EL MERCADO DEL SOFTWARE: CÓDIGO ABIERTO Y EFECTOS DE RED [*]

Germán Márquez Mejía[**]
gmarquez@javeriana.edu.co

PONTIFICIA UNIVERSIDAD JAVERIANA
Facultad de Ciencias Económicas y Administrativas
Carrera de Economía

Marzo de 2009

Resumen

A partir de un análisis de teoría de juegos se explora la interacción de dos paradigmas de producción y consumo de software: libre y propietario, en un mercado gobernado por efectos de red y con la presencia de una externalidad positiva en el desarrollo de programas bajo el paradigma de "código abierto", de creciente protagonismo en la industria. Después de una introducción a los conceptos relacionados con el software y una revisión de la incipiente literatura económica sobre el tema, se desarrolla un modelo de teoría de juegos, cuyas estrategias mixtas son endógenamente determinadas, para describir la estructura del mercado del software a la luz del equilibrio de Nash, encontrando que los proyectos más costosos son mejor recibidos bajo el paradigma abierto y que los programas libres poseen una tendencia natural a la competencia, mientras que el software propietario se comporta mejor en ambientes dominados por amplios márgenes de beneficio.

[*]Trabajo presentado como requisito para recibir el grado de economista.

[**]Bajo la dirección del profesor Juan Pablo Herrera Saavedra de la Pontificia Universidad Javeriana, Bogotá D.C., Colombia.

1. Introducción

En nuestro trabajo diario somos conscientes de cuan útiles e importantes son las computadoras, pero, para que hagan lo que deseamos, ellas deben recibir instrucciones creadas por humanos. Estas instrucciones se escriben en forma de *código fuente* (en lenguajes de programación fáciles de entender, como C++ ó Java) y luego son traducidas a *código de máquina* (secuencias de ceros y unos). El código fuente está compuesto por miles (incluso millones) de líneas de texto plano escrito por los programadores como un guión de lo que un programa deberá realizar cuando se ejecute. Sin embargo, antes de que sea posible utilizar el software, esto debe ser convertido en una secuencia de bits (datos binarios) para que la computadora pueda interpretarlo, es decir, código de máquina. La conversión se lleva a cabo mediante el proceso de compilación, donde un segundo programa traduce los archivos fuente y genera archivos binarios. Son éstos los que, finalmente, se ejecutan como una aplicación.

En la economía de la información, el software puede analizarse como un bien intangible con características muy particulares. Por su naturaleza lógica (no física), es de fácil reproducción, lo que le lleva a ser un recurso inagotable (o, en otros términos, no rival). Por otro lado, generalmente, cuando demanda la interacción con un usuario, requiere un proceso de aprendizaje antes de poder ser usado eficientemente, lo que causa que se presente complementariedad con otros servicios como capacitación y soporte. Pero no sólo existe este tipo de relación con los servicios, también con otros bienes, tanto de software como de hardware, ya que, en su operación, un programa debe comunicarse con componentes físicos externos y otras aplicaciones. De lo contrario, estaría aislado y brindaría escasas funciones útiles en sistemas complejos. La dependencia derivada de estas relaciones de complementariedad hace que el consumo de software esté fuertemente dominado por efectos de red. Además, y como última característica importante, el software es un bien intensivo en trabajo calificado. Por más sencilla que sea una aplicación, su desarrollo es complejo y la escritura del código demanda muchas horas hombre de trabajo mental alrededor del análisis, diseño, implementación y prueba del nuevo sistema.

Dada la naturaleza particular del software como bien, en los inicios de la era de la computación pocos se preocupaban por definir estrictamente los derechos de propiedad sobre él, principalmente en ambientes académicos. Sin embargo, a medida que más y más personas trabajaban en informática, surgieron conflictos y se vio la necesidad de sentar posiciones sobre cómo debería ser el proceso de creación, uso, modificación y

distribución. Preguntas como: ¿debe un usuario tener acceso al código fuente o sólo al código máquina?, ¿se debe poder distribuir copias de los programas libremente? o ¿cualquiera debería poder modificar cierta aplicación para adaptarla a sus necesidades? comenzaron a alimentar el debate. De esta controversia emergieron dos paradigmas que intentan responder a estas preguntas basados en argumentos de distinta índole: el del software libre y el del software propietario. Las respuestas que ofrece uno y otro enfoque guardan una gran distancia entre sí, y, por lo tanto, constituyen dos marcos legales de definición de derechos de propiedad muy diferentes.

Con el fin de modelar el mercado del software en un análisis aplicación por aplicación, se aclaran los aspectos técnicos y legales del mismo y las reglas de juego en su producción y uso (consumo). Se examinan los avances recientes de la literatura sobre el tema y luego se intenta describir el comportamiento de los agentes en el mercado mediante un modelo de interacción en ausencia de externalidades, por un lado, y, por otro, bajo el efecto de una externalidad positiva en el desarrollo de código abierto y de efectos de red en el consumo, utilizando como instrumento un juego estratégico donde la distribución de probabilidad que cada jugador asigna a sus estrategias determina endógenamente sus pagos. Se encuentra que a mayor costo de desarrollo, habrá mayores incentivos a producir una aplicación en su versión libre aprovechando la externalidad positiva, y que la liberación de código es una alternativa importante para considerar al interior de las firmas productoras de software propietario cuando se enfrentan una disminución en sus márgenes de beneficio.

2. Software y conceptos relacionados

No hay una definición propia y formal de software propietario. Por su carácter antagónico, se tiende a definirlo en términos inversos a los del software libre[1]. Este último se apoya en argumentos éticos de libertad, mientras que el primero lo hace bajo argumentos económicos tradicionales. De acuerdo con Stallman (2004)[2], para que un programa se considere software libre debe garantizar cuatro libertades básicas : i) libertad para ejecutarlo con cualquier propósito; ii) libertad para conocer su funcionamiento y adaptarlo a las necesidades particulares; iii) libertad para redistribuir copias; y iv) libertad para mejorarlo y hacer públicas dichas mejoras. Las libertades dos y cuatro

[1]Incluso también se utiliza el término *software no libre* para referirse a él.

[2]Quien es el fundador de la Free Software Foundation (FSF) y uno de los principales impulsadores del movimiento libre.

demandan la disponibilidad del código fuente, ya que es virtualmente imposible entender o modificar un programa complejo a partir del código máquina. En contraste, el software propietario es aquel que restringe una o más de estas libertades. La idea de este paradigma es que, en caso de garantizarlas, los desarrolladores dejarían de recibir ingresos de la venta del software que crean y no tendrían incentivos para continuar su labor, ya que invertirían muchas horas de trabajo sin recibir nada a cambio (Gates, 1976). El argumento ético del software libre se fundamenta en la necesidad de libertad, en la concepción del software como información a la cual deberían tener acceso todos los miembros de la sociedad y en la conservación del espíritu cívico de cooperación entre los ciudadanos. Los beneficios sociales derivados —sostienen sus partidarios— exceden a aquéllos de los programadores de software propietario (Stallman, 2004).

