El Caldero Mágico

Eric Steven Raymond

Thyrsus Enterprises

<esr@thyrsus.com>

Traducción Borja Prieto

www.alanta.info

borja.prieto@alanta.info

Esta es la versión 3.0

Resumen

Este ensayo analiza el sustrato económico evolutivo del fenómeno open source. Primero exploro algunos mitos dominantes sobre la financiación del desarrollo de programas y la estructura de precios del software. Después presento un análisis basado en la teoría de juegos de la estabilidad de la cooperación open source. Presento nueve modelos para una financiación sostenible del desarrollo open-source; dos no comerciales y siete comerciales. Después continúo desarrollando una teoría cualitativa de cuándo es económicamente racional para el software ser cerrado. Examino algunos mecanismos novedosos adicionales que el mercado está inventando ahora para financiar desarrollos open-source comerciales, incluyendo la reinvención del sistema de patronazgo y los mercados de tareas. Concluyo con algunas tentativas de predicción del futuro.

Indistinguible de la Magia

En el mito galés, la diosa Ceridwen tenía un gran caldero que, por arte de magia, producía comida cuando se lo mandaba un hechizo conocido sólo por la diosa. En la ciencia moderna, Buckminster Fuller nos dio el concepto de "efimeralización", según el cual la tecnología se hace a la vez más eficaz y menos cara según los recursos físicos invertidos en los diseños iniciales son sustituidos por más y más contenido de información. Arthur C. Clarke conectó las dos ideas observando que "cualquier tecnología suficientemente avanzada es indistinguible de la magia".

Para mucha gente, el éxito de la comunidad open-source parece una improbable forma de magia. Software de alta calidad se materializa "gratis", lo que está bien mientras dura, pero difícilmente parece sostenible en el mundo real de competición y recursos escasos. ¿Dónde está la trampa? ¿Es el caldero de Ceridwen sólo un truco? Y si no, ¿cómo funciona la efimeralización en este contexto, es decir, qué hechizo está usando la diosa?

Más allá de los regalos de los empollones

La experiencia de la cultura open-source ha trastocado ciertamente muchas de las asunciones de la gente que aprendió sobre el desarrollo de software fuera de ella. *La Catedral y el Bazar* describió las maneras en las que el desarrollo de software cooperativo y descentralizado da un vuelco a la Ley de Brook, alcanzando niveles de fiabilidad y calidad sin precedente en proyectos individuales. *Colonizando la Noosfera* examinó la dinámica social en la que se sitúa este desarrollo de tipo "bazar", argumentando que se entiende de manera más eficaz no en términos convencionales de economía de intercambio, sino en lo que los antropólogos llaman una "cultura del regalo", en la que los miembros compiten por el estatus regalando cosas. En este ensayo comienzo explorando algunos mitos comunes acerca de la economía de la producción de software; después continúo la línea de análisis de estos ensayos adentrándome en el reino de la economía, la teoría de juegos y los modelos de negocio, desarrollando nuevas herramientas conceptuales que se necesitan para entender la manera en que la cultura del regalo de los desarrolladores open-source puede sostenerse a sí misma en una economía de intercambio.

Para seguir esta línea de análisis sin distracción, necesitaremos abandonar (o al menos acordar ignorar temporalmente) el nivel de explicación de la "cultura del regalo". *Colonizando la Noosfera* postuló que el comportamiento de cultura de intercambio aparece en situaciones en las que los bienes para la supervivencia son tan abundantes que ya no hacen el juego de intercambio interesante; pero aunque esto parece suficientemente poderoso como explicación *psicológica* del comportamiento, le falta entidad como explicación del contexto *económico* mixto en el que muchos desarrolladores open-source operan en realidad. Para la mayoría, el juego de intercambio ha perdido su atractivo pero no su poder de limitación. Su comportamiento tiene que tener el suficiente sentido en una economía de recursos materiales escasos como para mantenerlos en una zona de excedentes que soporta una cultura del regalo.

Por tanto, este ensayo considerará (completamente *dentro* del reino de la economía de escasez) los modos de cooperación e intercambio que sostienen el desarrollo open-source. Haciendo esto se responderá a la pregunta práctica "¿Cómo gano dinero con esto?" en detalle y con ejemplos. Primero, sin embargo, mostraré que mucha de la tensión que hay detrás de esta pregunta deriva de que los modelos más populares de economía de la producción de software no responden a la realidad.

(Una última observación antes de la exposición: la discusión y promoción del desarrollo open-source en este ensayo no debe utilizarse como argumento para establecer que el desarrollo cerrado es intrínsecamente malo, ni como tesis contra los derechos de propiedad intelectual en el software,

ni como una llamada altruista a “compartir”. Aunque estos argumentos son todavía adorados por una minoría en la comunidad de desarrollo open source, la experiencia desde que *La Catedral y el Bazar* fue publicado ha dejado claro que son innecesarios. Hay suficientes argumentos a favor del desarrollo open source basándonos sólo en sus beneficios económicos y técnicos: mejor calidad, mayor fiabilidad, menor coste y mayores opciones.)

La Ilusión de la Fabricación

Necesitamos comenzar observando que los programas de ordenador, como todos los otros bienes de capital o herramientas, tienen dos tipos diferentes de valor económico. Tienen *valor de uso* y *valor de venta*.

El valor de uso de un programa es su valor económico como herramienta, un multiplicador de productividad. El valor de venta de un programa es su valor como algo vendible. (En la jerga de los economistas, el valor de venta es el valor como bien final, y el valor de uso el valor como bien intermedio).

Cuando la mayoría de la gente intenta razonar acerca de la economía de la producción de software, tiende a asumir un “modelo de fábrica” que se basa en las siguientes premisas fundamentales:

1. La mayoría del tiempo de desarrollo se paga mediante el valor de venta.

2. El valor de venta del software es proporcional a su coste de desarrollo (esto es, el coste de los recursos requeridos para replicarlo funcionalmente) y a su valor de uso.

En otras palabras, la gente tiene una fuerte tendencia a suponer que el software tiene las características de valor de cualquier bien manufacturado. Pero estas suposiciones son falsas, como se demuestra fácilmente.

Primero, el código escrito para ser vendido es sólo la punta del iceberg de la programación. En la era pre-microcomputador solía ser habitual que el 90% de todo el código del mundo fuera escrito internamente en bancos y compañías de seguros. Probablemente ya no sucede esto; otros sectores hacen un uso mucho más intenso del software ahora, y la parte del sector financiero con respecto al total debe haber caído, pero veremos pronto que hay evidencia empírica de que aproximadamente el 95% del código todavía está desarrollado internamente.

Este código incluye la mayoría de temas de MIS , y las personalizaciones de software financiero y de bases de datos que todas las compañías medianas y grandes necesitan. Incluye código técnico especializado como los drivers de dispositivos. Casi nadie gana dinero vendiendo drivers, un asunto al que volveremos más adelante. Incluye todo tipo de código embebido para nuestras máquinas que cada vez más están controladas por microchips, desde herramientas de maquinaria y aviones a reacción hasta microondas y tostadoras.

La mayoría de este código está integrado en su entorno de maneras que hacen su reutilización o copia muy difícil. (Esto es cierto tanto si el entorno es un conjunto de procedimientos de negocio como el sistema de inyección de combustible de una cosechadora). Así, cuando el entorno cambia, se requiere trabajar continuamente para mantener el software a la par.

