensar en una de las ferias conocidas como Salada o "saladitas".

3.2. Roles en la comunidad

Las personas que forman parte de una comunidad de software libre pueden cumplir varios roles: usuarios/as, diseñadores/as, desarrolladores/as, distribuidores/as, realizar soporte, traductores/as, instructores/as, entre otros. Sin embargo, existe una especie de categorización de los/as participantes: usuarios/as, contribuyentes[2] y mantenedores/as.

- Usuario/a: Quien comienza a utilizar un software libre con asiduidad se está uniendo a una comunidad de personas con un objetivo en común. Un/a usuario/a puede colaborar de muchas formas, tal como enviar informes de errores, realizar solicitudes de nuevas funcionalidades y promover el uso del software n ámbitos externos a la comunidad. Los/as usuarios/as proporcionan una muy necesaria retroalimentación a los contribuyentes y mantenedores.

- Contribuyente: Un/a contribuyente no solo reporta un error sino que investiga las causas y propone una solución o incluso la implementa. Para ser contribuyente no es necesario saber progrmar, se puede contribuir con código o con documentación. Puede que tengan voz en la toma de decisiones.

  Mantenedor: Los mantenedores son los/as líderes/sas del proyecto y comprometen mucho de su tiempo y energía en los proyectos. Normalmente, un mantenedor será el árbitro final de cualquier problema de diseño o codificación que surja dentro del proyecto. Es un título de prestigio, pero requiere una inversión superior en tiempo y esfuerzo. Son quienes deciden los pasos siguientes del proyecto y la orientación general de la comunidad.

Para pasar de usuario/a a contribuyente, e incluso avanzar en la complejidad e importancia de las contribuciones a un proyecto, existe un conjunto de pasos que se siguen, usualmente en forma escalonada, ya que cada paso requiere de mayor conocimiento del código del proyecto.

- Instalar el software en la computadora personal.
- Usar el software.
- Si es el caso, instalar el software en un servidor Web.
- Participar de los foros de discusión.
- Promover el software en ámbitos externos a la comunidad.

[2]Para las personas que contribuyen al proyecto generalmente se usa el término en inglés *contributors*, pero en este texto utilizaremos su versión en español.

- Realizar tareas de instrucción sobre el uso del software.
- Traducir, mejorar o crear documentación.
- Verificar las distintas versiones y reportar errores.
- Modificar el código para personalizar una operación o corregir un error.
- Crear un módulo para extender el software con una funcionalidad.
- Aportar un módulo para ser incorporado en la versión oficial del proyecto.

El primer paso para pasar de usuario a involucrarse más personalmente en un proyecto de software libre es contactar a los mantenedores personalmente o a través de los foros en línea de la comunidad, ofrecer sus servicios y preguntar qué es lo que la comunidad está necesitando más en ese momento.

3.3. Toma de decisiones

Arriba hemos mencionado que tanto contribuyentes como mantenedores/as aportan a la toma de decisiones dentro de la comunidad. En particular, el involucramiento en la toma de decisiones dependerá de varios factores, tanto personales como del proyecto.

- Experiencia e involucramiento de la persona en la comunidad: A más tiempo y dedicación a la comunidad, mayor será el nivel de responsabilidad y poder de toma de decisiones de una persona.
- Nivel de impacto de la decisión a tomar en el proyecto: Las decisiones más importantes son tomadas por un grupo de personas a cargo, generalmente decisiones respecto al *core*. Para decisiones de menor impacto las discusiones son menos, como la mejora de características de un paquete.
- Pertenencia a diferentes áreas del proyecto: Dentro de un proyecto de software existen diferentes áreas con (usualmente) diferentes responsables, por lo que ellos/as son quienes pueden tomar decisiones y guiar a sus miembros para llevarlas a cabo.

Más allá de las características personales de los/as miembros de la comunidad y de las características técnicas del software desarrollado, existen diferentes estilos de toma de decisiones utilizados por las comunidades de software libre.

- Monárquico: Donde las decisiones mas importantes del proyecto son llevadas a cabo por una persona o líder (por ejemplo, Linux).
- Comunitaria: Se toman las decisiones de manera democrática y no existe una persona o rol central en el proyecto (por ejemplo, PostgreSQL).
- Corporativa: El liderazgo del proyecto lo tiene una empresa privada o una fundación sin fines de lucro, y son ellas quienes toman las principales decisiones de manera formal (por ejemplo, Fedora de la empresa Red Hat o Firefox de la Fundación Mozilla).

Cabe destacar que entre los estilos Monárquicos y Comunitarios existe todo un espectro de posibles estilos intermedios, donde el número de quienes pueden participar (con voz o voto) en la toma de decisiones se va ampliando desde una única persona en el Monárquico a toda la comunidad en el Comunitario.

Además, otro aspecto importante en la toma de decisiones dentro de un proyecto de software libre es que en general el mérito dentro de la comunidad es tenido en cuenta. Aquellas personas que han contribuido de manera importante y que han ganado credibilidad con respecto a sus habilidades dentro de la comunidad son las que serán tenidas en cuenta a la hora de tomar una decisión, sea cual fuera la estructura de un proyecto.

3.4. Comunicaciones en la comunidad

Debido a la naturaleza de los proyectos de software libre y a que los participantes de estos pueden trabajar desde ubicaciones muy distantes, cobran una gran importancia los distintos medios de comunicación que se utilizan, tanto para coordinar las actividades de la comunidad, como para comunicar al exterior las novedades del proyecto.

La comunicación se hace principalmente a través de Internet, utilizando las siguientes herramientas.

- Sitio web: permite transmitir la información del proyecto al público en general y dar a conocer actualizaciones, decisiones y problemas.
- Listas de correo electrónico: permite la comunicación y discusión horizontal en la comunidad y es el más sencillo de implementar, pero es el más lento debido a que es asincrónico.
- Mensajería instantánea: sirven para la publicación de problemas, proponer soluciones y llevar adelante discusiones, permitiendo la participación sincrónica y horizontal.

La moderación de estos espacios de discusión es un tema muy importante en una comunidad de software libre y la falta de control puede llevar a que se tengan experiencias personales y profesionales muy negativas que lleven a la disgregación de la comunidad.

3.5. Herramientas de gestión

Existen dos canales de comunicación fundamentales en una comunidad de software libre que lleva adelante el desarrollo de un proyecto de software: el código fuente y las descripciones de *issues* (con esta palabra en inglés describimos tanto a los errores detectados como a las mejoras propuestas o a las nuevas funcionalidades deseadas para un software). Para su gestión existen herramientas digitales que permiten ordenar su uso, modificación y almacenamiento.

El código fuente de un software libre es requerido por definición y también para poder hacer el desarrollo a través de un grupo, generalmente distribuido globalmente. Para esto se utilizan servicios de infraestructura de alojamiento en la web, que no solo dan posibilidad de tener un lugar en el internet donde se puede tener acceso a las fuentes del proyecto, sino que brindan herramientas que facilitan la gestión del proyecto, tales como el manejo de permisos de modificación, publicación de *issues*, revisión de código propuesto y el almacenamiento de versiones anteriores.

En la actualidad los servicios más populares para este fin (aunque no son exclusivos para proyectos de software libre, ya que otros grupos o empresas pueden almacenar su código allí) son GitHub[3] y GitLab[4].

Estos servicios proveen, como dijimos, un conjunto completo de herramientas de gestión. En particular herramientas para gestión de *issues* y para control de versiones. Un *issue* es un problema o error detectado en el software o una tarea para realizar, ya sea una mejora, una corrección o el agregado de alguna funcionalidad al software. Dijimos previamente que uno de los roles de los/as usuarios/as avanzados/as en una comunidad de software libre era detectar errores en el software. A través de un sistema de gestión de *issues* los/as usuarios/as pueden reportar dichos errores en una forma normalizada, que exige la información mínima necesaria para que el error pueda ser reproducido y una solución propuesta por los/as contribuyentes. Además, información extra puede ser agregada o discutida por otros/as miembros de la comunidad, e incluso se pueden agregar rótulos que indican la prioridad o el tipo de error (por ejemplo, se los puede rotular como “crítico” o “adecuado para principiante”). Los repositorios de código mencionados proveen

[3] GitHub: `https://github.com/`
[4] GitLab: `https://gitlab.com/`

sus propios sistemas de gestión de *issues*, aunque también existen sistemas independientes, como por ejemplo Bugzilla[5].

Para cualquier proyecto de software, es muy importante la implementación de un sistema de control de versiones para poder gestionar los diversos cambios que se realizan sobre sus componentes. Esto cobra una importancia aún mayor en proyectos de software libre, ya que la heterogeneidad de su comunidad de contribuyentes requiere que las modificaciones al código tengan un control más estricto. Aunque existen muchos sistemas de control de versiones, tales como Subversion[6] (o SVN) y Mercurial[7], el más difundido en la actualidad es Git[8], creado originalmente por Linus Torvalds y que es la base de los dos repositorios mencionados (de ahí sus nombres).

3.6. Control de calidad

El enfoque actual para asegurar la calidad del software es utilizar procedimientos estándares durante el proceso de desarrollo, guiados por equipos de aseguranza de la calidad del software (QA, por las siglas de *Quality Assurance*). Implementar estos estándares es un desafío para las comunidades de software libre, ya que usualmente siguen un modelo horizontal al estilo bazar.

Algunas falencias comunes son la falta de pruebas unitarias asociadas al código presentado para su inclusión en el software siendo desarrollado, la inexistencia de ciclos de testing llevados adelante por un grupo o departamento de calidad independiente de los/as desarrolladores/as y la baja calidad de los reportes de defectos, con información incompleta o poco precisa, ya que usualmente son escritos por usuarios/as sin formación en informática. Actualmente, gracias al auge de los procesos de integración continua y a la integración de herramientas que los implementan en los repositorios, se han mejorado los primeros dos aspectos mencionados, ya que se exige que el código incluya pruebas unitarias y se ejecutan conjuntos de pruebas en forma automática antes de aprobar la solicitud de inclusión del código en el proyecto (usualmente llamado, en el ambiente de Git, *pull request*).

Respecto al bajo nivel de los reportes de defectos o *issues*, debe tenerse en cuenta que son el resultado de dos características positivas del software libre: el lanzamiento continuo de versiones mejoradas (las cuales aún tienen defectos conocidos o pueden tener nuevos defectos no detectados antes de su lanzamiento) y el involucramiento de toda la comunidad en la búsqueda de errores en el software. El

[5] Bugzilla: `https://www.bugzilla.org/`
[6] Apache Subversion: `https://subversion.apache.org/`
[7] Mercurial: `https://www.mercurial-scm.org/`
[8] Git: `https://git-scm.com/`

problema puede minimizarse dedicando tiempo de los/as contribuyentes y mantenedores/as a la revisión detallada de los defectos reportados, de modo que ningún defecto sea considerado como aceptado hasta que un/a desarrollador/a experimentado/a lo revise, reproduzca y apruebe.