Para aplicar uno u otro paradigma, se suele utilizar una herramienta facilitada por los sistemas legales de la mayoría de países: los acuerdos de licencia —o, simplemente, licencias— de software. En informática, un acuerdo de licencia es una figura contractual en la que el titular de los derechos de autor del software establece restricciones y/o concede permisos sobre el uso, modificación, precio o redistribución del mismo. Las licencias propietarias suelen impedir la modificación y redistribución, al igual que imponen restricciones a la cantidad de usuarios y propósito de uso de los programas que cubren. También es común que sus distribuciones contengan sólo el código máquina de las aplicaciones, reservando las fuentes para conocimiento exclusivo de las empresas que las producen. Algunas, menos restrictivas, permiten la distribución del código máquina, a veces sin costo alguno (*freeware*) o a cambio de alertas publicitarias, sugerencias de compra o características limitadas (*shareware*). Pero así como existen múltiples licencias propietarias, también las hay libres. Este tipo de licencias, al contrario de la creencia popular, no exigen la gratuidad del software cubierto[3]. Es completamente legal vender software libre, siempre y cuando la venta incluya de alguna forma el código fuente además del código máquina. Dos de las licencias libres más representativas, por sus diferencias en términos de incentivos, son la Licencia Pública General GNU (GNU GPL, por sus siglas en inglés) y la Licencia de Distribución de Software Berkeley (BSD). Mientras que ambas son iguales en espíritu, la GNU GPL exige que toda modificación y/o distribución del software sea hecha bajo exactamente los mismos términos de la licencia original (que asegura las cuatro libertades expuestas anteriormente). A esta

[3]Esta confusión es heredada principalmente del inglés, donde el término *free* (libre) tiene la acepción adicional de *gratuito*, por lo que *free software* (software libre) puede interpretarse también como software gratuito.

cláusula se le conoce comúnmente como *copyleft* (en contraposición a *copyright*). Por otro lado, la licencia BSD no impone ninguna restricción a la modificación y/o distribución posteriores, haciendo posible que, por ejemplo, se licencie de manera propietaria el software modificado. Se dice que la GNU GPL es una licencia *restrictiva* y la BSD, *permisiva*[4]. Como la mayor parte del software libre se licencia bajo acuerdos de tipo restrictivo, el análisis se centrará en la GNU GPL.

Existen muchos proyectos de desarrollo de software libre[5]. Algunos ejemplos muy conocidos son: GNU/Linux (un sistema operativo y aplicaciones libres relacionadas), Mozilla Firefox (un navegador web) y OpenOffice.org (un paquete ofimático), pero también se puede encontrar una gran variedad de proyectos profesionales especializados como Apache (un servidor web) o —en lo que concierne a la economía— R (un lenguaje de alto nivel para econometría) y LaTeX (un sistema de preparación de documentos técnicos). A su vez, muchas empresas han mostrado su interés y compromiso con proyectos como los anteriores[6] mientras que otras han surgido en el seno mismo del software libre[7], lo cual ha contribuido a que la controversia entre uno y otro paradigma de desarrollo se traslade a un debate entre uno y otro modelo de negocios. Sin embargo, este creciente interés ha planteado la disyuntiva entre el software libre "puro" y el denominado modelo de código abierto (*open source*), ya que el hacer público el código fuente es condición necesaria, mas no suficiente, para que, acorde con la definición que se ha dado, el software se considere como libre. Esta situación, sumada al hecho de que muchas firmas se desenvuelven en ambos mercados (de software libre y propietario), combinando incluso aplicaciones de uno y otro tipo en una misma distribución, hace que el umbral entre los dos enfoques sea cada vez más difícil de establecer. Para efectos del análisis económico, el concepto de código abierto parece más adecuado porque no involucra juicios normativos. Sin embargo, debido a que la GNU GPL garantiza simultáneamente que un proyecto sea de código abierto *y* de software libre, se usará este último término, procurando de paso, con esto, mantener presente la motivación ética que originó este movimiento.

[4]En el sentido de exhaustividad. A pesar de esta denominación, se puede advertir que la GNU GPL ofrece mayor protección de las libertades *de los usuarios* que la licencia BSD.

[5]Para noviembre de 2007 estaban registrados más de 160.000 en SourceForge.net, uno de los sitios web que aloja el código fuente de programas libres.

[6]Entre los ejemplos se puede citar a IBM, Sun Microsystems, Hewlett-Packard e Intel.

[7]Como Red Hat o Canonical.

3. Aspectos legales y actualidad del software en Colombia y en el resto del mundo

En Colombia, la legislación alrededor de los derechos de autor proviene principalmente de instrumentos internacionales como los de la Organización Mundial de la Propiedad Intelectual (OMPI), comenzando por la Convención Interamericana sobre el Derecho de Autor en Obras Literarias, Científicas y Artísticas de 1946, aprobada por la ley 6 de 1970, y seguida por la Convención Universal sobre Derechos de Autor de 1952-1971 y la Convención de Roma de 1961, que fueron ratificadas con la Ley 48 de 1975[8]. Posteriormente se promulgó la Ley 23 de 1982 que complementó la normatividad internacional, y se adoptó el Convenio de Berna de 1971 por medio de la Ley 33 de 1987. Sin embargo, hasta entonces nada protegía los derechos sobre la autoría del software. Sólo fue hasta la expedición del Decreto 1360 de 1989 que se le reconoció dicha protección (denominándolo *soporte lógico*) y se reglamentó su inscripción en el Registro Nacional del Derecho de Autor. En este decreto se incluyen los manuales, los esquemas de funcionamiento y la descripción de procesos como partes integrales del soporte lógico, y no establece la diferencia entre código fuente y código máquina. Sin embargo, más allá de las disposiciones anteriores, el contenido de mayor relevancia, porque marca la tendencia a seguir en la concepción legal de la naturaleza del software, es la consideración del mismo como una creación propia del dominio literario. A pesar de esto, existe cierta distancia entre un programa de computador y un libro. Para algunos son dos tipos de creaciones de características muy diferentes cuyos derechos de autor suelen ser manejados de distinta manera, como se explica más adelante[9].

La Decisión Andina 351 de 1993 estableció un régimen común para los derechos de autor relacionados con TI en la Comunidad Andina de Naciones (CAN), lo que representó un avance en materia de normatividad al precisar varios conceptos —antes inexistentes— relacionados con el software y sus usos lícitos. En ella, igualmente se consideran los programas de ordenador como creaciones literarias, pero se especifica que todo tipo de código es susceptible de protección. Adicional a esto, se explicitan varias libertades de los usuarios como la de copia y modificación del software para uso exclusivamente personal o de respaldo de información sin la necesidad de contar con autorización alguna del titular de los derechos. Posteriormente, por medio de la ley 565

[8]Curiosamente, por un vicio de forma (no se incluyó la totalidad del texto de las convenciones), estos tratados tuvieron que volver a ser ratificados 25 años después mediante el Decreto 1821 de 2000.

[9]Véase Stallman (2004, pp. 63-70) para un punto de vista sobre estas diferencias.

de 2000, se aprobó el Tratado de la OMPI sobre Derecho de Autor de 1996 que, en lo relacionado con las TI, sólo ratifica, como ya sucedía en el país, la pertenencia de los programas de ordenador, independientemente de su forma de expresión, al dominio literario.

Con la entrada en vigencia del nuevo Código Penal Colombiano en 2001, se establecieron sanciones contra el uso ilegal de software, ausentes hasta entonces, que comprenden desde multas hasta una pena máxima de cinco años de prisión (que fue aumentada a ocho años por la ley 1032 de 2006). Sin embargo, se deja claro que la reproducción, ejecución, o comercialización de software no son ilegales cuando el titular de los derechos las autoriza previa y expresamente. Es esta autorización la que pretende formalizar y reglamentar la licencia GNU GPL si bien aún no se ha puesto a prueba en los estrados judiciales[10].