A esto se le llama “mantenimiento”, y cualquier ingeniero de software o analista de sistemas le dirá que conforma la inmensa mayoría (más del 75%) de lo que los programadores hacen para que les paguen. Igualmente, la mayoría de las horas de programador se gastan (y la mayoría de los sueldos de programador se pagan) para escribir o mantener código interno que no tiene ningún valor de venta; un hecho que el lector puede comprobar fácilmente examinando las ofertas de trabajo de programador en cualquier periódico con sección de Ofertas de Empleo.

Ojear la sección de empleo del periódico local es un experimento ilustrativo que conmino al lector a

realizar por sí mismo (o por sí misma). Examine los trabajos que aparecen con el título programador, proceso de datos e ingeniería de software para identificar puestos que implican desarrollo de software. Clasifique cada trabajo según el software sea desarrollado para uso interno o para venta.

Rápidamente aparecerá claro que, incluso utilizando la definición más amplia de "para la venta", por lo menos 19 de cada 20 de los salarios ofertados se financian estrictamente con su valor de uso (esto es, su valor como bien intermedio). Esta es nuestra razón para creer que sólo un 5% de la industria está dirigida por el valor de venta. Observe, sin embargo, que el resto del análisis en este ensayo es relativamente indiferente a este número; si fuera un 15% o incluso un 20%, las consecuencias económicas permanecerían esencialmente las mismas.

Cuando hablo en conferencias técnicas, normalmente empiezo mi charla preguntando dos cosas: cuántos en la audiencia cobran por escribir software, y para cuántos sus salarios dependen del valor de venta del software. Normalmente consigo un bosque de manos para la primera pregunta, algunas o ninguna para la segunda, y una sorpresa considerable en la audiencia por la proporción.

Segundo, la teoría de que el valor de venta del software está ligado a sus costes de desarrollo o sustitución es todavía más fácilmente demolida examinando el comportamiento real de los clientes. Hay muchos bienes para los que una proporción de este tipo se mantiene (antes de la depreciación): alimentos, coches, máquinas. Hay incluso muchos bienes intangibles para los que el valor de venta está ligado fuertemente a los costes de desarrollo y sustitución: derechos de reproducción de música, mapas o bases de datos, por ejemplo. Estos bienes pueden retener o incluso incrementar su valor de venta después de que su fabricante original haya desaparecido.

En cambio, cuando un fabricante de productos de software deja el negocio (o si meramente discontinúa el producto), el precio máximo que los clientes pagarán por él cae rápidamente a casi cero, independientemente de su valor de uso teórico o el coste de desarrollo o un equivalente funcional. (Para comprobar esta afirmación, puede ver los estantes de saldos de cualquier tienda de software cercana).

El comportamiento de los distribuidores cuando un fabricante cierra es muy revelador. Nos dice que ellos saben algo que los fabricantes no. Lo que saben es esto: el precio que un cliente pagará está limitado en la práctica por el *valor futuro esperado del servicio del fabricante* (donde "servicio" se usa aquí de manera amplia para incluir mejoras, actualizaciones y nuevos proyectos).

En otras palabras, el software es principalmente una industria de servicios que opera bajo la persistente pero infundada ilusión de que es una industria de fabricación.

Merece la perna examinar por qué normalmente tendemos a creer lo contrario. Puede ser simplemente porque la pequeña parte de la industria del software que fabrica para la venta es también la única parte que anuncia sus productos. La tendencia mental común que considera la fabricación más "real" que los servicios, porque produce cosas que puedes tocar, puede estar actuando aquí [WG]. También, algunos de los productos más visibles y más fuertemente anunciados son efímeros, como los juegos, que tienen pocos requisitos de servicios continuados (la excepción, más que la regla) [SH].

También merece la pena observar que la ilusión de la facturación favorece estructuras de precios que están patológicamente desalineadas con la composición actual de los costes de desarrollo. Si (como se acepta generalmente) más de un 75% de los costes del ciclo de vida de un proyecto típico de software se dedicarán a mantenimiento, depuración y extensiones, entonces la política habitual de precios de cargar un coste de adquisición alto y tarifas de mantenimiento relativamente bajas o incluso cero está abocada a conducir a resultados negativos para todas las partes.

Los consumidores pierden porque, incluso aunque el software es una industria de servicios, los incentivos en el modelo de fábrica trabajan en contra de que un fabricante ofrezca un servicio *competente*. Si el dinero del fabricante viene de vender bits, la mayoría del esfuerzo irá a fabricar

bits y echarlos por la puerta; el centro de soporte a usuarios, que no es un centro de beneficio, será un vertedero donde colocar a los empleados menos eficaces y tendrá los mínimos recursos necesarios como para no perder un número crítico de clientes.

Todavía es peor. El uso real significa llamadas de soporte, que minan el margen de beneficio a menos que cobre por el servicio. En el mundo open-source, buscará la base de usuarios mayor posible, para tener el máximo feedback y los mercados secundarios más vigorosos posible; en el mundo cerrado buscará tantos compradores pero tan pocos usuarios como sea posible. Por tanto la lógica del modelo de fábrica recompensa con mayor fuerza a los fabricantes que producen shelfware : software con un marketing suficientemente bueno como para conseguir ventas pero inusable en la práctica.

La otra cara de esta moneda es que la mayoría de los fabricantes que creen en este modelo fracasarán a largo plazo. Financiar gastos de soporte que crecen indefinidamente por un precio fijo es sólo viable en un mercado que se expande tan rápido como para cubrir los costes de soporte y de ciclo de vida que implican las ventas de ayer con los ingresos de mañana. Una vez que el mercado madura y las ventas se desaceleran, la mayoría de los fabricantes no tendrán más opción que cortar los gastos dejando huérfano el producto [DPV].

Tanto si esto se hace explícitamente (discontinuando el producto) o implícitamente (haciendo el soporte difícil de obtener), tiene el efecto de derivar a los clientes hacia la competencia, porque destruye el valor futuro esperado del producto, que es dependiente de este servicio. A corto plazo, uno puede escapar de esta trampa haciendo que las versiones de corrección de errores parezcan nuevos productos con un nuevo precio aparejado, pero los consumidores se cansan rápidamente de esto. A largo plazo, por tanto, la única manera de escapar es no tener competidores: esto es, tener un monopolio efectivo en el propio mercado. Al final, sólo puede haber uno.

Y, sin duda, hemos visto repetidamente a este fallo de privación de soporte matar incluso a competidores fuertes que tenían una segunda posición en un nicho de mercado. (El patrón debería ser particularmente obvio para cualquiera que haya analizado la historia de los sistemas operativos de PC, los procesadores de texto, programas de contabilidad, o software de negocios en general.) Los incentivos perversos establecidos por el modelo de fábrica conducen a una dinámica de mercado del tipo “el ganador se lleva todo” en la que incluso los clientes del ganador salen perdiendo.

¿Y si no vale el modelo de fábrica, entonces cuál? Para manejar la estructura real de costes del ciclo de vida del software de manera eficiente (tanto en el sentido informal como en el económico de “eficiencia”), necesitamos una estructura de precios basada en contratos de servicios, suscripciones, y un intercambio *continuo* de valor entre el fabricante y el cliente. Esta es ya la estructura de precios de los mayores fabricantes de productos de software como los sistemas ERP (Planificación de Recursos Empresariales), para los que los costes de desarrollo son tan grandes que ningún precio fijo de compra podría cubrirlos; compañías como Baan y Peoplesoft en realidad ganan su dinero en las tarifas de consultoría post-venta. Bajo las condiciones de búsqueda de eficiencia en el mercado libre podemos predecir que este es el tipo de estructura de precios que la mayor parte de una industria del software madura seguirá al final.