Capítulo 4

Modelos de negocio con software libre

Ya debe haber quedado claro que el software libre no es necesariamente gratis, aunque, como dijimos previamente, el costo del código tiende a cero, por lo que el modelo de negocios tradicional en la industria del software no puede aplicarse a este tipo de software. Otra característica que surge del modelo de desarrollo de software libre es que es imposible, a diferencia del software propietario, obtener ventajas abusivas por posición dominante (característica conocida comúnmente como "monopolio"), ya que el código fuente está disponible para cualquier persona o empresa, y ante abusos del grupo de desarrollo se puede producir una división (a veces denominada *fork*) y generar un producto con las mismas características que el original.

En este capítulo presentaremos algunos de los modelos de negocios que pueden utilizarse en el marco del software libre o de fuentes abiertas, haciendo la aclaración que algunos de ellos pueden superponerse o combinarse en algunos proyectos.

4.1. Financiación pública

Siguiendo un modelo similar al de la financiación de la investigación básica, las instituciones públicas o fundaciones sin fines de lucro pueden proveer apoyo económico a proyectos de software libre. Según las características del proyecto y los objetivos de la institución financiante, puede que la institución participe o no en la gestión del proyecto y desarrollo del software.

Usualmente no busca recuperar la inversión, pero tiene objetivos claros. Por ejemplo, favorecer cierta industria, promover cierta tecnología o estándares, desarrollar herramientas de bajo costo para instituciones públicas, facilitar el acceso a

tecnología, etc.

Un caso cercano de este tipo es el desarrollo del sistema operativo Huayra, una distribución GNU/Linux, que es financiado por el gobierno argentino a través de un equipo de desarrollo propio[1]. Esta distribución, y algunas aplicaciones desarrolladas por el mismo equipo, es incluida en las computadoras del Plan Conectar Igualdad, que distribuye computadoras portátiles a estudiantes de escuelas secundarias públicas de todo el país[2].

4.2. Financiación privada

Algunos proyectos de software libre son llevados a cabo o financiados por empresas, es decir, instituciones con fines de lucro. En forma similar a la financiación anterior, tanto los objetivos como el involucramiento de la empresa en la gestión y desarrollo del proyecto pueden variar.

Un ejemplo interesante de estos dos tipos de financiación, y de cómo esta va variando a lo largo de la vida de un software, es el de la suite de oficina OpenOffice: el software privativo llamado StarOffice, fue adquirido en 1999 por la empresa Sun Microsystems para uso interno, pero en 2002 lo liberó como software libre con el nombre de OpenOffice para competir con la suite Microsoft Office y promover la utilización del estándar libre de documentación ODF (siglas de *Open Document Format*. En 2010 la empresa fue adquirida por Oracle Corporation y al año siguiente se informó que no se continuaría financiando el proyecto OpenOffice, el cual se donó a la Fundación Apache, que lo sigue desarrollando bajo el nombre de Apache OpenOffice[3]. Mientras tanto, un grupo independiente de la comunidad realizó un *fork* y creó la fundación The Document Foundation, la cual, en base al código de OpenOffice, desarrolla actualmente la suite LibreOffice[Watkins, 2018][4].

En las últimas décadas han surgido cooperativas (empresas gestionadas por sus trabajadores[5]) dedicadas a la tecnología basada en software libre o que desarrollan software libre. Lo mencionamos como un caso especial debido a que la filosofía del movimiento de software libre y los principios cooperativos tienen muchos puntos en común, tales como la pertenencia abierta y voluntaria a la comunidad o empresa, el control horizontal por parte de sus miembros, el compromiso con la

[1] Huayra GNU/Linux: `https://huayra.educar.gob.ar/`

[2] Plan Conectar Igualdad: `https://conectarigualdad.edu.ar/`

[3] Apache OpenOffice: `https://www.openoffice.org/es/`

[4] LibreOffice: `https://es.libreoffice.org/`

[5] Formalmente una cooperativa es una asociación autónoma de personas que se han unido voluntariamente para hacer frente a sus necesidades y aspiraciones económicas, sociales o culturales comunes por medio de una empresa de propiedad conjunta y democráticamente controlada.

comunidad, el compartir el conocimiento y el aporte del saber de cada miembro de la comunidad[Monk, 2014].

4.3. Financiación por quien necesita mejoras

Cuando un/a usuario/a o empresa requiere mejoras o funcionalidades específicas en un software libre puede financiar a un grupo (de la comunidad o ajeno a ella) para que las desarrolle. Cabe destacar que si el software mejorado resultante es para uso interno de la empresa no es requisito la redistribución libre del código generado, aunque el original tenga una licencia robusta. Sin embargo, si el software se redistribuye solo o como parte de un producto o servicio, una licencia robusta exigirá que la mejora (aún siendo pagada por la empresa) sea distribuida con la misma licencia que el código original. También es costumbre compartir el nuevo código aunque no sea un requisito legal, como aporte a la sociedad y en agradecimiento por el software original.

Este tipo de financiación permite que software desarrollado globalmente beneficie al mercado local, ya que usualmente se contrata a equipos de trabajo locales por una cuestión de cercanía, afinidad cultural y mayor simplicidad en los procesos comerciales y legales.

Una variación de este tipo de financiación es el aporte de tiempo de trabajo de desarrolladores/as de una empresa a un proyecto de software libre. Un grupo de desarrolladores/as de una empresa dedica su tiempo de trabajo, total o parcial, a desarrollar componentes o realizar mejoras a un software libre, formando parte de la comunidad. Este tiempo es pagado por la empresa como parte del sueldo de los/as desarrolladores/as. Este fue el caso de Corel, que en 1999 dedicó un equipo de ingenieros/as al proyecto Wine[6] para que le agregaran funcionalidades que permitiría que las aplicaciones de Corel, tales como WordPerfect, CorelDRAW y Quattro Pro, correr en Linux en forma eficiente y sin necesidad de reescribir el código de las aplicaciones [Corel, 1999].

4.4. Financiación por consultoría

Este tipo de negocios es similar al anterior, pero generalizado a cualquier tipo de actividad que requiera el cliente. Tanto puede ser realizar mejoras al software (arreglo de errores específicos, agregado de funcionalidades) como análisis para

[6]Wine (*Wine Is Not an Emulator*) es un software libre que implementa las API de Microsoft Windows en Linux, permitiendo que aplicaciones desarrolladas para Windows se ejecuten en un sistema Linux sin necesidad de ser recompiladas. `https://www.winehq.org/`

determinar si el software es el adecuado para la empresa o actividades de instalación de software particularmente complicado o en ambientes complejos.

El fundamento de este modelo de negocios es que las empresas usuarias de software libre generalmente ante un problema del software no pueden depender de los tiempos de la comunidad sino que requieren una solución inmediata. El firmar acuerdos de servicio técnico con empresas proveedoras le permite garantizar ciertos tiempos de respuesta adecuados para asegurar la continuidad del negocio.

Este es el enfoque de negocios de muchas de las empresas basadas en software libre más exitosas. El mayor ejemplo de esto es Red Hat[7], cuyo servicio de consultoría basado en una distribución del sistema operativo GNU/Linux, comenzando en 1993 la llevó a ser una empresa valuada en 33 mil millones de dólares, precio al cual la adquirió IBM en 2018.

4.5. Financiación por servicios web

Recientemente, y dependiendo de sus características, muchos proyectos de software libre ofrecen tanto una versión para instalar en los equipos de los/as usuarios/as como una versión en la nube, que provee servicios web con ciertos esquemas de pago (puede haber un servicio básico gratuito y mayores precios según los servicios provistos). Lo que se cobra en este caso no es el software, sino los servicios web (almacenamiento, gestión de usuarios, etc.).

Cabe notar que una ventaja de este enfoque comercial es que el/la usuario/a no necesita estar pendiente de las nuevas versiones, el software en la web se actualiza a medida que aquellas van siendo liberadas. Por otro lado, al ser software libre, si el/la usuario/a no está conforme con los servicios provistos puede descargarlo e instalarlo en sus equipos o en los equipos de un proveedor de servicios de nube y continuar usándolo sin mayores costos.

Un ejemplo de este tipo de negocio es Moodle[8], la plataforma de gestión de clases que utilizamos en esta asignatura. La instancia que utilizamos está instalada en los servidores del centro de cómputos de la UTN-FRC, pero instituciones que no tengan estas facilidades pueden adquirir diferentes planes y utilizar Moodle en la nube. En la Fig. 4.1 se muestran los diferentes planes al día de hoy.

4.6. Financiación por productos relacionados

Ya que no es razonable vender el código, una fuente de financiación de proyectos libres es la venta de productos asociados, ya sean tangibles o intangibles.

[7] Red Hat: `https://www.redhat.com/es`
[8] Moodle: `https://moodle.org/`

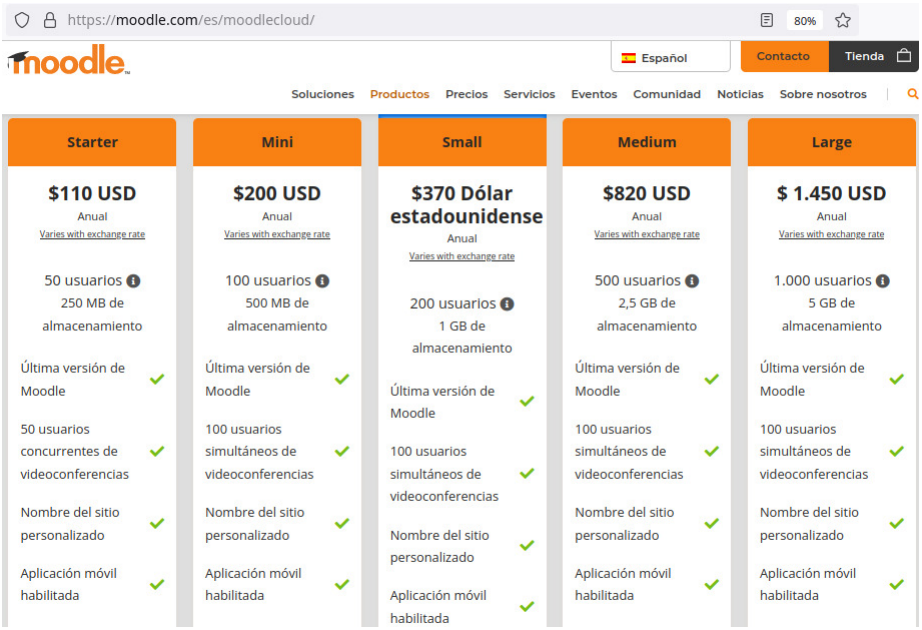

Figura 4.1: Planes de servicio de Moodle en la nube. Precios en dólares al 23/02/2022.

- ***Merchandising*:** Es común la venta de remeras, gorras, calcos y otro material sobre proyectos o comunidades de software libre. Para que los fondos obtenidos de esta manera beneficien a la comunidad que desarrolla el software es usual que la marca, nombre y/o logo del proyecto estén protegidos legalmente. Por ejemplo, Linux es una marca registrada de Linus Torvalds y Firefox es marca registrada de la Mozilla Foundation, así como el logo del navegador.