Hasta hoy en Colombia no hay ninguna ley, decreto, o tratado que se refiera explícitamente al software libre o propietario. Sin embargo, actualmente se tramita un proyecto de ley[11] que intenta implementar la utilización de software libre y estándares abiertos en las entidades del Estado con el objetivo de aumentar el control que éstas poseen sobre sus sistemas y promover la igualdad de acceso a la tecnología y la no dependencia de un único proveedor. Establece una política de apoyo a proyectos de investigación y desarrollo de software libre en las entidades de carácter científico y tecnológico con el fin de favorecer la apropiación tecnológica, la inclusión digital y la integración de las comunidades. Además pretende implantar soluciones libres en las instituciones de educación pública, si bien contempla la impartición de capacitación tanto en software libre como propietario en la educación formal. Para diciembre de 2007, el proyecto había sido aprobado en primer debate por la Comisión Sexta Constitucional Permanente de la Cámara de Representantes de Colombia tras importantes modificaciones introducidas por el Ministerio de Comunicaciones, con la cuales se eliminaron las referencias al software libre y se cambió sustancialmente el texto de la propuesta original. En el resto de Latinoamérica el panorama es similar. Sin embargo, países como Brasil y Venezuela han hecho de la implementación del software libre una política de Estado como un medio para promover el desarrollo tecnológico.

La legislación colombiana permite que los autores dispongan "de su obra a título gratuito u oneroso bajo las condiciones lícitas que su libre criterio les dicte"[12], así que es

[10] Lerner y Tirole (2002a, p. 3)
[11] http://slcolombia.org/Proyecto_de_Ley (consultada el 19/02/2009).
[12] Ley 23 de 1982.

posible establecer condiciones que cedan temporal o permanentemente a otros, algunos de los derechos sobre la propiedad intelectual. Entre las opciones de *vender* y *licenciar*, la última es explotada con mucha más frecuencia en el mundo del software que en el de los medios impresos, ya que, al licenciar un programa, se abre la posibilidad de incluir límites a la responsabilidad del autor en caso de que el producto no funcione como debería o para restringir algunos derechos del usuario como, por ejemplo, el de ejecución en más de una máquina (Lerner y Tirole, 2002a). La naturaleza del software se aleja considerablemente del dominio literario a pesar de que los programadores *escriben* los programas, por ello la industria ha buscado una salida en el recurso de las licencias que le brinda la ley, pero este hecho, lejos de respaldar la identificación literaria del soporte lógico, formula interrogantes acerca de si en realidad se cuenta con un marco jurídico adecuado en la era de las TI.

La Unión Europea es quizás una de las regiones que más ha avanzado en la conformación de un marco legal alrededor del software y en donde más preocupación se ha expresado por el surgimiento de un tipo de protección distinto al del derecho de autor: las patentes. La preocupación radica en que siempre se ha considerado al software como una creación original producto del talento e ingenio de su autor, pero los impulsores de las patentes sostienen que el software es, en sí, una invención, y, por lo tanto, es susceptible de ser patentado. Hay una gran diferencia entre el derecho de autor, que protege solamente la expresión del software (el código), y la patente, que protege también las ideas fundamentales del mismo (Lerner y Tirole, 2002a). El alcance de esta iniciativa es inmenso, ya que abre las puertas a que, por ejemplo, el desarrollador de un programa de gestión de tráfico en internet pueda no sólo ejercer derechos sobre éste, sino también sobre los principios que subyacen tras su funcionamiento, de manera que nadie más podría desarrollar una aplicación que opere basada en un algoritmo similar, por más que su código sea distinto. Hasta hoy, la patentabilidad del software no ha sido reconocida en Europa. Respecto a las políticas de Estado, las iniciativas de implantación de software libre en el sector público se han aplicado mayormente en gobiernos provinciales y municipales[13].

[13]Es el caso de la comunidad autónoma de Extremadura, en España (http://www.nccextremadura.org/); la ciudad de Múnich, en Alemania (http://www.heise.de/newsticker/meldung/37126); y la provincia de Santa Fe, en Argentina (http://www.fsfla.org/?q=es/node/63)

4. Áreas de estudio

Los trabajos en economía alrededor de las tecnologías de la información son relativamente nuevos, con un número reducido de autores y muchas preguntas sin resolver. Lerner y Tirole (2002b, 2005) hacen un primer esfuerzo por comprender el fenómeno del software libre estudiando la evolución de cuatro proyectos exitosos: Apache, Linux, Perl y Sendmail. Evalúan las posibles motivaciones de corto y largo plazo que tienen los programadores para contribuir a dichos proyectos, consideran ciertas estrategias de las firmas dedicadas a escribir software propietario cuando se enfrentan al paradigma libre y sugieren varias preguntas interesantes como puntos de partida para futuros trabajos. De allí se desprenden tres vertientes de investigación: una que estudia las decisiones e incentivos de los agentes durante el proceso de desarrollo del software, otra que se enfoca en la coexistencia con el enfoque de desarrollo propietario, y, por último, la que toca temas relacionados con políticas públicas de apoyo a iniciativas libres.

Dentro del primer grupo se puede ubicar a Johnson (2002), quien analiza el problema desde la perspectiva de la contribución por parte de agentes privados a la construcción de un bien público (un proyecto de software libre). Frente a la hipótesis que sostiene que los entornos de desarrollo de código abierto adolecen de redundancia en los esfuerzos de programación, concluye que los incentivos al parasitismo impiden que esto suceda. También encuentra que el paradigma libre es socialmente más eficiente que el propietario por dos razones básicas: que las firmas no conocen las preferencias de la gente tan bien como la gente misma, y que el software libre explota un conjunto de habilidades e información privada (el de la comunidad completa de desarrolladores-usuarios) más grande que la firma. Analiza tres hechos estilizados del mundo del software: atribuye la alta especialización de las aplicaciones libres a una correlación natural entre el capital humano y las tecnologías de producción, que hace que quienes son más aptos para desarrollar aplicaciones, lo hagan en su propio campo. El hecho de que la posibilidad de agregación de pequeñas contribuciones a los proyectos desemboque en un desarrollo acelerado es explicado porque, a mayor tamaño del proyecto, el desarrollo modular e incremental facilita la contribución de programadores-usuarios individuales disminuyendo sus costos de colaborar en relación con sus valoraciones privadas, pero hace la salvedad de que el modelo modular es superado por el no modular cuando la población de programadores-usuarios es reducida, lo que explica que la mayoría de proyectos deban comenzar en un entorno relativamente más cerrado con un pequeño grupo de desarrolladores como punto de partida. Finalmente, propone que la percepción de que

el software propietario generalmente posee un mayor número de características adicionales que el software libre se debe al proceso de maximización de beneficios de la firma, donde los precios de reserva de los consumidores son aditivos respecto a las diferentes características de una aplicación.