Lo anterior comienza a darnos alguna intuición acerca de por qué el open-source, cada vez más, aparece no solamente como un reto tecnológico sino también económico al orden establecido. El efecto de hacer el software “gratis”, parece, es obligarnos a entrar en este mundo dominado por las tarifas de servicios; y desvelar que apoyo más débil era después de todo la propuesta de valor de venta de los bits secretos del software cerrado.

Esta transición no será tan violenta como parece en un principio. Muchos consumidores saben que las copias piratas del software empaquetado (especialmente juegos, sistemas operativos y herramientas de productividad populares) son muy fáciles de conseguir. Por tanto, el precio de venta del software propietario, desde el punto de vista del consumidor, sólo merece la pena ser

pagado si se obtienen otros bienes: soporte del fabricante, manuales en papel, o un sentimiento de virtud. Las distribuciones comerciales del llamado “software libre” a menudo justifican su precio a los clientes exactamente de la misma manera; la única diferencia es que sus fabricantes no se engañan pensando que los bits por sí mismos tienen valor para el cliente.

El término “gratis” es equívoco en otro sentido también. Bajar el coste de un bien tiende a incrementar, más que decrementar, la inversión total en las personas y la infraestructura que lo soportan. Cuando el precio de los coches baja, la demanda de mecánicos sube; lo que significa que incluso ese 5% de programadores que ahora recibe compensación por el valor de venta tiene muy pocas probabilidades de sufrir en un mundo open-source. La gente que pierda en la transición no serán programadores, serán inversores que han apostado en estrategias de código cerrado cuando no son económicamente viables.

El mito “La Información quiere ser gratis”

Hay otro mito, igual y opuesto a la ilusión del modelo de fábrica, que a menudo confunde la opinión de la gente acerca de la economía del software open source. Es el que establece que “la información quiere ser gratis”. Normalmente esto conduce a la pretensión de que como el coste marginal de la reproducción digital es cero esto implica que su precio de salida debe ser cero (o que un mercado lleno de duplicadores le obligará a ser cero).

Es verdad que algunos tipos de información quieren ser libres, en el sentido débil de que su valor crece cuanta más gente tenga acceso a ella: un documento técnico de estándares es un buen ejemplo. Pero el mito de que *toda* la información quiere ser libre se viene abajo al considerar el valor de la información que constituye un indicador privilegiado para conseguir un bien valioso: el mapa de un tesoro, por ejemplo, o el número de cuenta de un banco suizo, o un derecho para acceder a servicios tal como una contraseña de una cuenta de ordenador. Incluso aunque la información para cobrar pueda duplicarse a coste cero, el bien que se consigue con ella no. De aquí que el coste marginal distinto de cero para el bien pueda ser heredado por la información que sirve para conseguirlo.

Mencionamos este mito principalmente para afirmar que casi no tiene relación con los argumentos de utilidad económica para el open source; como veremos más tarde, éstos en general se sostienen incluso bajo la suposición de que el software en realidad *tiene* la estructura de valor de un bien manufacturado. Por tanto no tenemos necesidad de abordar la cuestión de si el software “debería” ser libre o no.

Los Comunes Inversos

Habiendo echado un ojo escéptico en un modelo prevalente, veamos si podemos construir una explicación económica afinada acerca de lo que hace a la cooperación open source sostenible.

Esta es una cuestión que requiere ser examinada en un par de niveles diferentes. En un nivel, necesitamos explicar el comportamiento de los individuos que contribuyen a los proyectos open source; en otro, necesitamos entender las fuerzas económicas que sostienen la cooperación en proyectos open source como Linux o Apache.

De nuevo, primero debemos demoler un modelo popular que interfiere con nuestra comprensión. Sobre cada intento de explicar el comportamiento cooperativo se cierne la sombra de la “tragedia de los comunes” de Garret Hardin.

Hardin nos pide que imaginemos un campo que es propiedad común de un pueblo de campesinos,

que llevan a su ganado a pastar en él. Pero el pastar degrada el campo, arrancando la hierba y dejando trozos de barro, donde la hierba tarda en volver a crecer. Si no hay ninguna política acordada (e impuesta) para asignar derechos de pasto que eviten la sobreexplotación, todos los incentivos empujan a llevar tanto ganado y tan rápidamente como sea posible, intentando extraer el máximo valor antes de que el campo común se degrade hasta convertirse en un mar de barro.

La mayoría de la gente tiene un modelo intuitivo del comportamiento cooperativo que funciona de manera muy semejante a este. La tragedia de los comunes en realidad surge de dos problemas ligados, uno de sobreexplotación y otro de infraprovisión. Por el lado de la demanda, la situación de los comunes anima a una carrera hasta el fin hacia la sobreexplotación: lo que los economistas llaman un problema de bien público congestionado. Por el lado de la oferta, los comunes recompensan el comportamiento egoísta, eliminando o disminuyendo los incentivos para que los actores individuales inviertan en desarrollar más pastos.

La tragedia de los comunes predice sólo tres posibles resultados. Uno es el mar de barro. Otro es que algún actor con poder coercitivo imponga una política de asignaciones por el bien del pueblo (la solución comunista). El tercero es que el campo común se deshaga cuando los habitantes del pueblo vallen parcelas que pueden defender y gestionar de manera sostenible (la solución de los derechos de propiedad).

Cuando la gente aplica este modelo a la cooperación open source, esperan que sea inestable y que tenga una vida corta. Como no hay una manera evidente de imponer una política de asignación para el tiempo de programador en Internet, este modelo conduce directamente a la predicción de que el común se romperá, varios fragmentos de software se volverán cerrados y decrecerá rápidamente la cantidad de trabajo dedicada al fondo común.

En realidad, es empíricamente claro que la tendencia es opuesta a esto. La tendencia en amplitud y volumen de desarrollo open source puede medirse por las remisiones diarias a Metalab y SourceForge (los sitios líderes en código para Linux) o los anuncios por día en freshmeat.net (un sitio dedicado a anunciar nuevas versiones de software). El volumen en ambos está creciendo constante y rápidamente. Claramente hay un aspecto crítico en el que el modelo de la “tragedia de los comunes” falla a la hora de interpretar lo que está pasando en realidad.

Parte de la respuesta, por supuesto, yace en el hecho de que el uso del software no decrementa su valor. Sin duda, el uso extendido del software open source tiende a incrementar su valor, cuando los usuarios aportan sus propias mejoras (parches en el código). En este común inverso, la hierba crece más alta cuando se pasta.

Que este bien público no pueda degradarse por el sobreuso da cuenta de la mitad de la tragedia de Hardin, el problema de los bienes públicos congestionados. No explica por qué el open source no sufre infraprovisión. ¿Por qué la gente que sabe que la comunidad open source existe no muestra universalmente un comportamiento egoísta, esperando a que otros hagan el trabajo que necesitan o (si hacen el trabajo por sí mismos) sin molestarse en contribuir con su trabajo al común?

Parte de la respuesta radical en el hecho de que la gente no necesita simplemente soluciones, necesita soluciones *a tiempo*. Raramente es posible predecir cuándo otro terminará una pieza dada de trabajo necesario. Si el pago por arreglar un bug o añadir una funcionalidad es suficiente para cualquier contribuyente potencial, esa persona lo hará (y en ese punto el hecho de que los demás sean egoístas se hace irrelevante).

Otra parte de la respuesta yace en el hecho de que el valor de mercado putativo de los pequeños parches a un código base común es difícil de capturar. Suponiendo que escriba un arreglo para un defecto irritante, y suponiendo que mucha gente se da cuenta de que el arreglo tiene un valor monetario: ¿cómo cobro a toda esa gente? Los sistemas de pago convencionales tienen tantas complicaciones que hacen de esto un verdadero problema para el tipo de micropagos que habitualmente serían apropiados.