- **Formación:** Cursos, entrenamiento, manuales y libros sobre software libre también son fuentes de ingresos para las comunidades de software libre. Aunque cualquier persona o empresa puede desarrollar actividades de formación sobre un proyecto de software libre, usualmente quienes están involucrados en la comunidad están en ventaja para hacerlo, ya que poseen un conocimiento profundo del software en cuestión.

Como ejemplo de esto último mencionamos en particular a la editorial O'Reilly, que publica libros sobre diferentes proyectos de software libre, beneficiando generalmente a sus creadores originales. Debe notarse, además, que muchos de sus

libros a su vez tienen licencias libres[9].

4.7. Licenciamiento múltiple

En algunos casos, para financiar el proyecto libre el producto de software se distribuye bajo dos licencias o más, siendo al menos una de ellas libre y las otras propietarias. Generalmente las versiones bajo licencia propietaria tienen funcionalidades que la versión libre no tiene (pasado un tiempo esas funcionalidades se agregan a la versión libre, pero quienes pagan por la licencia propietaria pueden acceder antes a estas funcionalidades).

Usualmente las versiones bajo licencia propietaria incluyen servicios de consultoría y tiempos de respuesta rápidos a solicitudes del/la usuario/a.

Un clásico ejemplo, entre muchos, es la base de datos MySQL[10] que utiliza la licencia GPL para su *Community Edition* y una licencia propietaria de Oracle para su versión comercial.

4.8. Financiamiento por donaciones

Algunos proyectos de software libre se financian (al menos parcialmente) a través de donaciones de particulares y empresas. Para facilitar este tipo de financiación se han creado fundaciones que gestionan las finanzas del proyecto (como vimos en la sección 4.2), aunque esto no es requisito y dependerá de los montos de donaciones que se manejan en cada proyecto.

4.9. A modo de cierre

Hemos visto en este capítulo diferentes modos de financiar (total o parcialmente) un proyecto de software libre. Cabe aclarar que estos modelos de negocios no son excluyentes, muchas comunidades se financian a través de una combinación de estos.

[9] O'Reilly Open Books: `https://www.oreilly.com/openbook/`
[10] MySQL: `https://www.mysql.com/`

Capítulo 5

Sistemas operativos de software libre

Un sistema operativo libre es, como todo software libre, aquél que puede ser copiado, estudiado, modificado, utilizado libremente con cualquier fin y redistribuido con o sin cambios o mejoras. En este capítulo presentaremos algunos sistemas operativos libres y su historia, ya que son fundamentales en el *stack* de software libre.

5.1. Historia de los SO derivados de Unix

Los primeros sistemas operativos surgen en la década de 1950, pero la revolución es este campo ocurre durante los '60. Allí aparecen conceptos como sistemas multitareas, multiusuarios, multiprocesadores y de tiempo real. En esta década emerge Unix, base de la mayoría de los sistemas operativos libres actuales.

Unix fue desarrollado desde 1969 por Ritchie, Thompson y McIlroy en los laboratorios Bell de AT&T. Es un sistema operativo portable, multitarea y multiusuario originalmente destinado a mainframes. Se caracteriza por poseer un núcleo monolítico y estar originalmente escrito en C (lenguaje a su vez creado por Ritchie).

Debido a que un fallo judicial antimonopolio impedía a la empresa de telecomunicaciones AT&T comercializar software, Unix no fue comercializado inicialmente y se distribuía a las universidades con una licencia muy laxa. A partir de estas versiones de Unix varias universidades crearon sus propios sistemas operativos, el más exitoso y duradero es el sistema operativo libre BSD (por *Berkeley Software Distribution*)[1] realizado en la Universidad de Californa en Berkeley.

[1] BSD: `https://www.bsd.org/`

Por el problema de licencias mencionado y sus fortalezas técnicas, muchos sistemas operativos, libres o privativos, son variaciones de Unix. En la Fig. 5.1 vemos el árbol de sistemas operativos derivados de Unix, identificados por color según su esquema de licenciamiento: verde para fuentes abiertas, naranja pálido para los sistemas mixtos, con fuentes compartidas (pero no necesariamente abiertas o libres), y rosa para los sistemas privativos.

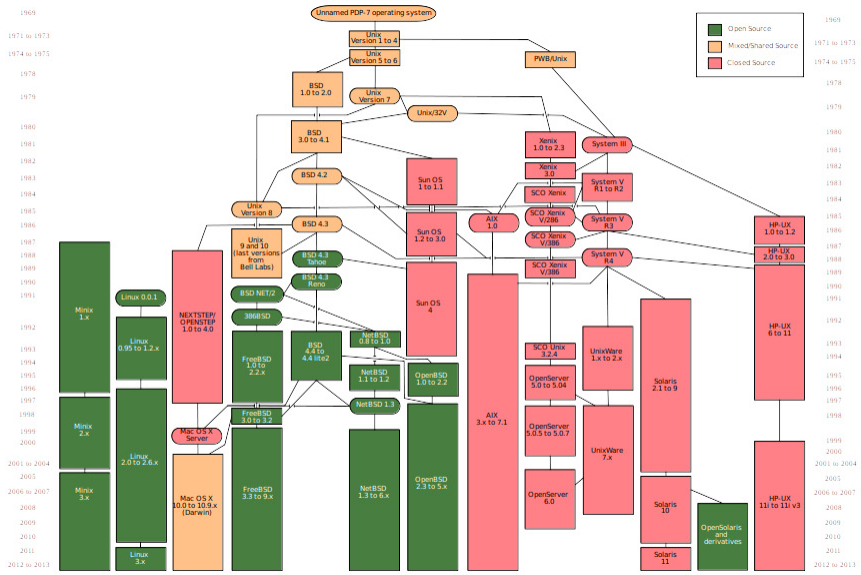

Figura 5.1: Sistemas operativos derivados de Unix.

Interesantemente, en las columnas de la izquierda podemos ver dos linajes derivados pero que no están conectados por una línea a Unix: Minix y Linux. La falta de conexión indica que, aunque derivados de Unix en el sentido de que siguen el estándar POSIX[2], el código de estos sistemas operativos fue escrito desde cero.

En el caso particular de Minix (llamado así por ser un MINI-uniX), este sistema operativo está basado en la tecnología de *microkernel*, a diferencia de todos los otros sistemas de este árbol, que presentan un kernel monolítico. Otra particularidad de Minix es que durante mucho tiempo el código fuente impreso era parte del libro "Sistemas Operativos: Diseño e Implementación" [Tanenbaum and Woodhull, 1987], lo que hacía que el libro tuviera una longitud de 719 páginas. Ediciones posteriores eran acompañadas por versiones de Minix en diskettes o CD. Actualmente la

[2]Esto quiere decir que se comportan igual que Unix, respondiendo a los mismos comandos de la misma forma. Estándar POSIX: `https://es.wikipedia.org/wiki/POSIX`

versión 3 se puede obtener de un sitio web oficial[3].

5.2. Distribuciones Linux

Como se mencionó en el capítulo 1, los sistemas operativos libres más exitosos actualmente están basados en el núcleo (o *kernel*) Linux. En vez de hablar de sistemas operativos diferentes, usualmente se los menciona como una "distribución Linux" (coloquialmente llamada *distro*): una distribución es un sistema operativo basado en el núcleo Linux que incluye paquetes de software extra que le permiten satisfacer las necesidades de un grupo específico de usuarios.

Una distribución Linux contiene típicamente el núcleo, herramientas y bibliotecas, software adicional, documentación, un sistema de ventanas, un administrador de ventanas y un entorno de escritorio. La mayor parte del software incluido es libreo o de fuentes abiertas y, por lo tanto, distribuido en binario compilado y en código fuente, permitiendo a sus usuarios modificar o compilar el código fuente original si lo desean. Muchas distribuciones incorporan software privativo, no disponible en forma de código fuente.

Si las distribuciones incluyen habitualmente las bibliotecas y herramientas del proyecto GNU, se la denomina GNU/Linux[4]). El sistema de ventanas más utilizado es X Window System, y los entornos de escritorio más comunes son: Gnome[5], KDE[6], XFCE[7] y Mate[8].

Aunque son todas libres, existen distribuciones que están soportadas comercialmente, como Fedora[9] (por Red Hat), openSUSE[10] (por Novell) o Ubuntu[11] (por Canonical), mientras que otras son mantenidas por una comunidad, como Debian[12] y Gentoo[13]. Incluso existen distribuciones que no están relacionadas con

[3]Minix 3: `http://www.minix3.org/`

[4]La FSF (*Free Software Foundation*) promueve el nombre de GNU/Linux para el sistema operativo en vez de solamente usar Linux, a fin de reconocer el aporte del proyecto GNU al sistema operativo, que no es sólo su núcleo o kernel, sino también controladores, aplicaciones de usuario y otros paquetes de software. Si bien el núcleo es el componente más grande del sistema, la sumatoria de todos los componentes GNU son mayores. Por su parte, quienes respaldan el uso del término Linux, dicen que es más corto y más fácil de nombrar. Además, argumentan que si se menciona uno de los colaboradores, se deberían mencionar todos.

[5]Gnome: `https://www.gnome.org/`

[6]KDE: `https://kde.org/`

[7]XFCE: `https://www.xfce.org/`

[8]Mate: `https://mate-desktop.org/`

[9]Fedora: `https://getfedora.org/`

[10]openSUSE: `https://www.opensuse.org/`

[11]Ubuntu: `https://ubuntu.com/`

[12]Debian: `https://www.debian.org/`

[13]Gentoo: `https://www.gentoo.org/`

ninguna empresa o comunidad, sino con un único desarrollador, como es el caso de Slackware[14] (también considerada como la distribución más antigua aún siendo mantenida).

Dado que seleccionando diferentes paquetes de software y empaquetándolos con sistemas automáticos se puede crear una nueva distribución a medida (de una empresa, un grupo o incluso una persona) en forma relativamente simple, es muy difícil llevar la cuenta de cuántas distribuciones Linux existen. El sitio DistroWatch[15] realiza un seguimiento de las distribuciones creadas y analiza si están activas o discontinuadas, además de publicar noticias sobre nuevas versiones, análisis de distribuciones y paquetes de software. Otra lista de distribuciones puede encontrarse en Wikipedia[16].

5.2.1. Clasificación de las distribuciones

Como puede verse en la página *List of Linux distributions* de Wikipedia, las distribuciones Linux se clasifican usualmente en base al sistema de gestión de paquetes que utilizan y a la distribución original de la cual derivan.