Por otro lado, ubicándose en el siguiente grupo, Gaudeul (2005) expone un modelo en el que los esquemas libre y propietario conviven simbióticamente. La existencia del software propietario afecta las estrategias de los agentes involucrados en el desarrollo de software libre y viceversa, dando como resultado que en ambos enfoques se dediquen esfuerzos a actividades que, de otra manera, no se hubieran emprendido, pero que son de gran utilidad para los usuarios. De este modelo se derivan principalmente tres hallazgos. El primero de ellos es el de la existencia de tres tipos de equilibrios de la estructura industrial duopolística: uno de ellos, en el que el software libre captura los usuarios con una alta valoración de sus características (como en ambientes científicos o de producción) a pesar de su carencia de interfaz; el segundo, en que el software libre sirve a usuarios con una baja valoración, pero que no se pueden permitir pagar por su equivalente propietario y deben incurrir en los costos de aprendizaje (como los países en vías de desarrollo o los estudiantes); y el tercero, en que el software libre no es usado y el programa propietario captura todo el mercado. El segundo hallazgo tiene que ver con la influencia de los efectos de red sobre los incentivos de los jugadores. Subraya que estos son la causa de que bajo el paradigma libre los desarrolladores se vean inclinados a escribir programas más completos e intuitivos, involucrando no sólo sus propias preferencias, sino aquéllas de los usuarios menos sofisticados, mientras que el efecto sobre los incentivos de las firmas de software propietario es ambiguo. Como tercera conclusión, encuentra que una firma de software propietario puede optar por dos estrategias: una es implementar las mismas características del software libre, pero incluyendo una interfaz; y la otra —más beneficiosa— es desarrollar características acordes con otros usuarios cuyas preferencias no coinciden con las de los desarrolladores, caso en el cual se preserva la estructura duopolística en que uno y otro programa sirven a un tipo diferente de usuario final.

En la tercera vertiente de literatura se encuentran argumentos opuestos. Schmidt y Schnitzer (2002) concluyen que los gobiernos no deberían intervenir en la elección del tipo de software a usar en el sector público, ya que su interferencia en un mercado con altos efectos de red[14] le imprime distorsiones al equilibrio haciendo más factibles los

[14]En general, reforzados por licencias "virales" como la GPL que —sostienen— impiden que se pueda construir software propietario completamente compatible con el software libre.

resultados ineficientes. Sin embargo, incluso en presencia de efectos de red despreciables, la intervención disminuye el grado de competencia y promueve comportamientos monopolísticos y de tráfico de influencias, derivando en precios más altos para los usuarios y menores tasas de innovación. Por lo tanto, sugieren que el apoyo gubernamental debería limitarse a la investigación básica y a la conservación de estándares que hagan más compatibles el software libre y propietario para que sea el mercado, con su sistema de incentivos, quien se encargue de llegar al grado óptimo de competencia e innovación. Un resultado distinto obtienen Varian y Shapiro (2003), quienes parten del análisis de un caso específico: el sistema operativo GNU/Linux, y sostienen que éste ha alcanzado una masa crítica de usuarios que reduce los riesgos y costos de usarlo. Proponen que su utilización puede promover el desarrollo del sector software en un país y, también, el desarrollo económico del software comercial para dicha plataforma. Subrayan que los temores sobre las licencias libres son infundados, ya que éstas no requieren que un programa sea libre para funcionar sobre Linux. Encuentran que los cálculos del costo total de propiedad (CTP), entendido como la suma de los costos de adquisición, soporte, capacitación, actualización y obsolescencia de un sistema, para ambos tipos de software, arrojan cifras muy similares. Sin embargo, resaltan que, debido a que este análisis se ha hecho principalmente en los Estados Unidos, donde los costos laborales representan una gran proporción del costo total, en los países menos desarrollados, que gozan de trabajo más barato, el CTP de adoptar Linux como plataforma puede ser mucho menor que el de optar por un sistema operativo propietario como Windows. Concluyen afirmando que, para estimular el crecimiento de una fuerte industria de software, una economía debería apelar a las universidades valiéndose del hecho de que el modelo libre permite a los desarrolladores acceder a la tecnología de manera más transparente y promueve la innovación, sin ataduras a las empresas de software propietario.

Es escasa la literatura con enfoque hacia los países menos desarrollados. Tan sólo existen algunos pocos intentos por delinear la realidad del software en naciones como Colombia. Un ejemplo es Kshetri (2004), quien, en términos generales, identifica varios aspectos, tanto micro como macroeconómicos, por considerar sobre los mercados de software en las economías en desarrollo. Especialmente en lo que concierne a consideraciones macro, encuentra que estos países carecen de incentivos para hacer cumplir los acuerdos internacionales antipiratería, ya que la estrategia de contención de la copia ilegal de software propietario les reporta gastos considerables sin beneficios tangibles en términos de desarrollo económico. Sin embargo, mediante la introducción del software libre por parte de los gobiernos, estos costos se reducen, pues se brinda una

opción que, además de reemplazar en cierto grado a sus antecesores propietarios, genera procesos de aprendizaje valiosos[15]. Además, expone que, en muchas naciones, los protocolos cerrados son percibidos como una amenaza a la seguridad nacional. Es el caso de China, cuyo gobierno ha utilizado a Linux como un poderoso medio para adelantarse tecnológicamente. También resalta que, para mediados de 2002, países como Brasil, México, Argentina y Perú tenían proyectos de ley para implementar su utilización en las organizaciones del Estado. Este tipo de procesos se ha dado recientemente impulsado por una creciente mejora en la facilidad de uso de muchas herramientas libres que antes no contaban con una interfaz lo suficientemente intuitiva para usuarios menos avanzados. Finalmente, expone que, como los gobiernos son a menudo los principales usuarios de TI en los países menos desarrollados, sus decisiones van de la mano con una gran influencia para el uso de aplicaciones libres a nivel nacional. Por esto no ve tan beneficioso que comprometan su apoyo a un sólo tipo de software, sino, más bien, propone que deberían generar espacios de interacción entre ellos, teniendo en cuenta que el contexto del software libre promueve la diversidad bajo estándares abiertos.

Siguiendo la primera línea de investigación, a continuación se modelará el mercado del software bajo el enfoque de los incentivos para la escogencia de uno u otro paradigma, desde la oferta y la demanda[16] y se explorará la hipótesis de la mayor eficiencia del código abierto (el desarrollo modular de Johnson), explicándola como consecuencia de una puesta en común de conocimiento y una mayor flexibilidad en los beneficios esperados.

5. Un modelo básico

El mercado del software por modelar tiene como agentes a *desarrolladores* y *usuarios*. El concepto de desarrollador, para efectos del modelo, es muy general. Basta con que sea un agente (por ejemplo una persona, una firma, una fundación o un grupo de investigación) dedicado a "producir"[17] software que ofrece en venta[18]. Se entiende por usuario el agente cuyo uso del software le proporciona utilidad. Si bien son numerosos los casos en que ambas categorías se superponen en un mismo ente, la cualidad

[15]Señala que muchos países asiáticos tienen incluso versiones nacionales de Linux. En la actualidad, también se viene desarrollando con rapidez una distribución conjunta por parte de firmas de China, Japón, Corea y Vietnam: Asianux (http://www.asianux.com).

[16]Centrando el análisis en la oferta.

[17]Más adelante se detallan las modalidades de producción.

[18]También en un sentido amplio (como cuando no hay un precio relacionado con la transacción).

de usuario o desarrollador depende de la actividad principal que se desempeñe y de lo que se esté considerando como software, de manera que si una firma de videojuegos hace uso de una aplicación de diseño gráfico para elaborar sus modelos, se consideraría desarrolladora si el software se define como el videojuego, y usuario si se habla de la aplicación de diseño.