Puede ser más acertado incluso observar que este valor no sólo es difícil de cobrar, sino que generalmente es incluso difícil de *asignar*. Como experimento mental, supongamos que Internet viniera equipado con el sistema seguro de micropagos teóricamente ideal, accesible universalmente, sin sobre costo. Ahora digamos que ha escrito un parche titulado "Arreglos diversos al kernel de Linux". ¿Cómo sabe qué precio pedir? ¿Cómo podría un comprador potencial, que todavía no ha visto el parche, saber cuál es un precio razonable?

Lo que tenemos aquí es casi como una imagen en un espejo deformante del "problema de cálculo" de F. A. Hayek: se requeriría un superser, a la vez capaz de evaluar el valor funcional de los parches y respetado como para definir los precios en consecuencia, para lubricar el negocio.

Desafortunadamente, hay una tremenda escasez de superseres, de modo que al autor del parche J. Cualquier Hacker le quedan dos opciones: sentarse en el parche o echarlo al común gratis.

Sentándose en el parche no gana nada. Sin duda, incurre en un coste futuro: el esfuerzo que implica refundir el parche en el código base en cada nueva versión. Así que el pago para esta opción es en realidad negativo (y está multiplicado por la rápida velocidad de publicación característica de los proyectos open source).

Para expresarlo de manera positiva, el contribuyente gana trasladando las cuestiones de mantenimiento del parche a los propietarios del código fuente y el resto del grupo del proyecto. También gana porque otros mejorarán su trabajo en el futuro. Finalmente, ya que no tiene que mantener el parche por sí mismo, puede ser capaz de disponer de más tiempo para otras personalizaciones incluso mayores que se ajusten a sus necesidades. Los mismos argumentos que favorecen la apertura del código para los paquetes completos se aplican también a los parches.

Echando el parche al común puede no ganar nada, o puede animar el esfuerzo recíproco de otros que resolverán alguno de los problemas de J. Cualquiera en el futuro. Esta opción, aparentemente altruista, es en realidad óptimamente egoísta en el sentido de la teoría de juegos.

Al analizar esta clase de cooperación, es importante observar que aunque hay un problema de egoísmo (el trabajo puede estar infraprovisionado en ausencia de dinero o compensación equivalente) éste no escala con el número de usuarios finales (vea la nota final en [ST] para su discusión). La complejidad y el sobrecosto de comunicación en un proyecto open source es casi por entero una función del número de desarrolladores involucrado: tener más usuarios finales que nunca miran el código fuente cuesta en la práctica nada. Puede incrementar el número de preguntas tontas que aparecen en las listas de correo del proyecto, pero esto se mitiga fácilmente manteniendo una lista de Preguntas Frecuentes e ignorando descaradamente a los que preguntan sin haberla leído (y de hecho ambas prácticas son típicas).

Los verdaderos problemas de egoísmo en el software open source son más una función de los costes de fricción al remitir parches que otra cosa. Un contribuidor potencial con poco terreno en el juego de la reputación cultural (ver *Colonizando la Noosfera*) puede, en ausencia de compensación económica, pensar "no merece la pena remitir este arreglo porque tendré que adecentar el parche, escribir una entrada en el ChangeLog, firmar los papeles de asignación de la FSF..." Es por esta razón que el número de contribuyentes (y, en segundo orden, el éxito de los proyectos) está fuertemente e inversamente correlacionado con el número de pasos por los que cada proyecto obliga a pasar a un usuario que quiere contribuir. Estos costes de fricción pueden ser políticos o mecánicos. Juntos, creo que explican por que la cultura floja y amorfa de Linux ha atraído a órdenes de magnitud más de energía cooperativa que los esfuerzos más estrechamente organizados y centralizados de BSD; y por qué la Free Software Foundation ha retrocedido en importancia relativa con respecto al ascenso de Linux.

Todo esto está bien hasta aquí. Pero es una explicación a posteriori de qué hace J. Cualquier Hacker con su parche después de que lo ha creado. La otra mitad que necesitamos es una explicación económica de cómo JCH pudo escribir ese parche en primer lugar, en lugar de tener que trabajar en un programa cerrado que podría haberle devuelto un valor de venta. ¿Qué modelos de negocio crean

nichos en los que el desarrollo open source puede florecer?

Razones para cerrar los Fuentes

Antes de clasificar los modelos de negocio del open source, deberíamos tratar el tema de los beneficios de la exclusión en general. ¿Qué estamos protegiendo exactamente cuando cerramos el código fuente?

Digamos que contrata a alguien para escribir (por ejemplo) un paquete de contabilidad especializado para su negocio. Ese problema no se resolverá mejor si los fuentes están cerrados en lugar de abiertos; los únicos motivos racionales por los que querría que fueran cerrados es si quiere vender el paquete a otra gente, o negar su uso a los competidores.

La respuesta obvia es que lo que está protegiendo es el valor de venta, pero para el 95% del software escrito para uso interno esto no se aplica. Así que ¿qué otras ganancias hay en que esté cerrado?

La segunda opción (proteger la ventaja competitiva) merece un poco de examen. Suponga que abre el código del paquete de contabilidad. Se hace popular y se beneficia de las mejoras hechas por la comunidad. Ahora, su competidor también empieza a usarlo. El competidor tiene el beneficio sin pagar el coste de desarrollo y le quita negocio. ¿Es este un argumento contra la apertura de fuentes?

Tal vez… y tal vez no. La verdadera pregunta es si su ganancia por repartir la carga de desarrollo excede su pérdida debida al incremento de competición del egoísta. Mucha gente tiende a razonar pobremente acerca de este trueque (a) ignorando la ventaja funcional de reclutar más ayuda para el desarrollo y (b) no tratando los costes de desarrollo como ya incurridos. Por hipótesis, tiene que pagar los costes de desarrollo de todas maneras, luego contabilizarlos como coste de abrir los fuentes (si elige hacer esto) es erróneo.

Otra razón citada a menudo es el miedo a que la difusión de los fuentes de una función especial de contabilidad pueda ser equivalente a revelar aspectos confidenciales de su plan de negocio. Este realmente es un argumento no para cerrar el código, sino contra el mal diseño; en un paquete de contabilidad bien escrito, el conocimiento del negocio no debe expresarse de ningún modo en el código sino en el esquema o lenguaje de especificación implementado por el motor de contabilidad (para un caso paralelo, considere la manera en que los esquemas de bases de datos separan el conocimiento del negocio de la mecánica del motor de la base de datos). La separación de la función debería permitirle guardar las joyas de la corona (el esquema) al tiempo que obtiene el máximo beneficio de abrir el código fuente del motor.

Hay otras razones para cerrar el código que son categóricamente irracionales. Puede, por ejemplo, trabajar bajo la ilusión de que cerrar el código hará que sus sistemas de negocio sean más seguros frente a crackers e intrusos. Si es así, le recomiendo una conversación terapéutica con un experto en criptografía inmediatamente. Los paranoicos profesionales saben que no se puede confiar en la seguridad de los programas cerrados, porque han aprendido de experiencias duras. La seguridad es un aspecto de la fiabilidad; sólo los algoritmos e implementaciones que han sido revisados a fondo entre colegas pueden ser considerados seguros.

Modelos de financiación de valor de uso

Un hecho clave que la distinción entre uso y valor nos permite observar es que sólo el *valor de venta* está amenazado por el cambio de cerrado a abierto; el valor de uso no.