Cada paquete de software contiene una aplicación específica o un servicio (por ejemplo, una biblioteca para manejar el formato de imagen PNG, una colección de tipografías o un navegador web), la cual es generalmente distribuida en su versión compilada y su instalación (y desinstalación) es controlada por un "sistema de gestión de paquetes". Los paquetes elaborados para un sistema particular de paquetes contienen metadatos (tales como fecha de creación, descripción y dependencias) que son reconocidos por el sistema y utilizados para la búsqueda de paquetes correspondientes a las dependencias, por ejemplo. Algunos de los sistemas de paquetes más usados son:

- RPM, creado por Red Hat y usado por un gran número de distribuciones de Linux, es el formato de paquetes del *Linux Standard Base*.
- DEB, originalmente introducido por Debian, pero también utilizados por otras distribuciones muy conocidas, como Knoppix[17] y Ubuntu.
- .tgz, usado por Slackware, empaqueta el software usando tar y gzip. Pero no reporta los errores de dependencias encontrados, por lo que se utilizan algunas herramientas de más alto nivel para tratar con este formato: slapt-get, slackpkg y swaret.

[14] Slackware: `http://www.slackware.com`

[15] DistroWatch: `https://distrowatch.com/`

[16] *List of Linux distributions* (la versión en español no es tan completa): `https://en.wikipedia.org/wiki/List_of_Linux_distributions`

[17] Knoppix: `http://knoppix.net/`

Entonces, la grilla de clasificación de distribuciones, considerando el sistema de paquetes que utilizan y la distribución original de la cual deriva es:

- RPM-based
  - CentOS/RHEL-based
  - Fedora-based
  - openSUSE-based
  - urpmi-based
  - apt-rpm based
  - Independent RPM distributions
- DEB-based
  - Debian-based
  - MEPIS-based
  - Knoppix-based
  - Ubuntu-based
- Arch-based
- Gentoo-based
- Slackware-based

5.2.2. ¿Android es una distro Linux?

Android, el sistema operativo más común en los teléfonos móviles, incluye un kernel Linux. Sin embargo, formalmente, existen diversas versiones de Android, en su mayoría desarrolladas para plataformas de hardware específicas, por lo que no pueden ser instaladas en cualquier computadora o teléfono móvil. Esta característica hace que no sean consideradas "distribuciones". Aunque existen algunos proyectos, tales como Android-x86[18], que pueden ser instalados en hardware genérico, por lo que son usualmente consideradas como distribuciones.

[18] Android-x86: `https://www.android-x86.org/`

5.3. BSD (*Berkeley Software Distribution*)

Berkeley Software Distribution (BSD)[19] es una familia de sistemas operativos derivada del sistema Unix. Tal como dijimos previamente, nació a partir de los aportes realizados a Unix por docentes e investigadores de la Universidad de California en Berkeley (Estados Unidos). Debido a la prohibición que pesaba sobre la empresa de telecomunicaciones AT&T de vender sistemas informáticos, en los primeros años del sistema Unix sus creadores, los Laboratorios Bell de AT&T, distribuyeron copias en varias universidades y les autorizaron a utilizar el código fuente y adaptarlo a sus necesidades.

Durante los años 1970 y 1980 Berkeley utilizó el sistema para sus investigaciones en materia de sistemas operativos. Cuando AT&T retiró el permiso de uso a la universidad por motivos comerciales, la universidad promovió la creación de una versión inspirada en el sistema Unix utilizando los aportes que ellos habían realizado, permitiendo luego su distribución con fines académicos y finalmente eliminando toda restricción respecto a su copia, distribución o modificación. Tales derechos se plasmaron en la licencia del sistema BSD, la cual no incluye la obligatoriedad de que los trabajos derivados sean distribuidos con la misma licencia (es decir, es una licencia "permisiva").

Esto posibilita, además, que su código fuente sea incorporado legalmente en software no libre. Así, otros sistemas operativos, tanto libres como propietarios, han incorporado código BSD. Por ejemplo, Microsoft Windows utiliza código derivado de BSD en su implementación de TCP/IP.

5.3.1. Versiones de BSD

Las diferentes versiones (o *flavors* en inglés[20]) del sistema operativo BSD fueron aportando componentes importantes a la familia Unix de sistemas operativos y también generaron una rama muy importante de sistemas derivados que aún seguimos utilizando.

- BSD 1 era un añadido a la sexta edición de Unix. Más que un sistema operativo completo era un Unix con el agregado de un compilador Pascal y un editor de texto.

- BSD 2 fue lanzada en 1978 e incluía dos nuevos programas que perduran en los sistemas Unix hasta hoy en día: el editor de textos `vi` y el *shell* del lenguaje C.

[19] BSD: `https://www.bsd.org/`

[20] La traducción de *flavors* al idioma español es "sabores".

- BSD 3 fue la primera distribución considerada como un sistema operativo completo. Era una modificación del novedoso Unix 7.

- BSD 4.3 incorporó la licencia permisiva libre BSD.

A partir de este linaje de versiones se derivaron varios sistemas operativos relevantes:

- SunOS[21] de la empresa Sun Microsystems. Utilizado en los poderosos servidores que esa compañía manufacturaba hasta principios de los '90.

- Variaciones de BSD para distintos públicos o usos: FreeBSD[22] para servicios web, NetBSD[23] para portabilidad, TrueOS (orignalmente llamado PC-BSD) para computadoras personales[24] y OpenBSD[25] basado en la seguridad.

- NeXTSTEP[26] de la empresa NeXT de Steve Jobs, que derivó a su vez en Mac OS X[Michán, 2010].

5.4. ReactOS

ReactOS[27] es un sistema operativo de código abierto que trata de emular la arquitectura de Windows NT. Este sistema operativo comenzó a desarrollarse en 1995 con el nombre de FreeWin95 pues era un clon de Windows 95. A partir de 1998 cambió su nombre a ReactOS y ha continuado con la incorporación gradual y permanente de características de las últimas versiones de Windows. Está principalmente escrito en C, con algunos elementos, tales como ReactOS Explorer y la pila del sonido, escritos en C++. Las licencias de este sistema operativo son: GNU GPL, LGPL y BSD.

El objetivo de ReactOS es el de proveer un sistema operativo el cual sea compatible a nivel binario con Windows y brindar una interfaz familiar para los y las usuarias. De este modo, podrán usar fácilmente este sistema como una alternativa libre a Windows, sin tener que cambiar el software al que están acostumbrados.

[21] SunOS: `https://es.wikipedia.org/wiki/SunOS`
[22] FreeBSD: `https://www.freebsd.org/es/`
[23] NetBSD: `https://netbsd.org/`
[24] TrueOS o PC-BSD: `https://es.wikipedia.org/wiki/TrueOS`
[25] OpenBSD: `https://www.openbsd.org/`
[26] NeXTSTEP: `https://es.wikipedia.org/wiki/NEXTSTEP`
[27] ReactOS: `https://reactos.org/`

5.5. Sistemas operativos para móviles

Además de los servidores, las computadoras personales y las portátiles, el dispositivo computacional más difundido hoy en el mundo son los teléfonos móviles o "inteligentes" (de *smartphone*, en inglés). Como ya se mencionó, el sistema operativo más utilizado en estos dispositivos es Android, el cual está basado en el núcleo Linux. Sin embargo, Android no es software libre y no existen a la fecha sistemas operativos libres para teléfonos móviles que tenga una porción de mercado mínimamente relevante.

El éxito de Android se debió a que fue el sistema operativo seleccionado por la *Open Handset Alliance* (OHA)[28], la cual es una alianza comercial de 86 compañías que desarrolla estándares para dispositivos móviles. Algunos de sus miembros son Google, HTC, Dell, Intel, Motorola, Qualcomm, Texas Instruments, Samsung, LG, T-Mobile, Nvidia y AMD.

No faltaron intentos de producir sistemas operativos libres para teléfonos móviles, entre los que recordamos a Firefox OS[29], Tizen[30] y Ubuntu Touch[31], pero ninguno de ellos fueron comercialmente exitosos.

[28] OHA: `https://www.openhandsetalliance.com/`
[29] Firefox OS: `https://support.mozilla.org/es/products/firefox-os`
[30] Tizen: `https://www.tizen.org/`
[31] Ubuntu Touch: `https://ubuntu-touch.io/es/`

Capítulo 6

Las reglas en una comunidad

En toda comunidad se deben tener reglas, y como tal, las comunidades de software libre no son la excepción. En particular, las comunidades de software libre suelen estar distribuidas globalmente, por lo cual se deben armonizar las diferentes costumbres y expectativas de comportamiento para las personas participantes de un proyecto.

En principio podríamos dividir las reglas de una comunidad de software libre en dos aspectos: código de conducta y reglas de diseño y programación. En las siguientes secciones cubriremos someramente estos aspectos, que a veces se combinan en un único conjunto de reglas.

6.1. Código de conducta

Se aconseja establecer un código de conducta tan pronto como sea posible. Idealmente debe hacerse al comenzar el proyecto y crear la comunidad, y debe establecerse por escrito, usualmente en un archivo denominado `codigo_de_conducta` en el directorio raíz del proyecto y enlazar al mismo desde el archivo `Read.me` para que se encuentre visible a toda la comunidad.

Algunas reglas de códigos de conducta que recurrentemente vemos en las mejores comunidades son las siguientes.

- Respetar las personas que forman la comunidad, cualquiera sea su rol o nivel de participación.
- Debatir y estudiar cualquier cambio significativo antes de comprometerse.
- En caso de duda sobre cualquier procedimiento, preguntar primero a personas más experimentadas (aunque, si existen historiales de discusiones pre-

vias, primero debe asegurarse de que no sea una pregunta ya contestada o discusión ya concluida).

- Ejecutar pruebas en los cambios del código antes de proponerlos.
- No asumir compromisos que no se podrán cumplir.
- Documentar el trabajo realizado.
- Informar regularmente a otros del progreso propio.

Además de comunicar las expectativas de comportamiento, un código de conducta debería describir cómo se aplica y qué sucede en caso de que no se cumplan dichas reglas.

- Dónde tendrán efecto las reglas de conducta: en las actividades técnicas del proyecto (*issues* y *pull requests*), en las comunicaciones (foros, emails, chat) y/o en actividades presenciales de la comunidad (eventos).
- A quién o a quiénes se les aplicará el código de conducta (es decir, a quiénes se les considera parte de la "comunidad").
- De qué manera puede reportarse una violación del código de conducta.
- Quién (o quiénes) determinan si efectivamente ha ocurrido una violación del código de conducta.
- Qué ocurrirá en caso de que alguien viole el código de conducta.

Estos tres últimos puntos son claves. Debe explicarse de qué manera se reportan violaciones al código, quién determina si fue una violación y cuáles serán las consecuencias para quien lo haya hecho. Esto demostrará la intención de tomar acciones cuando sea necesario, hará sentir a la comunidad más segura respecto a que sus reclamos son revisados, a la vez que brindará a la comunidad la seguridad de que el proceso de revisión es justo y transparente.

Luego de recibida una denuncia, las personas responsables de analizarla deben recolectar toda la información posible acerca de la situación. Hay que considerar, entre otras cuestiones, si la persona reportada lo es por primera vez o si es reincidente (para lo cual debería mantenerse un registro de denuncias y sanciones).