Apelando a la teoría de juegos, se configurarán dos modelos de elección simultánea: uno en estrategias puras y otro en estrategias mixtas. Siguiendo un análisis estricto en estrategias puras, desarrolladores y usuarios se enfrentan cada uno a la elección entre producir/usar software propietario o producir/usar software libre, bajo las definiciones de ambos tipos dadas más arriba. A partir de esta aproximación de elecciones binarias, se hará evidente la necesidad de ampliar el análisis y se dará un segundo paso en la modelación hacia una configuración más compleja. En ambos modelos se hallará el conjunto de estrategias que maximicen unilateralmente los pagos de agentes racionales participantes en un juego de decisión simultánea con información completa pero imperfecta, dadas las mejores estrategias de los contrincantes, es decir, el equilibrio de Nash[19]. Derivando este equilibrio se puede establecer un punto de referencia del mercado cuando se halla en un estado estacionario de largo plazo en que ningún agente tiene incentivos para cambiar unilateralmente de estrategia, pues dicho cambio haría que sus pagos, que se encuentran en un máximo, en el mejor de los casos, disminuyesen. Un conjunto de estrategias de equilibrio [de Nash] como éste se traduce en un estado óptimo para cada uno de los agentes *por separado*, mas no implica que el mercado haya encontrado un punto óptimo en el sentido de Pareto, al que se podría llegar mediante la cooperación. La importancia de este hecho radica en que los resultados que se expondrán se basan exclusivamente en el estudio del equilibrio o equilibrios de Nash, emanando conclusiones alrededor de la situación "óptima" a la que llega el mercado y no a algún otro conjunto de estrategias mediante el cual se pudiese alcanzar una mayor suma total de pagos, pero al que, dados los incentivos individuales, sería imposible llegar en un juego no cooperativo como el presente.

5.1. Elección binaria de los agentes

Como se mencionó antes, el desarrollador enfrenta una disyuntiva entre si escribir software propietario (SP_W), sobre el cual tendrá plenos derechos de propiedad, o escribir software libre (SL_W), que deberá hacer público bajo una licencia GPL o similar. Es

[19]Nash (1950).

claro que también existen tipos de licencias híbridas, al igual que agentes que trabajan con ambos tipos de software al tiempo, pero, para efectos del modelo, sólo se tomarán ambos extremos de la escala de libertades del software. De igual forma, se supone que el tipo de licencia que cobija el software no determina el costo fijo inicial de escribirlo, pues dos programas con los mismos requisitos de funcionamiento escritos en un mismo lenguaje demandarán exactamente el mismo esfuerzo (agregado o no) en ambos casos *en ausencia de externalidad alguna*. No obstante, a pesar de estar configurados con un costo fijo de desarrollo común, los pagos en el modelo no serán necesariamente iguales[20], y se distanciarán aun más en la versión más compleja de la Sección 5.2.

Con lo anterior, si el desarrollador opta por i) producir software propietario, incurrirá en el costo fijo de desarrollo d y, siempre y cuando el usuario opte por adquirir su producto, cobrará un precio c por el mismo, mayor a cuanto le costó producirlo. Como se ve, al elegir esta opción, el desarrollador entra a hacer parte de la industria del software *como producto* a cambio del cual percibe un ingreso. Si, en cambio, decide adoptar la estrategia de ii) producir software libre, persistirá el mismo costo de desarrollo, pero, en caso de que el usuario se incline hacia la alternativa libre, recibirá ingresos s por el valor presente del flujo de pagos futuros al *servicio* de soporte y adaptación del programa después de que éste esté en manos del usuario. Como se advierte, al tomar la decisión de producir software libre, el desarrollador interviene en la industria con un servicio *asociado* al software de libre distribución, uso y modificación, y no con un producto de un solo pago.

La decisión que toma el usuario es análoga a la del desarrollador, si bien sus estrategias son de consumo, no de producción. Lo que para el segundo es un ingreso, para el primero es un gasto. Sin embargo, éste está dispuesto a incurrir en dicho gasto porque espera recibir a cambio un beneficio de la aplicación como aumentos en productividad, mejor gestión de su negocio o, simplemente, la utilidad que le genera destinar su tiempo libre a divertirse con un juego para computador. Este beneficio derivado del uso se notará con la letra b y no depende del tipo de software por el cual se haya inclinado el usuario. Es así como, si opta por la alternativa propietaria (SP_U), recibirá un beneficio b a cambio de un precio c, y si, al contrario, escoge la alternativa libre (SL_U), recibirá el mismo b, pero a cambio del flujo futuro de pagos por el soporte s. Para que el usuario pueda obtener el beneficio de una u otra elección el tipo de software escogido debe estar siendo efectivamente producido por alguien. En caso contrario el usuario no

[20]Por ahora, por causas exógenas.

		Usuario	
		SP_U	SL_U
Desarrollador	SP_W	$c-d, b-c$	$-d, 0$
	SL_W	$-d, 0$	$s-d, b-s$

Figura 1: Juego resultante en estrategias puras.

podrá pagar por el producto/servicio, ni tampoco obtendrá beneficio alguno. En suma, sus pagos en este tipo de situaciones serán nulos. La Figura 1 ilustra el juego resultante de la elección binaria.

Sin embargo, este tipo de modelación se aleja a todas luces de la interacción real entre los agentes del mercado del software y del comportamiento de las variables en cuestión, porque, bajo restricciones impuestas a las variables[21], restringe el equilibrio a un mercado en donde sólo una de las alternativas de software sobrevive, e ignora la característica más importante que se ha resaltado desde el principio: la fuerte presencia y relevancia de los efectos de red en la industria. Para corregir esta omisión, y, al mismo tiempo, poder incluir un efecto adicional que permita modelar la existencia de una externalidad positiva dentro del paradigma de desarrollo de código abierto, será necesario ampliar el modelo hasta ahora estudiado y llegar a una versión más compleja que se acerque en mayor grado a la realidad del mercado del software.

5.2. Externalidades y efectos de red

A grandes rasgos, los efectos de red que se involucrarán al análisis pertenecen al mismo concepto genérico de externalidad. Sin embargo, se subraya la distinción de categorías con el fin de hacer explícito el tipo de agente que se ve afectado *directamente* por cada uno de los efectos. De esta forma, se entenderá por externalidad en el mercado del software la *puesta en común de conocimiento* derivada del desarrollo público bajo el paradigma libre (o de código abierto). Esta característica de la programación cooperativa hace que la acción de desarrollar un programa para el cual se ofrecerá soporte, genere un beneficio dirigido a los demás desarrolladores, disminuyendo sus costos de realizar la misma acción, ya que parte del trabajo habrá sido realizada ya por otro. De esta forma, el costo de desarrollo de una aplicación libre será inversamente proporcional al número de desarrolladores que estén trabajando simultáneamente, o hayan trabajado, en el proyecto. Es de anotar que este tipo de externalidad positiva sólo se presenta en

[21] Las necesarias para que todos los pagos sean no negativos.

el modelo libre, porque, al ser el propietario un modelo cerrado, no hay lugar para la puesta en común de conocimiento y cada desarrollador se enfrenta al reto de escribir por sí mismo todo el código. Por otra parte, el efecto de red se entenderá como una externalidad que afecta directamente al usuario y que éste tendrá en cuenta a la hora de configurar su estrategia. Como se mencionó en la introducción de este trabajo, el carácter orgánico de una aplicación como parte de un sistema interconectado de procesos crea fuertes relaciones de dependencia entre cada tipo de programas, causando que los asuntos de compatibilidad, interoperatividad y persistencia de la información jueguen un papel principal a la hora de escoger entre las alternativas que ofrece el mercado. Un usuario no le asignará el mismo valor a un cliente de correo electrónico cuando es la única persona que lo utiliza y no puede aprovechar todas sus funcionalidades disponibles para interactuar con otros programas similares, que si el resto del universo de usuarios compartiera la misma aplicación. Por tanto, se encontrará una relación directa entre la proporción de usuarios de un tipo de software y el beneficio que éstos perciben del mismo. A esto es a lo que se llamará en el modelo el *efecto de red*. Para formalizar la adición de estas novedades se apelará a un análisis que, a pesar de basarse en estrategias mixtas, se distancia del método canónico usado para formular y resolver los juegos estratégicos normales, ya que, en el juego a continuación, los perfiles de probabilidad asignados a la estrategias puras tienen un doble efecto sobre los pagos esperados, es decir, las probabilidades figuran dentro de los pagos de cada agente de forma no convencional.