Si el valor de uso más que el valor de venta es en realidad el mayor conductor del desarrollo de software, y (como se argumentó en *La Catedral y el Bazar*) el desarrollo open source es en realidad más eficaz y eficiente que el desarrollo cerrado, entonces debemos esperar encontrar circunstancias en las que sólo el valor de uso esperado baste para financiar el desarrollo open source.

Y de hecho no es difícil identificar al menos dos modelos importantes en los que los salarios de desarrolladores a tiempo completo para proyectos open source están financiados estrictamente a partir del valor de uso.

El caso de Apache: compartir el coste

Supongamos que trabaja para una firma que tiene un requerimiento crítico para el negocio de un servidor web de alta fiabilidad y capacidad. Quizá es para comercio electrónico, quizá se trata de una publicación de gran visibilidad que vende publicidad, quizá es un portal. Necesita disponibilidad 24x7, necesita velocidad y necesita personalización.

¿Cómo puede conseguir esto? Hay tres estrategias básicas que puede seguir:

Comprar un servidor web propietario. En este caso, está apostando por que la agenda del fabricante coincide con la suya y que el fabricante tiene la competencia técnica para implementar adecuadamente. Incluso suponiendo que estas cosas sean ciertas, es probable que el producto quede corto en personalización; será capaz de modificarlo sólo mediante los enlaces que el fabricante haya elegido proporcionarle. Podemos ver en los estudios mensuales de Netcraft que este camino propietario no es el más popular, y se hace menos popular cada vez.

Lanzar el suyo. Construir su propio servidor web no es una opción para descartar a priori; los servidores web no son muy complejos, ciertamente menos que los navegadores, y uno especializado puede ser muy delgado y pequeño. Siguiendo este camino, tendrá exactamente las funcionalidades y personalización que quiere, aunque pagará por ello en tiempo de desarrollo. Su empresa también podría descubrir que tiene un problema cuando se retire o deje el puesto.

Unirse al grupo de Apache. El servidor Apache fue construido por un grupo de webmasters conectado por Internet que se dio cuenta de que era más inteligente unir sus esfuerzos para mejorar un código base que desarrollar una gran número de esfuerzos de desarrollo paralelos. Haciendo esto fueron capaces de capturar la mayoría de las ventajas del desarrollo propio y el poderoso efecto de depuración de la revisión entre pares masivamente paralela.

La ventaja de la opción Apache es muy fuerte. Exactamente cómo de fuerte, podemos juzgarlo por el estudio mensual de Netcraft, que ha mostrado que Apache está ganando cuota de mercado constantemente frente a todos los servidores web propietarios desde su nacimiento. En noviembre de 2.000, Apache y sus derivados tenían el 60% de cuota de mercado: sin propietario legal, sin promoción, y sin organización de servicios contratada detrás de todo ello.

La historia de Apache se generaliza en un modelo en el que usuarios de software que compiten entre sí encuentran una ventaja en financiar cooperativamente el desarrollo open source porque al hacerlo así obtienen un producto mejor, a menor coste, del que podrían tener de otra manera.

El caso de Cisco: Extender el riesgo

Hace algunos años, a dos programadores de Cisco (el fabricante de equipos de red) les asignaron el trabajo de escribir un sistema de impresión por colas distribuido para su uso en la red corporativa de Cisco. Esto suponía un desafío importante. Además de soportar la capacidad del usuario arbitrario A para imprimir en la impresora arbitraria B (que podría estar en la habitación contigua o a miles de kilómetros), el sistema tenía que asegurarse de que en el caso de falta de papel o de toner el trabajo se redirigiera a una impresora alternativa cercana al objetivo. El sistema también necesitaba ser capaz de reportar estos problemas a un administrador de impresoras.

La pareja salió con un inteligente conjunto de modificaciones al software de colas de impresión estándar de Unix, más algunos scripts, que hacían el trabajo. Entonces se dieron cuenta de que ellos, y Cisco, tenían un problema.

El problema era que no era probable que ninguno de ellos permaneciera en Cisco para siempre. En algún momento ambos programadores se habrían ido, y el software estaría sin mantener y comenzaría a corromperse (esto es, a perder sintonía con las condiciones del mundo real). Ningún desarrollador quiere ver que esto le sucede a su trabajo, y el intrépido dúo sentía que Cisco había pagado por una solución bajo la expectativa razonable de que sobreviviría a su propia situación de empleados allí.

De acuerdo con esto, hablaron con su jefe y le pidieron que autorizara la liberación del software de colas de impresión como open source. Su argumento era que Cisco no tenía ningún valor de venta que perder, y sí mucho que ganar. Animando el crecimiento de una comunidad de usuarios y co-desarrolladores extendida en muchas empresas, Cisco podría protegerse efectivamente contra la pérdida de los desarrolladores originales del software.

La historia de Cisco muestra que el open source puede funcionar no sólo para reducir los costes sino para repartir y mitigar el riesgo. Todas las partes encuentran que la apertura del código, y la presencia de una comunidad cooperativa financiada por múltiples fuentes independientes de ingresos, proporciona una protección frente a fallos que es en sí misma económicamente valiosa: suficientemente valiosa como para atraer financiación.

Por qué el valor de venta es problemático

El open source hace bastante difícil capturar el valor de venta directo del software. La dificultad no es técnica; el código fuente no es ni más ni menos fácil de copiar que el binario, y la imposición de las leyes de copyright y licencia para capturar el valor de venta no tiene que ser necesariamente más difícil para los productos open source que para los cerrados.

La dificultad radica más bien en la naturaleza del contrato social que soporta el desarrollo open source. Por tres razones que se refuerzan entre sí, las principales licencias open source prohíben la mayoría de tipos de restricciones en el uso, redistribución y modificación que podrían facilitar la captura de beneficios por la venta directa. Para entender estas razones, debemos examinar el contexto social en el que las licencias evolucionaron: la cultura hacker de Internet.

A pesar de los mitos acerca de la cultura hacker que todavía están muy extendidos fuera de él, ninguna de estas razones tiene que ver con la hostilidad hacia el mercado. Mientras que una minoría de hackers sin duda permanece hostil a la motivación del beneficio, la disposición general de la comunidad para cooperar con paquetes comerciales Linux como Red Hat, SuSE y Caldera demuestra que la mayoría de los hackers trabajarán felizmente con el mundo corporativo cuando sirva a sus intereses. Las verdaderas razones de los hackers para recelar de las licencias de captura directa del beneficio son más sutiles e interesantes.

Una razón tiene que ver con la simetría. Mientras que la mayoría de los desarrolladores open source no tiene intrínsecamente nada que objetar a que otros se beneficien de sus regalos, también demanda que nadie (con la posible excepción del creador de una pieza de código) esté en una posición privilegiada para extraer beneficios. J Cualquier Hacker está dispuesto a que UnaSA se beneficie vendiendo su software o parches, pero sólo si el mismo JCH puede también hacerlo.

Otra tiene que ver con las consecuencias no deseadas. Los hackers han observado que las licencias que incluyen restricciones y tarifas para uso comercial o venta (la manera más normal de intentar recapturar el valor directo de venta, y que en principio parece razonable) tienen serios efectos de congelación. Uno específico es arrojar una sombra legal sobre actividades como la redistribución en recopilaciones baratas en CD-ROM, que idealmente tenderíamos a fomentar. Más generalmente, las restricciones en el uso/venta/modificación/distribución (y otras complicaciones en el licenciamiento) imponen un sobrecosto para el seguimiento de la conformidad y (según aumenta el número de paquetes que maneja una persona) una explosión combinatoria de la incertidumbre percibida y los riesgos legales potenciales. Este resultado se considera perjudicial, y por tanto hay una fuerte presión social para mantener las licencias simples y libres de restricciones.