Luego de recolectar y procesar suficiente información, las personas a cargo deben decidir qué acción tomar, en base al código de conducta (nunca tomar decisiones por fuera de dicho código). Se debe tener en mente que el objetivo es fomentar un ambiente seguro, respetuoso y colaborativo en la comunidad, no castigar por el

solo hecho de mostrar que se puede o tomar revancha por algún problema previo. Las acciones a tomar pueden ir desde una advertencia pública a la persona en falta, explicando cómo su comportamiento ha impactado negativamente en los demás y cómo se espera que actúe en el futuro, hasta, en casos de extrema gravedad y reincidencia, expulsarla de la comunidad en forma temporal o permanente.

6.2. Reglas de diseño

Las reglas de diseño y programación sirven de guía para el desarrollo en general, guiando a la comunidad en la toma de decisiones al resolver una tarea y para minimizar el esfuerzo de leer e interpretar el código escrito por otros miembros, ya que la mayoría del tiempo de desarrollo se estará modificando, corrigiendo o mejorando código ya escrito.

Como en muchos otros ámbitos, en el desarrollo de software existen diferentes caminos posibles para resolver un problema y cada uno de ellos tendrá sus ventajas y desventajas. Las reglas de diseño intentan ser una guía para que la comunidad pueda evaluar las decisiones propuestas durante la resolución de un problema.

Este tipo de reglas son comunes en cualquier equipo de desarrollo, ya sea de software libre o privativo, de empresas o comunidades, de grupos localizados o de grupos globales; pero al ser las comunidades de software libre grupos heterogéneos y globalizados, es muy recomendable poner estas reglas por escrito lo antes positle. Algunas reglas de diseño y programación que recurrentemente vemos en los mejores equipos de desarrollo son las siguientes.

1. Modularidad: escribir partes simples conectadas por interfaces simples. La modularidad es la descomposición de un sistema complejo en piezas más simples, llamadas “módulos”. De esta manera el código fuente de un módulo puede ser escrito y mantenido independientemente del código fuente de otros módulos.

2. Claridad: la premisa consiste en ser claro a la hora de escribir código. La claridad en un programa es muy importante, ya que la mayor parte del tiempo de mantenimiento del mismo se emplea en estudiar y comprender el código existente.

3. Economía: el tiempo del programador es caro y es por ésto que hay que conservarlo y buscar ser lo más eficiente posibles.

4. Separación: consiste en separar las reglas del funcionamiento y las interfaces del mecanismo. Se recomienda realizar una separación en capas que se encarguen de distintas funciones.

5. Simplicidad: diseñar para la simplicidad, añadiendo complejidad sólo donde sea estrictamente necesario. Escribir código complejo sólo cuando sea evidente que no existe otra solución posible.

6. Transparencia: diseñar para la visibilidad, para hacer más fácil la inspección y la corrección de fallos.

7. Representación: convierte el conocimiento en datos, para que la lógica de los programas pueda ser sencilla y robusta.

8. Mínima Sorpresa: cuando se usa un sistema, se construye un modelo mental de la manera en que el mismo actuará, y si el sistema se comporta de manera inesperada, las personas que lo utilizan se confundirán.

9. Silencio: cuando un programa no tenga nada necesario que decir a la persona que lo usa, no debería decir nada.

10. Reparación: cuando se produzca un error, el mismo debe ser claro y mostrarse lo antes posible.

11. Generación: evitar hacer cosas a mano; escribir programas que escriban programas siempre que se pueda.

12. Diversidad: desconfiar de las afirmaciones tipo “esta es la única forma de hacerlo”, todos los problemas pueden resolverse de diferentes maneras.

13. Extensibilidad: diseñar para el futuro, que el programa tenga bases sólidas desde el comienzo que le permitan crecer.

6.3. Reglas de programación

Por su parte, las reglas de programación generalmente indican qué “estilo” darle al código, como por ejemplo la cantidad y tipo de caracteres que se utilizan para indentar (espacios o *tabs*), cómo alinear las distintas partes de las estructuras del código y convenciones respecto de los nombres utilizados para las variables.

Estas reglas no cambian en sí mismo el programa resultante (como sí lo hacen las reglas de diseño), pero sí facilitan la lectura del código entre los integrantes de la comunidad, algo clave cuando los grupos son tan heterogéneos como en las comunidades globales de software libre.

Existen herramientas automáticas que permiten evaluar si un código aportado a un proyecto cumple con las reglas de programación. Usualmente, además de ejecutar los tests funcionales, es obligatorio ejecutar estas herramientas antes de hacer un *pull request*.

6.4. Ejemplos

En esta sección presentamos códigos de conducta y reglas de diseño y programación de algunas comunidades, a modo de ejemplo. Unificamos los ejemplos de reglas de conducta, de diseño y de programación en una única sección porque, como se verá a continuación, las reglas de las comunidades de software libre muchas veces combinan los tres tipos de reglas.

6.4.1. Código de conducta de Debian

El código de conducta de la distribución "madre" Debian[1] es el siguiente.

1. **Sea respetuoso.** En un proyecto del tamaño de Debian, inevitablemente habrá personas con las cuales estará en desacuerdo, o le resulte difícil cooperar. Acéptelo, y aun así, siga siendo respetuoso. El desacuerdo no es excusa para un comportamiento pobre o ataques personales y una comunidad donde la gente se sienta amenazada no es una comunidad sana.

2. **Asuma buena fe.** Los contribuidores de Debian tienen muchas maneras de alcanzar nuestro objetivo común de un sistema operativo libre, que pueden diferir de sus maneras. Asuma que las otras personas están trabajando hacia este objetivo.

   Tenga en cuenta que muchos de nuestros contribuidores no son hablantes nativos de inglés, o pueden tener distintos rasgos culturales.

3. **Sea colaborador.** Debian es un proyecto grande y complejo; siempre hay algo que aprender dentro de Debian. Es bueno pedir ayuda si la necesita. Igualmente, las propuestas de ayuda deben verse en el contexto de nuestro objetivo común de mejorar Debian.

   Cuando haga algo para beneficio del proyecto, esté dispuesto a explicar a otros cómo funciona, así pueden construir en base a su trabajo, e incluso mejorarlo.

4. **Intente expresarse concisamente.** Tenga presente que lo que escribe una vez lo leerán cientos de personas. Escribir un correo electrónico corto significa que la gente pueda entender la conversación de la manera más eficientemente posible. Cuando se necesita una explicación larga, considere añadir un resumen.

[1] Código de conducta de Debian: `https://www.debian.org/code_of_conduct.es`

Intente proporcionar argumentos nuevos a una conversación, para que cada correo agregue algo único al hilo, teniendo presente que el resto del hilo sigue conteniendo los otros mensajes con los argumentos que ya se han dado.

Intente mantenerse dentro del tema, especialmente en discusiones que ya son bastante largas.

5. **Sea abierto.** Muchas vías de comunicación usadas en Debian permiten comunicación pública y privada. Según el párrafo tres del contrato social, debería usar preferentemente métodos públicos de comunicación para mensajes relacionados con Debian, a no ser que publique información sensible.

   Esto también se aplica para mensajes de ayuda o soporte relacionados con Debian; una petición pública de soporte no sólo le proporcionará una respuesta con más probabilidad, sino que también posibilita que los errores inadvertidos que cometan las personas que responden a su pregunta se detecten más fácilmente y se corrijan.

6. **En caso de problemas.** Aunque los participantes deberían adherirse a este código de conducta, reconocemos que a veces las personas pueden tener un mal día, o no estar al tanto de alguna de las directrices de este código de conducta. Cuando eso ocurre, puede responderles y dirigirles a este código de conducta. Esos mensajes pueden ser públicos o privados, como sea más apropiado. Sin embargo, ya sea el mensaje público o privado, debería seguir las partes relevantes de este código de conducta; en particular, no debería ser abusivo o irrespetuoso. Asuma buena fe, es más probable que los participantes no se den cuenta de su mal comportamiento que que intenten conscientemente degradar la calidad de la discusión.

   Se inhabilitará temporal o permanentemente la comunicación por cualquiera de los medios de Debian a los ofensores graves o reincidentes. Las quejas deben dirigirse (en privado) a los administradores del foro de comunicación de Debian en cuestión. Para encontrar información de contacto sobre los administradores, vea la página de la estructura organizativa de Debian.

6.4.2. Estándares de codificación de GNU

El proyecto GNU de la FSF tiene su estándar de codificación[2] a fin de obtener un sistema GNU limpio, consistente y fácil de instalar. Proponen que se lea como una guía para escribir programas C en forma portable, robusta y confiable. Pero no solo se refiere a cómo escribir programas sino también a cómo asignarle números

[2]GNU Coding Standards: `https://www.gnu.org/prep/standards/standards.html`

a las versiones o cómo referenciar código y documentación que no tengan licencias libres. A modo de ejemplo, incluimos sólo una subsección de reglas (la 5.1), traducidas del original en inglés en forma libre por los y las autoras de este manual.

5.1 Formato de su código fuente

Por favor, mantenga la longitud de las líneas de código en 79 caracteres o menos, a fin de maximizar la legibilidad en la amplia gama de entornos.

Es importante poner la llave de apertura que comienza el cuerpo de una función C en la columna 1, de modo que comience la definición de la función. Varias herramientas buscan la llave de apertura en la columna 1 para encontrar el comienzo de la función. Estas herramientas no funcionarán con código que no tenga ese formato.

Evite poner llaves, paréntesis o corchetes de apertura en la columna 1 cuando están dentro de una función. La llave de apertura que comienza el cuerpo de una estructura puede ir en la columna 1 si encuentra útil tratarla como la definición de una función.

También es importante poner, en la definición de una función, su nombre en la columna 1. Esto ayuda a las personas a buscar por definiciones de funciones, y puede ayudar a ciertas herramientas a reconocerlas. Así, usando sintaxis estándar de C, el formato de `concat` es:

```
static char *
concat (char *s1, char *s2)
{
  ...
}
```

O bien, si desea utilizar la sintaxis C tradicional:

```
static char *
concat (s1, s2)        /* Name starts in column one here */
     char *s1, *s2;
{                     /* Open brace in column one here */
  ...
}
```

En C estándar, si los argumentos no entran bien en una línea, se dividen así:

```
int
lots_of_args (int an_integer, long a_long, short a_short,
              double a_double, float a_float)
...
```

Para tipos `struct` y `enum`, también deben ponerse las llaves en la columna 1, a menos que todo el contenido entre en una única línea:

```
struct foo
{
  int a, b;
}
```

O bien:

```
struct foo { int a, b; }
```

El resto de esta sección da recomendaciones sobre otros aspectos del estilo de formato para C, el cual también es el estilo por defecto para el programa en su versión 1.2 o superior. Corresponde a las opciones:

```
-nbad -bap -nbc -bbo -bl -bli2 -bls -ncdb -nce -cp1 -cs -di2
-ndj -nfc1 -nfca -hnl -i2 -ip5 -lp -pcs -psl -nsc -nsob
```

No creemos que estas recomendaciones sean requisitos, porque no se producen problemas para los usuarios si dos programas diferentes tienen distintos estilos de formato.

Pero cualquiera sea el estilo que uses, por favor, úsalo consistentemente, ya que una mezcla de estilos dentro de un mismo programa tiende a verse feo. Si estás aportando cambios a un programa existente, por favor sigue el estilo de dicho programa.