Supóngase un mercado del software con una masa finita de usuarios y de desarrolladores. Imaginaremos el conjunto de posibles estrategias a que se enfrenta un agente representativo (de los desarrolladores y de los usuarios). El desarrollador escogerá producir software propietario con una probabilidad p, por lo que el software libre será producido con una probabilidad $1 - p$. Por el lado del usuario, éste escogerá el programa propietario con probabilidad q y la alternativa libre con probabilidad $1 - q$. Dado que ambos agentes representan cada uno a su respectivo universo, al escoger la distribución de probabilidad óptima sobre sus estrategias puras, *estarán también reflejando la proporción de usuarios/desarrolladores de uno y otro tipo de software*. Este resultado hace posible la coexistencia de ambos paradigmas de negocio, como sucede en la realidad, y permite involucrar la externalidad positiva en el paradigma libre, y el efecto de red en general. Para ello, comencemos analizando los nuevos pagos[22].

[22]Nótese que, en el análisis subsiguiente, el flujo total de ingresos del desarrollador ya no equivale al precio que paga el usuario, sino que el ingreso total para el productor (c o s) es resultado de

		Usuario	
		$SP_U\ (q)$	$SL_U\ (1-q)$
Desarrollador	$SP_W\ (p)$	$c-d, qb-\frac{c}{q}$	$-d, 0$
	$SL_W\ (1-p)$	$-pd, 0$	$s-pd, (1-q)b-\frac{s}{1-q}$

Figura 2: Representación normal del juego en estrategias mixtas teniendo en cuenta la puesta en común de conocimiento y los efectos de red.

Los pagos del desarrollador bajo el paradigma propietario no se ven alterados, pero bajo el enfoque libre sí lo hacen: asumiendo que existen usuarios que harán uso de su software, el desarrollador seguirá recibiendo un valor total de ingresos s, pero, debido al supuesto de la puesta en común de conocimiento inherente al modelo libre, *el costo de desarrollo en que incurre se reduce a medida que aumenta el número de colegas que trabajan conjuntamente en el mismo proyecto* $(1-p)$. Algebraicamente, el costo total para un desarrollador en código abierto tendrá la forma funcional pd.

A diferencia de lo que sucede desde la perspectiva del desarrollador, la nueva situación para el usuario de software propietario, siempre y cuando haya quien supla la demanda del mismo, afecta los resultados del juego en cada una de sus estrategias puras. Si prefiere la versión propietaria, el beneficio que perciba dependerá de qué tan popular sea el software entre sus vecinos. Por lo tanto, entre mayor participación en el mercado tenga la aplicación propietaria, mejor podrá aprovechar todas sus características; luego, *el beneficio que perciba dependerá directamente de la proporción de usuarios de software propietario* (q). Se adopta, entonces, la forma qb para expresarlo, y, simétricamente, el beneficio en caso de optar por la aplicación libre será $(1-q)b$. La representación en forma normal del juego resultante se muestra en la Figura 2.

De esta configuración formal del juego al interior del mercado del software es posible, entonces, hallar el equilibrio de Nash que surge de la interacción, y comenzar a analizarlo como referente del comportamiento de los agentes cuando carecen de incentivos para desviar unilateralmente su estrategia. Para una exposición detallada del desarrollo del modelo, refiérase al Apéndice A.

La modificación del marco de elección binaria arroja, como estaba previsto, un equilibrio que permite la coexistencia en el mercado de los dos paradigmas, ya que contempla

la agregación de los pagos de *todos* los consumidores de cada tipo de software ($\frac{c}{q}$ o $\frac{s}{1-q}$ según la estrategia), lo que hará posible el equilibrio en un mercado con costo marginal de producción nulo y elimina la posibilidad de un equilibrio en el que uno u otro tipo de software capture todo el mercado.

proporciones no negativas[23] de productores y consumidores en cada segmento:

$$\{(p, 1-p), (q, 1-q)\} = \left\{ \left(\frac{c-2d}{c-2d+s}, \frac{s}{c-2d+s} \right), \left(\frac{s}{c-2d+s}, \frac{c-2d}{c-2d+s} \right) \right\} \quad (1)$$

El sentido de los efectos de las variables exógenas sobre este equilibrio se resume en el Cuadro 1, donde se muestra el signo de las relaciones o su ausencia. El carácter inclusivo de este resultado lo convierte en un punto de paso obligatorio en el análisis subsiguiente con el fin de aclarar cómo influyen las variables de interés (d, c, s y b) en la decisión de las empresas e individuos sobre qué tipo de plataformas tecnológicas construir y en qué basar sus procesos productivos del día a día. No obstante, el equilibrio también se destaca por varias carencias: i) no predice el tamaño de la industria, puesto que la masa total de desarrolladores y usuarios es exógena, por lo que no se pueden extraer conclusiones de política respecto a incentivos para el desarrollo tecnológico o para el sector de las TI; ii) no contempla mercados con un solo tipo de software en su versión más compleja, lo que invalida las comparaciones con los casos extremos de la elección binaria; y iii) supone información completa de los agentes, condición de escaso cumplimiento en la vida real. Habida cuenta de estas limitaciones, sólo se analizarán en profundidad dos de los efectos exógenos, con el fin de centrar la atención en los resultados más útiles, confiables y menos predecibles del modelo[24].

De lo anterior se pueden extraer conclusiones que son consecuencia directa de los supuestos establecidos al formular el modelo: como se ve, el beneficio que el usuario deriva del software no influye en su elección, ya que se asume que dos versiones de la misma aplicación: una propietaria y una libre, le proporcionan exactamente la misma utilidad al usuario, independientemente del paradigma de desarrollo en que se hayan gestado. Aunque este resultado —y el supuesto que lo originó— no invalida el análisis, un refinamiento ulterior consistiría en involucrar la percepción del software que tiene el usuario, dadas las diferencias en el acceso e inversión en publicidad, en cada uno de los dos ambientes de desarrollo.

Siguiendo este mismo razonamiento, es posible explorar las consecuencias de la adición de la externalidad positiva de la puesta en común de conocimiento al modelo de desarrollo libre:

[23] Las condiciones necesarias sobre las variables para que esto sea cierto se detallan igualmente en el Apéndice A.

[24] Razones adicionales se exponen en el apéndice.