La razón última y más crítica tiene que ver con la preservación de la dinámica de revisión entre pares y cultura del regalo descrita en *Colonizando la Noosfera* [HtN]. Las restricciones de licencia diseñadas para proteger la propiedad intelectual o capturar el valor directo de venta a menudo tienen el efecto de hacer legalmente imposible ramificar el proyecto. Este es el caso, por ejemplo, con las licencias de Sun llamadas “Fuentes de la Comunidad” para Jini y Java. Aunque la bifurcación se ve con recelo y se considera el último recurso (por razones discutidas en profundidad en *Colonizando la Noosfera*), se considera críticamente importante que ese último recurso esté presente en caso de incompetencia o defección del mantenedor (por ejemplo, para pasar a una licencia cerrada).

La comunidad hacker ha cedido algo en la cuestión de la simetría: así, tolera licencias como la Licencia Pública de Netscape (NPL) que concede algunos privilegios a los creadores del código (específicamente en el caso de NPL, el derecho exclusivo a usar el código fuente de Mozilla en productos derivados, que pueden incluir código cerrado). Ha cedido menos en la cuestión de las consecuencias no deseadas, y nada en absoluto en preservar la opción de bifurcar (lo cual es la razón de que las Licencias de Fuente de Comunidad para Java y Jini de Sun hayan sido ampliamente rechazadas por la comunidad).

(Merece la pena repetir aquí que nadie en la comunidad hacker *quiere* que los proyectos se dividan en líneas de desarrollo competidoras; sin duda (como observé en *Colonizando la Noosfera*) hay una presión social muy fuerte contra la bifurcación, y por buenas razones. Nadie quiere estar en un piquete, en un juzgado o disparando. Pero el derecho a bifurcar es como el derecho a la huelga, a litigar o el derecho a llevar armas: no quiere ejercitar ninguno de estos derechos, pero es una señal seria de peligro que cualquiera intente quitárselos).

Estas razones explican las cláusulas de la Definición Open Source, que fue escrita para expresar el consenso de la comunidad hacker en lo que respecta a las características críticas de las licencias estándar (la GPL, la licencia BSD, la licencia MIT y la licencia Artística). Estas cláusulas tienen el efecto (aunque no la intención) de hacer que el valor directo de venta sea muy difícil de capturar.

Modelos de Valor de Venta Indirecto

Existen maneras de crear mercados en servicios relacionados con el software que capturan algo semejante al valor de venta indirecto. Hay cinco modelos conocidos y dos potenciales de este tipo (podrían desarrollarse más en el futuro).

Líder en pérdidas / Posicionador de mercado

En este modelo, se usa software open source para crear o mantener una posición de mercado para el software propietario que genere una fuente de ingresos directa. En la variante más común, el software de cliente open source posibilita las ventas de software de servidor, o ingresos asociados a suscripciones o publicidad en un portal.

Netscape Communications perseguía esta estrategia cuando liberó el navegador Mozilla en 1.998. La parte de navegador de su negocio suponía el 13% de los ingresos y seguía descendiendo cuando Microsoft lanzó Internet Explorer (IE). Un marketing intenso de IE (y ciertas prácticas de empaquetamiento que más tarde serían el asunto central de un juicio antimonopolio) redujo rápidamente la cuota de mercado de Netscape, creando la preocupación de que Microsoft pretendía monopolizar el mercado de navegadores y después usar el control de facto de HTML y HTTP para echar a Netscape del mercado de servidores.

Liberando el todavía popular navegador Netscape, Netscape negaba de manera eficaz a Microsoft la posibilidad del monopolio en navegadores. Esperaban que la colaboración open source aceleraría el desarrollo y depuración del navegador, y esperaban que Explorer se viera reducido al papel de segundón, evitando que pudiera definir HTML en exclusiva.

Esta estrategia funcionó. En Noviembre de 1.998 Netscape empezó a recuperar cuota de mercado con respecto a IE. Para cuando Netscape fue comprado por AOL a comienzos de 1.999, la ventaja competitiva de mantener a Mozilla en juego era tan clara que uno de los primeros compromisos públicos de AOL fue continuar soportando el proyecto Mozilla, incluso aunque estaba todavía en estado alfa.

Escarchar dispositivos

Este modelo es para fabricantes de hardware (hardware, en este contexto, incluye cualquier cosa, desde tarjetas Ethernet hasta ordenadores completos). La presión del mercado ha obligado a las compañías de hardware a escribir y mantener software (desde drivers hasta herramientas de configuración, incluso sistemas operativos), pero el software en sí mismo no es un centro de beneficio. Es un sobrecosto; y a menudo uno muy importante.

En esta situación, abrir el código es obvio. No hay una fuente de ingresos que perder, así que no hay inconvenientes. Lo que el fabricante gana es un equipo de desarrollo mucho más grande, respuestas más rápidas y flexibles a las necesidades de los usuarios y mejor fiabilidad mediante la revisión entre pares. Consigue migraciones a otros entornos gratis. Probablemente también gana en mayor lealtad de los usuarios según los técnicos de sus clientes van invirtiendo cada vez más tiempo en el código para mejorarlo según sus necesidades.

Hay un par de objeciones que los fabricantes presentan habitualmente a la liberación de los drivers de hardware. En lugar de mezclar estas objeciones con la discusión de temas más generales aquí, he escrito un epílogo específico sobre este tema.

El efecto de “protección frente al futuro” del open source es particularmente fuerte con respecto al Escarchar dispositivos. Los productos hardware tienen un ciclo de vida de producción y soporte finito; después, los clientes se las tienen que arreglar por sí mismos. Pero si tiene acceso al código del driver y pueden ajustarlo según sus necesidades, es más probable que sean clientes felices que

vuelvan a comprar.

Un ejemplo dramático de la adopción del modelo Escarchar dispositivos fue la decisión de Apple Computer en marzo de 1.999 de liberar “Darwin”, el núcleo de su sistema operativo Mac OS X.

Regala la receta y abre un restaurante

En este modelo, uno libera software para crear una posición de mercado, no para el software cerrado (como el caso del líder en pérdidas / posicionador de mercado) sino para los servicios.

(Solía llamar a esto “Regala la maquinilla, vende hojas de afeitar”, pero la relación no es tan precisa como la analogía de la maquinilla de afeitar implica.)

Este modelo lo usó por primera vez Cygnus Solutions, probablemente el primer negocio open source (1.989). En aquella época, las herramientas GNU proporcionaban un entorno de desarrollo común entre varias máquinas, pero cada herramienta usaba un proceso de configuración diferente y requería un conjunto diferente de parches para correr en cada plataforma. Cygnus domesticó las herramientas GNU y creó el script “configure” para unificar el proceso de construcción (la receta), y después vendió servicios de soporte y binarios empaquetados con su versión de las herramientas GNU (el restaurante). De acuerdo con la GPL, permitían a los clientes usar, distribuir y modificar libremente el software que distribuyeron, pero el contrato de servicios terminaría (o se requeriría pagar una tarifa superior) si había más usuarios de los servicios de soporte de los estipulados en el contrato (no se permite compartir el bufé de ensaladas).

Esto es también lo que hacen Red Hat y otros distribuidores de Linux. Lo que en realidad venden no es el software, los bits en sí mismos, sino el valor añadido al ensamblar y probar un sistema operativo funcional del que se garantiza (aunque sea de manera implícita) que es vendible y que es compatible con otros sistemas operativos de la misma marca. Otros elementos de su proposición de valor incluyen el soporte gratuito para la instalación y la provisión de opciones para contratos de soporte continuado.