Para el cuerpo de una función, nuestro estilo recomendado luce como este:

```
if (x < foo (y, z))
  haha = bar[4] + 5;
else
  {
    while (z)
      {
        haha += foo (z, z);
        z--;
      }
    return ++x + bar ();
  }
```

Encontramos que es más fácil leer un programa cuando tiene espacios antes de los paréntesis de apertura y después de las comas. Especialmente después de las comas.

Cuando divides una expresión en múltiples líneas, divídelas antes de un operador, no después. Esta es la forma correcta:

```
if (foo_this_is_long && bar > win (x, y, z)
    && remaining_condition)
```

Intenta evitar tener dos operadores de precedencia diferente al mismo nivel de indentación. Por ejemplo, no escribas esto:

```
mode = (inmode[j] == VOIDmode
        || GET_MODE_SIZE (outmode[j]) > GET_MODE_SIZE (inmode[j])
        ? outmode[j] : inmode[j]);
```

En vez de esto, utiliza paréntesis extra, de modo que la indentación muestre el anidamiento:

```
mode = ((inmode[j] == VOIDmode
         || (GET_MODE_SIZE (outmode[j]) > GET_MODE_SIZE (inmode[j])))
        ? outmode[j] : inmode[j]);
```

Inserta paréntesis extra así Emacs[3] indentará el código adecuadamente. Por ejemplo, la indentación siguiente luce bien si la haces a mano:

```
v = rup->ru_utime.tv_sec*1000 + rup->ru_utime.tv_usec/1000
    + rup->ru_stime.tv_sec*1000 + rup->ru_stime.tv_usec/1000;
```

Pero Emacs la alteraría. Agregando un conjunto de paréntesis produce algo que luce igual de bien, y que Emacs preservará:

```
v = (rup->ru_utime.tv_sec*1000 + rup->ru_utime.tv_usec/1000
     + rup->ru_stime.tv_sec*1000 + rup->ru_stime.tv_usec/1000);
```

Formatea las sentencias `do-while` de esta forma:

```
do
  {
    a = foo (a);
  }
while (a > 0);
```

[3]Emacs es un editor de textos desarrollado y propuesto por el proyecto GNU: `https://www.gnu.org/software/emacs/`

Por favor, usa Control-L para dividir el programa en páginas en lugares lógicos (pero no dentro de una función). No importa cuán largas son las páginas, dado que no tienen que entrar en una página impresa.

6.4.3. Reglas del núcleo Linux

Los/as programadores/as del núcleo Linux tienen 20 reglas muy estrictas[4]. Debido a su longitud, aquí solo mostraremos dos de ellas, traducidas libremente por las/os autoras/es de este manual, que creemos muestran el nivel de detalle al que se llega[5].

1) Indentación

Los tabs tienen una longitud de 8 caracteres, y por lo tanto las indentaciones también tienen una longitud de 8 caracteres. Hay movimientos heréticos que intentan hacer indentacions de 4 (¡o incluso 2!) caracteres, y eso es similar a definir el valor del número *pi* como 3.

Explicación: la idea de indentar es definir claramente dónde comienza y termina un bloque de control. Especialmente cuando has estado mirando a la pantalla durante 20 horas, encontrarás mucho más fácil ver como funciona la indentación si es lo suficientemente grande.

Ahora, algunas personas indican que tener indentaciones de 8 caracteres hace que el código se mueva muy a la derecha y hace difícil leerlo en una terminal de 80 caracteres. La respuesta a esto es que si necesitas más de 3 niveles de indentación, te mandaste una macana y deberías arreglar tu programa.

Resumiendo, indentaciones de 8 caracteres hacen las cosas más fáciles de leer y tienen el beneficio adicional de alertarte cuando estás anidando las funciones demasiado profundamente. Debes prestar atención a eso.

El modo preferido de reducir los múltiples niveles de indentación en una sentencia `switch` es alinear el `switch` y sus rótulos en la misma columna, en vez de aplicar una indentación más a cada rótulo. Por ejemplo:

```
switch (suffix) {
case 'G':
case 'g':
        mem <<= 30;
        break;
```

[4]Linux kernel coding style: `https://www.kernel.org/doc/html/v4.10/process/coding-style.html`

[5]A pesar de ser reglas muy detalladas y estrictas, las lectoras y los lectores notarán que están escritas jocosamente. Es más, comienzan con esta "mojada de oreja" al proyecto GNU: "Antes que nada, sugiero imprimir una copia de los estándares de codificación de GNU y NO leerla. Quémela, es un gran gesto simbólico."

```
case 'M':
case 'm':
        mem <<= 20;
        break;
case 'K':
case 'k':
        mem <<= 10;
        /* fall through */
default:
        break;
}
```

No pongas múltiples sentencias en una misma línea a menos que tengas algo que esconder:

```
if (condition) do_this;
  do_something_everytime;
```

No pongas tampoco múltiples asignaciones en una única línea. El estilo de codificación del núcleo es super simple. Evita expresiones engañosas.

Aparte de los comentarios, la documentación y en Kconfig, los espacios nunca deben ser usados para indentación, y en el ejemplo de arriba esta regla está rota a propósito.

Utiliza un editor decente y no dejes espacios en blanco al final de las líneas.

4) Nombres

C es un lenguaje espartano[6], y así debería ser tu forma de asignar nombres. De manera diferente a como hacen los/as programadores/as de Modula-2 y de Pascal, los/as programadores/as C no usan lindos nombres como EstaVariableEsUnContadorTemporario. Un/a programador/a C llamaría a esta variable como `tmp`, lo cual es mucho más fácil de escribir y no más dificultoso de entender.

SIN EMBARGO, mientras nombres con mayúsculas y minúsculas son mal vistos, los nombres descriptivos para las variables globales son necesarios. Llamar `foo` a una función global es una ofensa pagada con la muerte.

Las variables GLOBALES (que deben ser utilizadas solamente si las necesitas) deben tener nombres descriptivos, igual que las funciones globales. Si tienes una función que cuenta el número de usuarios activos, deberías llamarla `count_active_users()` o algo similar, en vez de llamarla `cntusr()`.

Codificar el tipo de la función en el nombre (la así llamada "notación húngara") es de locos - de todas maneras el compilador conoce los tipos y puede chequearlos,

[6] Aquí "espartano" no significa que nació en Esparta, sino que es un lenguaje austero.

y esto solo confunde al programador. No es sorpresa que Microsoft haga programas con errores.

Los nombres de variables LOCALES deberían ser cortos y al pie. Si tienes algún contador de ciclos, debería llamarse `i`. Llamarlo `loop_counter` es antiproductivo, no hay chances de que `i` no se entienda. Similarmente, `tmp` puede ser cualquier variable que se use para contener un valor temporario.

Si tienes miedo de confundir los nombres de tus variables locales, entonces tienes otro problema, el cual es llamado síndrome de desbalanceo de la hormona de crecimiento en la función. Ver el capítulo 6 (Funciones).

6.4.4. Zen of Python

El Zen de Python[7] es una colección de 19 principios de software que influyen en el diseño del lenguaje de programación Python. Fueron escritos por Tim Peters en 1999.

1. Lindo es mejor que feo.
2. Explícito es mejor que implícito.
3. Simple es mejor que complejo.
4. Complejo es mejor que complicado.
5. Plano es mejor que anidado.
6. Espaciado es mejor que denso.
7. La legibilidad es importante.
8. Los casos especiales no son lo suficientemente especiales como para romper las reglas.
9. Sin embargo la practicidad le gana a la pureza.
10. Los errores nunca deberían pasar silenciosamente.
11. A menos que se silencien explícitamente.
12. Frente a la ambigüedad, evitar la tentación de adivinar.
13. Debería haber una, y solamente una, manera obvia de hacerlo.

[7] The Zen of Python: `http://www.thezenofpython.com/`

14. A pesar de que no sea obvio.
15. Ahora es mejor que nunca.
16. A pesar de que nunca es muchas veces mejor que justo ahora.
17. Si la implementación es dificil de explicar, es una mala idea.
18. Si la implementación es fácil de explicar, quizás sea una buena idea.
19. Los namespaces son una gran idea, ¡tengamos más de esos!

Capítulo 7

La filosofía del software libre en otros ámbitos

Tanto el software libre como su filosofía de trabajo comunitario, de compartir libremente la información y de aportes voluntarios han tenido impacto en otros ámbitos por fuera del desarrollo de software.

7.1. Software libre en el Estado

El software libre impacta de varias maneras en las administraciones públicas: permite un mejor aprovechamiento de recursos, fomenta la industria local, ofrece independencia de proveedores monopólicos, permite el escrutinio público y garantiza el acceso a los datos a largo plazo.

- **Aprovechamiento más adecuado de los recursos:** Muchas aplicaciones utilizadas o promovidas por las administraciones públicas son también utilizadas por muchos otros sectores de la sociedad. Por ello, cualquier inversión pública en el desarrollo de un producto libre redundará en beneficios no sólo en la propia administración, sino en todos los ciudadanos u otras administraciones (municipales, provinciales, nacionales) que puedan encontrarle utilidad.
- **Fomento de la industria local:** Una de las mayores ventajas del software libre es la posibilidad de desarrollar industria local de software. Cuando se usa software propietario, todo lo gastado en licencias va directamente al fabricante del producto, y además esa compra redunda en el fortalecimiento de su posición. Lo cual no es necesariamente perjudicial, pero poco eficiente para las regiones vinculadas a las diferentes administraciones, comparado

con la alternativa de usar un programa libre. En este caso, las empresas locales podrán competir proporcionando servicios (y el propio programa) a la administración, en condiciones muy similares a cualquier otra empresa.

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

  En el caso del software propietario, para cada producto no hay más que un proveedor. Si se especifica un producto dado, se está decidiendo también qué proveedor contratará con la administración. Y en muchos casos es prácticamente imposible evitar especificar un cierto producto cuando estamos hablando de programas de ordenador. Razones de compatibilidad dentro de la organización, o de ahorros en formación y administración, hacen habitual que una administración decida usar un cierto producto.

  La única salida a esta situación es que el producto especificado sea libre. En ese caso, cualquier empresa interesada podrá proporcionarlo, y también cualquier tipo de servicio sobre él. Además, en caso de contratar de esta manera, la administración pertinente podrá en el futuro cambiar inmediatamente de proveedor si así lo desea, y sin ningún problema técnico, pues aunque cambie de empresa, el producto que usará será el mismo.

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

  Cuando una administración compra un producto propietario, modificarlo pasa normalmente por alcanzar un acuerdo con su productor, que es el único que legalmente (y muchas veces técnicamente) puede hacerlo. En esas condiciones, es difícil realizar buenas negociaciones, sobre todo si el productor no está excesivamente interesado en el mercado que le ofrece la administración en cuestión. Sin embargo, usando un producto libre, esa administración

puede modificarlo a su antojo, si dispone de personal para ello, o contratar externamente la modificación. Esta contratación la puede realizar en principio cualquier empresa que tenga los conocimientos y capacidades para ello, por lo que es de esperar que haya concurrencia de varias empresas. Este hecho, necesariamente tiende a abaratar los costes y a mejorar la calidad.