	∂d	∂c	∂s	∂b
∂p	$-$	$+$	$-$	0
∂q	$+$	$-$	$+$	0

Cuadro 1: Efectos de las variables sobre el equilibrio

Teorema 1 *A mayor costo fijo de desarrollo, mayor será la probabilidad de que el proyecto sea emprendido bajo el paradigma libre.*[25]

Es claro que, si el proyecto de software al que se enfrenta el desarrollador implica incurrir en un costo fijo cada vez más grande, éste se verá obligado, debido a la diferencia temporal entre el desarrollo y la venta de su producto a causa del *costo de la primera copia*[26] (Schmidt y Schnitzer, 2002, p. 5), a embarcarse en labores de financiación y apalancamiento de una magnitud cada vez mayor que contribuirán a desincentivar su empresa a menos que comparta el riesgo y, asimismo, sus beneficios, con sus pares. Esto tiene una gran relevancia, en el sentido de que respalda una de las ventajas que resaltan los promotores del movimiento de código abierto: la eficiencia en la gestión de proyectos de gran envergadura. De esta manera se distribuye eficientemente el costo y se hace posible la producción de software que, de otra forma, demandaría grandes cantidades de capital inicial. Aquí vemos que el enfoque de desarrollo de código abierto no es víctima del problema del parásito, como sería de esperar bajo una configuración como la de la externalidad positiva, sino que, al contrario, esta característica se aprovecha desde la óptica de la puesta en común de conocimiento, en que la ventaja esencial del paradigma no es que muchos ojos permanezcan pendientes del mismo código para hacerlo estable y seguro, sino que permite generar un modelo de negocio rentable y sostenible debido a la gran flexibilidad financiera que proporciona el poder compartir el costo fijo. Sin embargo, se debe subrayar que este costo de desarrollo no necesariamente está relacionado con la "importancia" del software para los usuarios —capturada en b—, por lo que no impide que, a pesar de un alto costo de producción, la demanda de una aplicación indispensable pudiese ser satisfecha en ausencia del paradigma libre.

El otro hallazgo tiene que ver con las condiciones de existencia del equilibrio:

Teorema 2 *La existencia del equilibrio depende de que el beneficio del desarrollador propietario exceda el costo fijo de desarrollo.*

[25] Se prueba en el Apéndice A al hallar el signo de $\frac{\partial d}{\partial p}$.

[26] Pues sólo le será posible venderlo cuando el grueso del desarrollo haya terminado y pueda ofrecer una versión estable de su sistema.

Partiendo del equilibrio en (1), y asumiendo que todas las variables exógenas tienen valores no negativos, es posible darse cuenta de que:

$$q = 1 - p \tag{2}$$

Por lo tanto, para asegurar la existencia del equilibrio debemos garantizar que, bien sea p ó q, pertenecen al intervalo $(0, 1)$[27]. Tomando q como referencia, encontramos que, para que $q > 0$:

$$s > 0 \Rightarrow c + s > 2d \tag{3}$$

La alternativa involucra considerar valores negativos de s, por lo que no se tiene en cuenta; luego, será necesario que el costo fijo de desarrollo ascienda, a lo sumo, a la media de los ingresos totales de cada tipo de desarrollador para asegurar el límite inferior del intervalo. Ahora, para que $q < 1$:

$$c > 2d \tag{4}$$

De donde se deriva que la condición en (4) es suficiente para que se cumpla el consecuente de (3) ■.

Este hallazgo sugiere que los desarrolladores propietarios esperan altos beneficios, lo que, sumado al hecho de que cada uno de ellos asume el total del costo de desarrollo, predice una industria del software propietario con flujos de ingresos mucho mayores a los que se presentan en el ámbito del software libre. Esto es razonable, porque es de esperar que los productores de software propietario tengan productos altamente diferenciados, lo cual aumenta su poder de mercado, mientras que, por el lado del paradigma libre, gobernado por un producto homogéneo, la obtención de un margen se hace mucho más difícil. La evidencia empírica actúa en favor de esta conclusión para una vasta mayoría de software. El ejemplo que primero viene a la mente está en el campo de los sistemas operativos: Windows *vs.* Linux, donde el fabricante del primero goza de una reconocida posición privilegiada en el mercado, con grandes utilidades, mientras que el segundo se comercializa en un mercado descentralizado, con muchas compañías ofreciendo sus "sabores" empaquetados en lo que se denominan *distribuciones*. Es posible encontrar servicios de soporte barato para aplicaciones libres porque, en contraste con lo que sucede con los ingresos del desarrollador propietario, s no tiene un límite inferior distinto

[27] Abierto, ya que, por construcción, el modelo con externalidades no contempla estrategias puras.

de 0. Esta característica le da mucha más flexibilidad financiera a un proyecto libre que a uno propietario, explicando el que los desarrolladores suelan ser ingenieros de grandes compañías que dedican su tiempo por fuera del trabajo a contribuir en algún proyecto de código abierto (Johnson, 2002, p. 655), esto apoya la notable incidencia del desarrollo "subterráneo" libre, pero no niega la posibilidad de crear firmas altamente rentables bajo el mismo paradigma, conclusión que obtiene soporte del hecho de que un aumento en el flujo de ingresos, por concepto de los servicios relacionados al software libre, inclina la balanza hacia la producción de aplicaciones de código abierto[28].

6. Conclusión

Los resultados de la Sección 5, derivados de la construcción de un modelo del mercado del software, pueden servir para inferir ciertas conclusiones importantes en lo que toca al comportamiento real de los agentes cuando se enfrentan en el contexto de una aplicación informática. Dichos hallazgos nos pueden dar pistas sobre cómo funciona la industria del software, las preferencias de los consumidores del mismo o las estrategias de producción, pero, dadas las limitantes del modelo, no cuentan con el potencial de arrojar luz acerca de las políticas que aplican o pueden aplicar los gobiernos en torno a las TI. No obstante, se puede abrir un campo a la profundización futura si se tiene en cuenta que la presente disertación supone que todos los agentes involucrados *conocen de la existencia de ambos tipos de software*, lo cual, en el mundo real, dista mucho de ser cierto.

Contrario a lo que se prevería, la externalidad positiva en la producción, presente dentro del paradigma de desarrollo de código abierto, no es víctima de parasitismo, pues este efecto es más que contrarrestado por la puesta en común de conocimiento. El que el paradigma de código abierto sea viable y se comporte tan bien en proyectos con altos costos de desarrollo indica que quienes impulsan este movimiento tienen un fundamento sólido de eficiencia para defenderlo y explica fenómenos como la alta especialización de las aplicaciones libres al interior de pequeños nichos de usuarios sofisticados[29] destacada por Johnson (2002), o el porqué de muchas de las acciones emprendidas por firmas del sector tecnológico en pro de la liberación a la comunidad del código de sus productos bajo una licencia tipo GPL cuando el mantenimiento de estos se hace insostenible bajo la licencia propietaria, de forma que podemos explicar fenómenos como la liberación de

[28] Aunque esto es un resultado natural provocado por la ley de la oferta.

[29] Donde los costos de desarrollo son relativamente altos comparados con el mercado potencial.

Netscape[30] (hoy Mozilla Firefox), o la de StarOffice[31] (hoy OpenOffice.org en su versión libre).

La otra razón por la cual una firma, que inicialmente regía el desarrollo de sus productos con el paradigma propietario, cambia de licencia, será entonces la transición de una operación con pérdidas a una operación mucho más flexible financieramente a raíz de la distribución de esfuerzos entre la comunidad, que muchas veces pasa de estar compuesta de simples usuarios, a canalizar el capital humano externo de los desarrolladores/usuario. A medida que la firma ve reducido su poder de mercado, su mejor estrategia deja de ser el desarrollo propietario y, cuando sus beneficios descienden por debajo de un límite, la única opción es dar el giro hacia la liberación del código. Sin embargo, el razonamiento inverso no obedece a la misma mecánica, pues, como resultado de la aplicación de la cláusula de *copyleft*, después de licenciado el software bajo la GPL, un paso atrás significaría comenzar desde cero e incurrir en todo el costo fijo de desarrollo que implicaría una aplicación similar, pero propietaria.