El efecto de construcción de mercado del open source puede ser extremadamente poderoso, especialmente para compañías que están inevitablemente en una posición de prestación de servicios para empezar. Un caso reciente muy instructivo es Digital Creations, una empresa de diseño de sitios web fundada en 1.998 que se especializa en sitios con transacciones y bases de datos complejas. Su principal herramienta, la joya de la corona de la propiedad intelectual de la compañía, es un publicador de objetos que ha pasado por diversos nombres y encarnaciones pero ahora se llama Zope.

Cuando la gente de Digital Creations estaba buscando financiación, el socio capitalista que incorporaron evaluó cuidadosamente su nicho de mercado, su personal y sus herramientas. El socio capitalista recomendó a Digital Creations que liberara Zope.

Según los estándares tradicionales de la industria del software, esto parece un movimiento completamente absurdo. La sabiduría convencional de las escuelas de negocios dice que la propiedad intelectual como Zope es la joya de la corona de una compañía, y nunca, bajo ninguna circunstancia, debe cederse. Pero el socio capitalista tenía dos ideas relacionadas.

Una es que el verdadero valor central de Zope es en realidad los cerebros y habilidades de su gente. La otra es que es más probable que Zope genere valor como creador de mercado que como herramienta secreta.

Para ver esto, compare dos escenarios. En el convencional, Zope permanece como el arma secreta de Digital Creations. Establezcamos que es muy efectiva. Como resultado, la firma será capaz de producir con más calidad en menos tiempo, pero *nadie lo sabe*. Será fácil satisfacer a los clientes, pero más difícil construir una base de clientes con la que empezar.

El socio capitalista, en cambio, vio que liberar Zope podría ser una publicidad crítica para el

verdadero activo de Digital Creations: su gente. Esperaba que los clientes que evaluaban Zope considerarían más eficiente contratar a los expertos que desarrollar conocimientos internos en Zope.

Uno de los directivos de Zope ha confirmado públicamente que su estrategia open source ha "abierto muchas puertas en la que no habríamos entrado de otra manera" [sic]. Los clientes potenciales responden sin duda a la lógica de la situación; y Digital Creations está prosperando.

Otro ejemplo actual es e-smith. Esta compañía vende contratos de soporte para software de servidores de Internet que es open source, un Linux personalizado. Uno de los directivos, describiendo la cantidad de descargas gratuitas del software de e-smith, dice "La mayoría de las empresas lo considerarían piratería de software; nosotros lo consideramos marketing gratuito".

Accesorios

En este modelo, usted vende accesorios para el software open source. En la gama baja, tazas y camisetas; en la gama alta, documentación editada y producida profesionalmente.

O'Reilly & Associates Inc., editores de muchos volúmenes excelentes de referencia sobre software open source, es un buen ejemplo de este tipo de compañía. En realidad, O'Reilly contrata y da soporte a muchos hackers reconocidos (como Larry Wall y Brian Behlendorf) como una vía para construir su reputación en el mercado que han elegido.

Libera el futuro, vende el presente

En este modelo, produce software en código binario y fuente con una licencia cerrada, pero que incluye una fecha de caducidad en sus condiciones. Por ejemplo, puede escribir una licencia que permita la libre distribución, prohíba el uso comercial sin licencia, y garantice que el software se acogerá a la licencia GPL un año después de su lanzamiento o si el fabricante cierra.

Bajo este modelo, los clientes pueden estar seguros de que el producto es adaptable a sus necesidades, porque tienen el fuente. El producto está protegido frente al futuro: la licencia garantiza que una comunidad open source puede adoptar el producto si la compañía original muere.

Debido a que el precio de venta y el volumen están basados en las expectativas de los clientes, la compañía original debería disfrutar de mayores ingresos por sus productos que si los lanzara con una licencia completamente cerrada. Más aún, como el código antiguo tiene licencia GPL, conseguirá una revisión seria, corrección de errores, y pequeños añadidos de funcionalidad, que reducirán parte de la carga del 75% de mantenimiento del fabricante.

Este modelo lo ha seguido con éxito Aladdin Enterprises, fabricante del popular programa Ghostscript (un intérprete de PostScript que puede traducir al lenguaje nativo de muchas impresoras).

El principal inconveniente de este modelo es que las condiciones de cierre de licencia tienden a inhibir la revisión y la participación en las fases tempranas del ciclo del producto, precisamente cuando se necesitan más.

Libera el software, vende la marca

Este es un modelo de negocio teórico. Usted libera una tecnología de software, retiene un juego de pruebas o un conjunto de criterios de compatibilidad, y luego vende a los usuarios una marca que certifica que su implementación de la tecnología es compatible con todos los demás que llevan la marca.

(Así es como Sun Microsystems debería manejar Java y Jini).

Actualización: en Julio de 2.000, Sun anunció que liberaría Star Office, y que vendería el uso de la marca Star Office a desarrollos del código base que pasan el juego de pruebas de Sun.

Libera el software, vende el contenido

Este es otro modelo de negocio hipotético. Imagine algo como un servicio de suscripción a datos de bolsa. El valor no está ni en el software cliente ni en el servidor, sino en proporcionar información objetivamente fiable. Así que libera todo el software y vende suscripciones al contenido. Según los hackers vayan migrando el cliente a nuevas plataformas y lo mejoren de cualquier manera, su mercado se expande automáticamente.

(Esta es la razón por la que AOL debería liberar su software cliente).

Cuándo ser abierto, cuándo ser cerrado

Habiendo revisado modelos de negocio que soportan el desarrollo de software open source, podemos aproximarnos a la pregunta general de cuándo tiene sentido desde el punto de vista económico ser abierto y cuándo ser cerrado. Primero, debemos aclarar cuáles son los beneficios de cada estrategia.

¿Cuáles son los beneficios?

La aproximación de cerrar el fuente le permite obtener rentas de sus bits secretos; por otro lado, cierra la posibilidad de una revisión entre pares verdaderamente independiente. La aproximación open source establece condiciones para la revisión entre pares, pero no consigue rentas de sus bits secretos.

El beneficio de tener bits secretos se entiende bien; tradicionalmente, los modelos de negocio de software se han construido alrededor de ellos. Hasta recientemente, el beneficio de la revisión entre pares independiente no se entendía bien. El sistema operativo Linux, sin embargo, nos trae una lección que deberíamos haber aprendido probablemente hace años de la historia del software central de Internet y otras ramas de la ingeniería: que la revisión entre pares del open source es el único método escalable para conseguir alta fiabilidad y calidad.

En un mercado en competencia, por tanto, los clientes que buscan alta fiabilidad y calidad recompensarán a los productores de software que opten por el open source y descubran cómo mantener una corriente de ingresos en los mercados de servicios, valor añadido y accesorios asociados con el software. Este fenómeno es lo que está detrás del sorprendente éxito de Linux, que surgió de la nada en 1.996 para convertirse en el segundo sistema operativo más popular en el mercado de servidores para empresas a mediados de 2.000 (y algunos estudios muestran que sobrepasó la cuota de mercado de Microsoft a finales de 2.000). A comienzos de 1.999 IDC proyectó que Linux crecería más rápido que todos los demás sistemas operativos juntos hasta 2.003: esta proyección se ha mostrado cierta hasta la fecha.

Un beneficio casi tan importante como del open source es su utilidad como manera de propagar estándares abiertos y construir mercados a su alrededor. El dramático crecimiento de Internet debe mucho al hecho de que nadie es dueño de TCP/IP; nadie tiene un cerrojo propietario en los protocolos centrales de Internet.