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

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

Si bien las ventajas de uso del software libre en las administraciones son muchas, también son muchas las dificultades cuando se enfrentan a su puesta en práctica.

- **Desconocimiento y falta de decisión política:** El primer problema que se encuentra el software libre para su introducción en las administraciones es uno que comparte con otras organizaciones: es aún muy desconocido entre quienes toman las decisiones. Éste es un problema que cada vez se va solucionando mejor, pero en muchos ámbitos de las administraciones el software libre aún es percibido como algo extraño, sin garantías o hecho por amateurs, y tomar la decisión de usarlo implica asumir ciertos riesgos técnicos y/o políticos.

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

- **Falta de estrategia de implantación:** En muchos casos, el software libre comienza a usarse en una administración simplemente porque el coste de adquisición es más bajo. Es muy habitual en estos casos que el producto en cuestión se incorpore al sistema informático sin mayor planificación, y en general sin una estrategia global de uso y aprovechamiento de software libre. Esto causa que la mayor parte de las ventajas del software libre se pierdan en el camino, ya que todo queda en el uso de un producto más barato.

  Si a esto unimos que la migración de un software privativo de uso extendido en la administración a un software libre supone costes de transición no despreciables, hace que experiencias aisladas de uso de software libre en la administración puedan resultar fallidas y frustrantes.

- **Escasez o ausencia de productos libres en ciertos segmentos:** La intención de implantar software libre en cualquier organización puede chocar con la falta de alternativas libres de la calidad adecuada para cierto tipo de aplicaciones. En esos casos, la solución es complicada o riesgosa: tratar de promover la aparición del producto libre que se necesita. Las administraciones públicas están en una buena posición para estudiar seriamente si les conviene fomentar, o incluso financiar o cofinanciar el desarrollo de ese producto.

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

- Comprando programas y servicios relacionados con ellas. Las administraciones, como grandes usuarios de informática, son un actor fundamental en el mercado del software.
- Promoviendo de diversas formas el uso (y la adquisición) de ciertos programas en la sociedad. Esta promoción se hace a veces ofreciendo incentivos económicos (desgravaciones fiscales, incentivos directos, etc.), a veces con información y recomendaciones.
- Financiando (directa o indirectamente) proyectos de investigación y desarrollo que están diseñando el futuro de la informática.

7.1.1. Seguridad de los datos de los ciudadanos

El Estado debe almacenar y procesar información relativa a los ciudadanos. La relación entre el individuo y el Estado depende de la privacidad e integridad de estos datos, que por consiguiente deben ser adecuadamente resguardados contra tres riesgos específicos:

- Riesgo de filtración: los datos confidenciales deben ser tratados de tal manera que el acceso a ellos sea posible exclusivamente para las personas e instituciones autorizadas.
- Riesgo de imposibilidad de acceso: los datos deben ser almacenados de tal forma que el acceso a ellos por parte de las personas e instituciones autorizadas esté garantizado durante toda la vida útil de la información.
- Riesgo de manipulación: la modificación de los datos debe estar restringida, nuevamente, a las personas e instituciones autorizadas.

La concreción de cualquiera de estas tres amenazas puede tener consecuencias graves tanto para el Estado como para el individuo. El software libre permite al usuario la inspección completa y exhaustiva del mecanismo mediante el cual procesa los datos. Sin la posibilidad de la inspección, es imposible saber si el programa cumple con su función, o si además incluye vulnerabilidades intencionales o accidentales que permitan a terceros acceder indebidamente a los datos, o impedir que los usuarios legítimos de la información puedan usarlo. El hecho de permitir la inspección del programa es una excelente medida de seguridad, ya que al estar

expuestos los mecanismos, estos están constantemente a la vista de profesionales capacitados, con lo que se vuelve inmensamente más difícil ocultar funciones maliciosas, aún si el usuario final no se toma el trabajo de buscarlas él mismo.

7.1.2. Leyes de uso de software en el Estado

Con lo antes dicho hemos visto que el software tiene un profundo impacto en las actividades realizadas por el Estado. Los riesgos que involucra una elección desafortunada no son menores: imposibilidad de auditar la función pública, falta de garantías por parte del Estado sobre la manipulación de la información privada de sus ciudadanos, dependencia de un proveedor para el correcto desempeño de las funciones del Estado, imposibilidad de los ciudadanos de acceder a su información, entre otras.

Es claro entonces que la adquisición de software por parte del Estado debiera estar regulada por una ley que, previendo este tipo de situaciones, fije las condiciones bajo las cuales el proveedor debe suministrar los programas en cuestión. No puede dejarse librada a cada funcionario responsable de un área de la administración pública la decisión de las condiciones de contratación o compra de software, dado los peligros que podría acarrear para el conjunto de la comunidad una elección desafortunada.

7.2. Software libre en la educación

Analizaremos los distintos aspectos relacionados con el uso de software en la educación de ciudadanos/as. Desarrollaremos dicho análisis desde tres enfoques diferentes: el económico, el académico y el ético.

- **Aspecto económico**

  Aunque, como ya indicamos repetidamente, la libertad del software no implica su gratuidad, esto no invalida el hecho de que el software propietario es altamente costoso tanto en su adquisición como en su mantenimiento y que el acceso al software libre es, en principio, gratuito.

  El alto precio de las licencias de uso, en las que se basa el modelo de comercialización y distribución propietario, hacen que los montos que debieran invertirse en la adquisición del derecho de utilización sean muy elevados. Dichos costos contrastan notoriamente con los costos de adquisición de los productos licenciados como software libre.

  Cabe resaltar una diferencia radical entre ambos modelos: en el caso del software propietario, lo que se adquiere al pagar el precio de una licencia es el

permiso de ejecutar el programa en cuestión bajo determinadas condiciones, en tanto que al adquirir una pieza de software libre se obtiene una copia del programa (incluyendo su código fuente) y el permiso del autor para hacer uso de las cuatro libertades que definen a este tipo de software.

El impacto de las licencias del software propietario no sólo debe tenerse en cuenta a la hora de evaluar los costos de equipar un laboratorio informático en una institución educativa, sino que también debe atenderse al hecho de que aquellos alumnos que quieran utilizar las mismas herramientas en su hogar también deberán adquirir dichas licencias.

- **Aspecto académico**

  Aunque el objetivo de la educación no es formar especialistas en determinadas tecnologías, es indudable que a la hora de aplicar los conceptos teóricos debe optarse por alguna en particular. Es una obligación, por parte del docente, el seleccionar cuidadosamente las tecnologías más con venientes a tal efecto.

  En el caso del software, a través de la historia, una serie de empresas -que han gozado de una situación de monopolio- han pretendido marcar el rumbo tecnológico. Invariablemente, estas empresas han tratado de ejercer presión sobre el sistema académico para lograr el acercamiento de los estudiantes a sus productos, invirtiendo en muchos casos elevadas sumas de dinero a tal efecto (bajo la forma de convenios o donaciones).

  Creemos que el uso de herramientas propietarias, generalmente ligadas a tecnologías específicas bajo el control de determinadas empresas, contribuye a lograr una fuerte dependencia de los y las estudiantes, en cuanto ciudadanos digitales. Cada uno/a de ellos/as actuará como un agente de ventas de dicha empresa al utilizar, recomendar y hasta exigir soluciones basadas exclusivamente en los productos que conoce.

  Por contrapartida, al utilizar herramientas libres, se le brinda al estudiante la posibilidad de conocer los detalles de la implementación de las mismas y los fundamentos tecnológicos en los que éstas se basan.

- **Aspecto ético**

  Con relación al alto costo de las licencias de uso, es una actividad muy común en la actualidad la utilización de “copias no autorizadas” de productos propietarios, bajo la permisiva mirada de los y las docentes. La práctica de este tipo de actividades conlleva las siguientes consecuencias:

- La violación de las condiciones de la licencia correspondiente al producto utilizado de forma irregular constituye lisa y llanamente una violación de la ley.
- El uso de copias no autorizadas contribuye a la estrategia utilizada por las empresas productoras de software propietario para promocionar sus productos y provocando así el llamado "efecto de red", ya que al tratarse de tecnología cerrada y bajo el control de dichas empresas, se está propiciando el establecimiento de la misma como un estándar "de facto".

Los y las docentes están, entonces, promoviendo o al menos permitiendo, que se viole la ley, lo cual está en las antípodas del comportamiento ético que se espera de ellos/as.

7.2.1. Necesidad de una política institucional

Ante los peligros que puede acarrear la incorrecta elección de herramientas propietarias en la educación, creemos que ésta es una decisión que no puede quedar librada al criterio de cada docente, sino que debe definirse una política institucional que determine los lineamientos para la elección de herramientas y tecnologías informáticas en las distintas actividades desarrolladas dentro de su ámbito.

Un ejemplo de software libre utilizado por una decisión institucional en nuestro país es el sistema operativo Huayra[1], el cual es una distribución GNU/Linux desarrollada por el Estado Nacional. Se incorporó desde 2013 en las netbooks que el Programa Conectar Igualdad (PCI[2]) entrega a estudiantes secundarios de escuelas públicas de todo el país. El PCI, aún antes de incorporar a Huayra, incluía en las netbooks software libre para uso educativo, incluso en el sistema operativo privativo con el que vienen todas las máquinas en modo *doble booteo*[3].

7.2.2. Recursos Educativos Abiertos

El portal educ.ar: Dispone de diferentes materiales desarrollados a partir de software libre: desde juegos hasta un CD de la Colección educ.ar, con actividades específicas, como por ejemplo contenidos pensados para introducir a los docentes en el mundo del software libre: sus postulados ideológicos, teóricos, técnicos y metodológicos, acompañados por sugerencias de trabajo con programas, herramientas

[1]Huayra: `https://huayra.educar.gob.ar/`

[2]Programa Conectar Igualdad: `https://conectarigualdad.edu.ar/inicio`

[3]Con excepción de las entregadas en 2021 bajo el Plan Federal Juana Manso, que solo incluyen Huayra versión 5 [**?**]

y modelos de esta nueva modalidad de producción, construcción y difusión de los contenidos.

7.3. Software libre en organizaciones sociales

Hay muchas razones, por sus principios filosóficos por razones jurídicas, sociales y socieconómicas, funcionalidad, accesibilidad y seguridad. Para las organizaciones y movimientos sociales de América Latina y del mundo, el software libre puede ser incorporado como herramienta tecnológica y, por qué no, también como herramienta de liberación.

Por sus principios filosóficos:

- Considera la libertad y la solidaridad como principios éticos fundamentales.
- Considera el conocimiento como un bien público que beneficia a la colectividad en general y permite el desarrollo igualitario.
- El software es conocimiento y debe difundirse sin trabas.
- Coloca el beneficio de la humanidad por encima de cualquier cosa.