El software libre apenas se está dando a conocer. Aún es visto por la mayoría de personas de a pie como un tema abstruso reservado sólo para genios informáticos[32], y tal vez ése es uno de los principales obstáculos que enfrenta la expansión de la masa de usuarios. Con tan serios problemas de información es de esperar que los resultados a los que se llegó en este trabajo sigan velados para muchos casos a partir de un análisis aplicación por aplicación, pero, a medida que el software libre amplíe su difusión, podremos apreciar con un ojo científico la evolución del mercado de la información.

Referencias

FEDESOFT (2002). «Descripción del sector del software». Análisis de mercado.

Feller, Joseph y Fitzgerald, Brian (2000). «A Framework Analysis of the Open Source Development Paradigm». *Informe técnico*, University College Cork, Ireland. `http://www.csis.ul.ie/staff/bf/oss-icis00.pdf`

Free Software Foundation (2007). «GNU General Public License v. 3». `http://www.fsf.org/licensing/licenses/gpl.html`

[30] Hamerly y otros (1999).

[31] http://news.cnet.com/2100-1001-243239.html (consultada en 19/02/2009).

[32] Lo que aún es cierto para una parte importante del software que se produce en el entorno del código abierto.

García, Belfor (2004). «Software libre: política pública».

Gates, William Henry III (1976). «An open letter to hobbyists».
http://www.blinkenlights.com/classiccmp/gateswhine.htm

Gaudeul, Alexandre (2005). «Competition between open-source and proprietary software: the (LA)TEX case study». *Informe técnico*, University of East Anglia - Norwich and ESRC Centre for Competition Policy.
http://www.idei.fr/doc/conf/sic/papers_2005/gaudeul%.pdf

Hamerly, Jim; Paquin, Tom y Walton, Susan (1999). «Freeing the Source. The Story of Mozilla». En: *Open Sources: Voices from the Revolution*, pp. 197–206. O'Reilly, 1ª edición.
http://oreilly.com/catalog/opensources/book/netrev.html

Hotelling, Harold (1929). «Stability in Competition». *The Economic Journal*, **39(153)**, pp. 41–57. ISSN 00130133.
http://www.jstor.org/stable/2224214

Johnson, Justin Pappas (2002). «Open Source Software: Private Provision of a Public Good». *Journal of Economics & Management Strategy*, **11(4)**, pp. 637–662. ISSN 1058-6407.

Kshetri, Nir (2004). «Economics of Linux Adoption in Developing Countries». *IEEE Software*, **21(1)**, pp. 74–81. ISSN 0740-7459.

Lerner, Josh y Tirole, Jean (2002a). «The Scope of Open Source Licensing». *Informe técnico*, Institut d'Économie Industrielle (IDEI), Toulouse.
http://idei.fr/doc/wp/2003/scope_open_source.pdf

—— (2002b). «Some Simple Economics of Open Source». *The Journal of Industrial Economics*, **50(2)**, pp. 197–233.

—— (2005). «The Economics of Technology Sharing: Open Source and Beyond». *Journal of Economic Perspectives*, **19(2)**, pp. 99–120.

Nash, John Forbes (1950). «Equilibrium Points in n-Person Games». *Proceedings of the National Academy of Sciences of the United States of America*, **36(1)**, pp. 48–49. ISSN 00278424.
http://www.jstor.org/stable/88031

Schmidt, Klaus M. y Schnitzer, Monika (2002). «Public Subsidies for Open Source? Some Economic Policy Issues of the Software Market». *Informe técnico*, Universität München.
http://idei.fr/doc/conf/sic/papers_2002/schmidt.pdf

Stallman, Richard M. (2004). *Software libre para una sociedad libre*. Desde Abajo, Bogotá, 1ª edición.

Varian, Hal R. y Shapiro, Carl (2003). «Linux Adoption in the Public Sector: An Economic Analysis». *Informe técnico*, UC Berkeley.
http://sims.berkeley.edu/~hal/Papers/2004/linux_adoption_in_the_public_sector.pdf

Apéndice

A. Desarrollo del modelo

Asumiendo una función de pagos tipo Bernoulli, se computan los pagos esperados de cada estrategia pura para los agentes, dando como resultado que, de dedicarse a la producción de software propietario, el desarrollador percibirá un beneficio total igual a la diferencia entre el valor esperado de los ingresos por la venta del programa qc y lo que le cuesta desarrollarlo d.

$$E_D(SP_W, \alpha_U) = qc - d \tag{A.1}$$

Si, en cambio, se dedica al negocio del software libre, esperará no sólo unos ingresos análogos a los que ofrece la opción propietaria $(1-q)s$, sino que, además, su beneficio se verá aumentado gracias a la externalidad positiva que distribuye el costo de desarrollo entre todos los productores involucrados en el proyecto pd.

$$E_D(SL_W, \alpha_U) = (1-q)s - pd \tag{A.2}$$

Para el caso del usuario, los pagos esperados de sus estrategias, dada la estrategia del desarrollador son como sigue:

$$E_U(SP_U, \alpha_D) = p\left(qb - \frac{c}{q}\right) \tag{A.3}$$

$$E_U(SL_U, \alpha_D) = (1-p)\left[(1-q)b - \frac{s}{1-q}\right] \tag{A.4}$$

Al maximizar los pagos esperados de cada jugador, dada la mejor estrategia del contrincante, se obtiene el equilibrio en (1).

Donde c, b, s, $d > 0$; y, como se concluye de la prueba del Teorema 2, $c > 2d$.

Al computar las derivadas parciales de p respecto a cada una de las variables exógenas se encuentra que un aumento *cæteris paribus* en el flujo total de ingresos dentro del paradigma libre causará una disminución de la proporción de desarrolladores propietarios, ya que en:

$$\frac{\partial p}{\partial s} = \frac{2d - c}{(c - 2d + s)^2} \tag{A.5}$$

c es estrictamente mayor a $2d$, como se explica en la Sección 5.2. Prosiguiendo con el análisis, dado que los valores de las variables exógenas se suponen todos positivos, la siguiente expresión:

$$\frac{\partial p}{\partial c} = \frac{s}{(c - 2d + s)^2} \tag{A.6}$$

tiene signo positivo, indicando una relación monótona creciente entre el flujo total de ingresos del desarrollador propietario promedio y p.

Replicando los cómputos y el análisis, a la luz de las condiciones establecidas sobre c, s y d, se encuentra también que:

$$\frac{\partial p}{\partial d} = -\frac{2s}{(c - 2d + s)^2} \tag{A.7}$$

tiene signo negativo. Por otra parte:

$$\frac{\partial p}{\partial b} = 0 \tag{A.8}$$

Los resultados que arrojan (A.5) y (A.6) reflejan el comportamiento natural de la industria, en que los productores persiguen las opciones que les reportan mayor beneficio. Como vemos, este cuadro puede presentarse tanto en el mundo del software propietario como en el libre. Los resultados de (A.7) y (A.8) se tratan con mayor profundidad en la Sección 5.2.

Al ser $q = 1 - p$ como se determinó, los efectos sobre las estrategias de los usuarios son exactamente del signo contrario a aquellos efectos sobre las estrategias de los desarrolladores, siendo:

$$\frac{\partial q}{\partial s} > 0 \tag{A.9}$$

$$\frac{\partial q}{\partial c} < 0 \tag{A.10}$$

$$\frac{\partial q}{\partial d} > 0 \tag{A.11}$$

e, igual al caso de p:

$$\frac{\partial q}{\partial b} = 0 \tag{A.12}$$

Sin embargo, dado que c y s no representan exactamente el precio que enfrenta el consumidor, la posibilidad de inferencia a partir de los efectos hallados para el usuario es limitada.

Estos resultados se resumen en el Cuadro 1.