Por razones jurídicas:

- Es legal compartir y regalar software libre a otras personas.
- Evita los problemas de uso ilegal de licencias de software propietario/privativo y sus posibles implicaciones legales (demandas, multas, entre otros).

Por razones sociales y socioeconómicas:

- Promueve la economía solidaria y el comercio justo.
- Contempla a las personas usuarias como sujetos de derechos no como consumidores de productos.
- Se desarrolla en términos colaborativos y participativos.
- Las organizaciones no cuentan con muchos recursos económicos y se necesita destinar los que tiene para el logro de su misión y objetivos.
- Las licencias del software propietario/privativo son muy caras para las organizaciones sociales (costo de pagar licencias para cada computadora y programas especializados).

- En la mayoría de los casos, las organizaciones no podrían costear las multas impuestas judicialmente como resultado de demandas legales por el uso ilegal de licencias de software privativo.
- Se podría decir que el software libre es amigable con el ambiente, al no requerir de la sustitución tan rápidamente de equipo, se produce menos basura tecnológica.

Funcionalidad, accesibilidad y seguridad:

- El software libre garantiza la seguridad que implica corrección de problemas de forma oportuna; en el movimiento de software libre hay equipos humanos trabajando para corregir los problemas de seguridad, adaptado a las personas, contextos e idioma.
- Mucha de la información que manejan las organizaciones de la sociedad civil es información "delicada" que necesita ser resguardada, el software libre también ofrece herramientas para asegurar la información.

7.4. Voto Electrónico

Consideraciones sobre el voto electrónico

- Hay dos tipos: DREs (Direct Recording Electronic) vs EBPs (Electronic Ballot Printers).
- Requerimientos Básicos del Sistema de Voto Electrónico: secreto del voto, fidelidad del voto, solo los electores autorizados pueden votar, cada lector debe emitir un solo voto.
- No es Fácil hacer un sistema que pueda ser auditado.
- Problema de las amenazas.
- El acceso al código debe ser lo más abierto posible (Open source Software).

Casos fallidos en el mundo

- Alemania : Los sistemas usados hasta el momento se declararon inconstitucionales.
- Holanda : dejó de usarse en 2007 al probarse que los votos podían ser leídos a varios metros de distancia (en algunas máquinas), y que los programas podían ser alterados. EEUU múltiples errores en diversos estados.

- India: hackers lograron manipular los resultados con un celular.

- Irlanda : evaluaron un sistema en elecciones piloto y determinaron que no se podía garantizar la integridad de ninguna elección que usara ese sistema. Costo del experimento: 54 millones de euros.

7.5. Licencia Creative Commons

Es una organización sin fines de lucro dedicada a promover el acceso y el intercambio de cultura. Desarrolla un conjunto de instrumentos jurídicos de carácter gratuito que facilitan usar y compartir tanto la creatividad como el conocimiento. Los instrumentos jurídicos desarrollados por la organización consisten en un conjunto de "modelos de contratos de licenciamiento" o licencias de derechos de autor que ofrecen al autor de una obra una manera simple y estandarizada de otorgar permiso al público para compartir y usar su trabajo creativo bajo los términos y condiciones de su elección. En este sentido, las licencias Creative Commons permiten al autor cambiar fácilmente los términos y condiciones de derechos de autor de su obra de "todos los derechos reservados" a "algunos derechos reservados". Las licencias Creative Commons no reemplazan a los derechos de autor, sino que se apoyan en estos para permitir elegir los términos y condiciones de la licencia de una obra de la manera que mejor satisfaga al titular de los derechos. Por tal motivo, estas licencias han sido entendidas por muchos como una manera en que los autores pueden tomar el control de cómo quieren compartir su propiedad intelectual. Existe una serie de licencias Creative Commons, cada una con diferentes configuraciones, que permite a los autores poder decidir la manera en la que su obra va a circular en internet, entregando libertad para citar, reproducir, crear obras derivadas y ofrecerla públicamente, bajo ciertas restricciones.

Las licencias Creative Commons están compuestas por cuatro módulos de condiciones:

- Atribución (Attribution): En cualquier explotación de la obra autorizada por la licencia será necesario reconocer la autoría (obligatoria en todos los casos).

- No Comercial (Non commercial): La explotación de la obra queda limitada a usos no comerciales.

- Sin obras derivadas (No Derivate Works): La autorización para explotar la obra no incluye la posibilidad de crear una obra derivada.

- Compartir Igual (Share alike): La explotación autorizada incluye la creación de obras derivadas siempre que mantengan la misma licencia al ser divulgadas.

Con estas condiciones se pueden generar distintas combinaciones que producen las licencias Creative Commons:

- Atribución (by): Se permite cualquier explotación de la obra, incluyendo la explotación con fines comerciales y la creación de obras derivadas, la distribución de las cuales también está permitida sin ninguna restricción. Esta licencia es una licencia libre según la Freedom Defined.
- Reconocimiento – Compartir Igual (by-sa): Se permite el uso comercial de la obra y de las posibles obras derivadas, la distribución de las cuales se debe hacer con una licencia igual a la que regula la obra original. Esta licencia es una licencia libre según la Freedom Defined.
- Atribución – No Comercial (by-nc): Se permite la generación de obras derivadas siempre que no se haga con fines comerciales. Tampoco se puede utilizar la obra original con fines comerciales. Esta licencia no es una licencia libre.
- Atribución – No Comercial – Compartir Igual (by-nc-sa): No se permite un uso comercial de la obra original ni de las posibles obras derivadas, la distribución de las cuales se debe hacer con una licencia igual a la que regula la obra original. Esta licencia no es una licencia libre.
- Atribución – Sin Obra Derivada (by-nd): Se permite el uso comercial de la obra pero no la generación de obras derivadas. Esta licencia no es una licencia libre.
- Atribución – No Comercial – Sin Obra Derivada (by-nc-nd): No se permite un uso comercial de la obra original ni la generación de obras derivadas. Esta licencia no es una licencia libre, y es la más cercana al derecho de autor tradicional.

Obtener la licencia

Cuando hayas hecho tu elección tendrás la licencia adecuada para tu trabajo expresada de tres formas:

- Commons Deed: Es un resumen fácilmente comprensible del texto legal con los íconos relevantes.

- Legal Code: El código legal completo en el que se basa la licencia que elegiste.

- Digital Code: El código digital, que puede leer la máquina y que sirve para que los motores de búsqueda y otras aplicaciones identifiquen tu trabajo y sus condiciones de uso.

Utilizar la licencia

Una vez escogida la licencia tenés que incluir el ícono de Creative Commons "Algunos derechos reservados" en tu sitio, cerca de tu obra. Este ícono enlaza con el Commons Deed, de forma que todos puedan estar informados de las condiciones de la licencia. Si encontrás que alguien cometió una infracción a la licencia, entonces tendrás las bases para poder defender tus derechos.

Creative commons en Argentina

Creative Commons Argentina está a cargo de dos organizaciones afiliadas: Fundación Vía Libre y Wikimedia Argentina.

7.6. Licencias libres para documentación

Las siguientes licencias reúnen las condiciones necesarias para calificarse como licencias de documentación libre.

- **La Licencia de documentación libre de GNU o GFDL**
  Es una licencia copyleft para contenido libre, diseñada por la Fundación para el Software Libre (FSF) para el proyecto GNU.
  Esta licencia, a diferencia de otras, asegura que el material licenciado bajo la misma esté disponible de forma completamente libre, pudiendo ser copiado, redistribuido, modificado e incluso vendido siempre y cuando el material se mantenga bajo los términos de esta misma licencia (GNU GFDL).
  Dicha licencia fue diseñada principalmente para manuales, libros de texto y otros materiales de referencia e institucionales que acompañaran al software GNU. Sin embargo puede ser usada en cualquier trabajo basado en texto, sin que importe cuál sea su contenido.

- **FreeBSD Documentation License**
  Esta es una licencia de documentación libre y permisiva, sin copyleft, incompatible con la FDL de GNU.

- **Apple's Common Documentation License**
  Esta en una licencia de documentación libre, incompatible con la FDL de GNU. Es incompatible porque la sección (2c) dice «No agregue otros términos ni condiciones a esta licencia», y la FDL de GNU posee cláusulas adicionales que no están contempladas en la Common Documentation License.

- **Open Publication License**
  Esta licencia puede ser utilizada como de documentación libre. Es una licencia libre con copyleft para documentación , siempre y cuando el titular del copyright no ejerza ninguna de las «OPCIONES DE LA LICENCIA» que se mencionan en la sección VI. Pero si se invoca cualquiera de esas opciones, la licencia se vuelve privativa. En cualquier caso, es incompatible con la FDL de GNU.

Bibliografía

[Corel, 1999] Corel (1999). Corel's contributions to the wine project. Linux Today. https://www.linuxtoday.com/developer/corels-contributions-to-the-wine-project/.

[Gates, 1976] Gates, B. (1976). Open letter to hobbysts. *Homebrew Computer Club Newsletter*, 2(1). Mountain View, CA.

[Gerasimenko, 2016] Gerasimenko, S. (2016). Open source for a startup (the good, the bad and the ugly). Nordic Founders. https://medium.com/nordic-founders/open-source-for-a-startup-the-good-the-bad-and-the-ugly-a531ebbd38ed.

[Greene, 2001] Greene, T. C. (2001). Ballmer: 'linux is a cancer'. The Register. https://www.theregister.com/2001/06/02/ballmer_linux_is_a_cancer/.

[Kamau, 2016] Kamau, K. (2016). Give your startup a competitive edge with open source. COD.e. https://medium.com/cod-e/give-your-startup-a-competitive-edge-with-open-source-9163ffaea21.

[Michán, 2010] Michán, M. (2010). Nexstep, el eslabón perdido de mac os x. *Applesfera*. https://www.applesfera.com/apple/nextstep-el-eslabon-perdido-de-mac-os-x.

[Monk, 2014] Monk, L. (2014). Cooperativas, software libre y gestión de conocimiento. In *Foro de Responsables Informáticos de las Universidades Nacionales de Argentina - TICAR 2014*. https://www.ticar.org.ar/files/GCOOP.pdf.

[Raymond, 2001] Raymond, E. S. (2001). *The Cathedral & the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. O'Reilly Media.

[Singh, 2021] Singh, D. (2021). Top 5 open source tools you can use in startup. Medium1. https://medium.com/cod-e/give-your-startup-a-competitive-edge-with-open-source-9163ffaea21.

[Tanenbaum and Woodhull, 1987] Tanenbaum, A. S. and Woodhull, A. S. (1987). *Operating Systems: Design and Implementation*. Prentice Hall.

[Watkins, 2018] Watkins, D. (2018). Libreoffice: A history of document freedom. *opensource.com*.

[Wiggers, 2021] Wiggers, K. (2021). Open source nlp is fueling a new wave of startups. VentureBeat. https://venturebeat.com/2021/12/23/open-source-nlp-is-fueling-a-new-wave-of-startups/.

[Williams, 2002] Williams, S. (2002). *Free as in Freedom: Richard Stallman's Crusade for Free Software*. O'Reilly Media.