Los efectos de red que están detrás del éxito de TCP/IP y Linux están bastante claros y se reducen al final a cuestiones de confianza, y socios potencialmente simétricos que comparten una infraestructura confiarán más en ella si pueden ver cómo trabaja hasta el fondo, y preferirán una infraestructura en la que todas las partes tengan derechos simétricos a una en la que una parte está en una posición privilegiada para extraer beneficios o ejercer el control.

No es, sin embargo, necesario en realidad suponer efectos de red para que las cuestiones de simetría sean importantes para los consumidores de software. Ningún consumidor de software elegirá

racionalmente cerrarse a sí mismo en un monopolio controlado por un suministrador haciéndose dependiente de un código cerrado si está disponible alguna alternativa open source de calidad aceptable. Este argumento gana fuerza según el software se hace más crítico para el negocio del consumidor de software: cuanto más vital es, menos puede tolerar el cliente que esté controlado por un tercero.

Hay una cuestión tangencial a esta. Los economistas saben que, en general, la información asimétrica hace que los mercados trabajen con ineficacias. Los bienes de mayor calidad quedan fuera cuando es más lucrativo obtener beneficios de la información privilegiada que invertir en la producción de mejores productos. En general, y no sólo en el software, el secreto es enemigo de la calidad.

Por último, un beneficio importante para el cliente del software open source que tiene que ver con la cuestión de la confianza es que está protegido frente al futuro. Si el fuente está abierto, el cliente tiene algún recurso si el fabricante quiebra. Esto puede ser particularmente importante para el "escarchado de dispositivos", ya que el hardware tiende a tener ciclos de vida cortos, pero el efecto es más general y se traduce en incremento de valor para todo tipo de software open source.

¿Cómo interactúan?

Cuando el beneficio de los bits secretos es mayor que el retorno del open source, tiene sentido desde el punto de vista económico ser cerrado, Cuando el retorno del open source es mayor que el beneficio de los bits secretos, tiene sentido hacer open source.

Por sí misma, esta es una observación trivial. Se hace no trivial cuando observamos que el beneficio del open source es más difícil de medir y predecir que el beneficio de los bits secretos; y dicho esto, el beneficio es infravalorado mucho más a menudo que sobrevalorado. Sin duda, hasta que el mundo corporativo comenzó a repensar sus premisas siguiendo la liberación de los fuentes de Mozilla a comienzos de 1.998, se suponía, equivocada pero comúnmente, que el beneficio del open source era cero.

¿Así que cómo podemos evaluar el beneficio del open source? Es una pregunta difícil en general, pero podemos aproximarlo como haríamos con cualquier otro problema predictivo. Podemos empezar por los casos observados en los que la aproximación open source ha tenido éxito o ha fracasado. Podemos intentar generalizarlos a un modelo que proporcione al menos una indicación cualitativa para los contextos en los que el open source es una ganancia neta para el inversor o el negocio que intentar maximizar los beneficios. Después podemos volver a los datos e intentar refinar el modelo.

Del análisis presentado en *La Catedral y el Bazar*, podemos esperar que el open source tenga un beneficio alto cuando (a) la fiabilidad/estabilidad/escalabilidad sean críticas, y (b) la corrección del diseño y la implementación no sean fácilmente verificables por otros medios distintos a la revisión independiente entre pares. (El segundo criterio lo cumplen en la práctica la mayoría de los programas no triviales).

Un deseo racional del consumidor de evitar verse atrapado en el monopolio de un proveedor incrementará su interés en el open source (y, por tanto, el valor competitivo en el mercado de sus proveedores) según el software se hace más crítico para dicho cliente. Así, otro criterio (c) empuja hacia el open source cuando el software es un bien crítico para el negocio (como, por ejemplo, en muchos sistemas de información de gestión corporativos).

Para el área de aplicaciones, observamos antes que la infraestructura open source crea efectos de confianza y simetría que, con el tiempo, tenderán a atraer más clientes y se impondrá a la infraestructura cerrada; y a menudo es mejor tener una porción más pequeña de un mercado en rápida expansión que una mayor de uno cerrado y estancado. Igualmente, para el software de infraestructura, la apuesta del open source por la ubicuidad es muy probable que tenga un mayor

retorno a largo plazo que la apuesta del software cerrado por la renta de la propiedad intelectual.

De hecho, la capacidad de los clientes potenciales para razonar acerca de las consecuencias futuras de las estrategias de los fabricantes y su resistencia a aceptar un monopolio de fabricantes implica una restricción más fuerte: sin tener todavía un poder abrumador en el mercado, puede elegir entre una apuesta por la ubicuidad del open source o una apuesta por los ingresos directos a partir del código cerrado; pero no puede apostar por ambas. (Analogías a este principio aparecen en todas partes; por ejemplo, en los mercados de electrónica donde los clientes a menudo evitan comprar diseños de una única fuente). El caso puede expresarse menos negativamente: donde los efectos de red (externalidades de red positivas) dominan, el open source es probablemente la opción correcta.

Podemos resumir esta lógica haciendo notar que el open source parece tener más éxito para generar mayores ingresos que el código cerrado en software que (d) establece o habilita una infraestructura de computación y comunicaciones común.

Por último, podemos observar que los proveedores de servicios únicos o altamente diferenciados tienen más incentivos para temer la copia de sus métodos por sus competidores que los proveedores de servicios para los que los algoritmos críticos y las bases de conocimiento están bien entendidas. Según esto, es más probable que el open source domine cuando (e) los métodos clave (o equivalentes funcionales) son parte de un conocimiento de ingeniería común.

El software nuclear de Internet, Apache, y la implementación de Linux del API estándar de Unix son ejemplos que cumplen los cincos criterios. El camino hacia el open source en la evolución de estos mercados está bien ilustrado por la convergencia de las redes de datos en el TCP/IP a mediado de los 90, después de quince años de intentos fallidos de construir imperios con protocolos cerrados como DECNET, XNS, IPX y similares.

Por otro lado, el open source parece tener menos sentido para compañías que tienen la posesión en exclusiva de una tecnología de software generadora de valor (que cumple estrechamente el criterio (e)) que es (a) relativamente insensible a fallos, que puede (b) ser fácilmente verificada por medios diferentes a la revisión independiente entre pares, que no es (c) crítica para el negocio, y cuyo valor no se vería sustancialmente incrementado por (d) efectos de red o ubicuidad.

Como ejemplo de este caso extremo, a comienzos de 1.999 me preguntaron “¿Deberíamos liberar nuestro código?” desde una compañía que escribe software para calcular patrones de corte para serrerías que quieren extraer el máximo metraje de planchas a partir de los troncos. Mi conclusión fue “No”. El único criterio que se acerca al cumplimiento es el (c); pero en un apuro, un operario con experiencia podría generar patrones de corte a mano.

Observe que mi respuesta podría haber sido muy diferente si el calculador de patrones de corte hubiera sido escrito por un fabricante de maquinaria para serrerías. En ese caso, abrir el código podría haber incrementado el valor de la maquinaria asociada que venden. Observe también que si existiera ya algún calculador de patrones de corte open source (quizás el escrito por el fabricante de herramientas para serrerías) el producto cerrado tendría problemas para competir con él; no tanto por razones de precio, sino porque los clientes percibirían en el software abierto ventajas en cuanto a capacidad de personalización y otros rasgos.

Un punto importante es que dónde un producto o tecnología concretos se asiente en estas escalas puede cambiar con el tiempo, como veremos en el siguiente caso de estudio.

En resumen, los siguientes discriminadores empujan hacia el open source:

(a) La fiabilidad/estabilidad/escalabilidad son críticas.

(b) La corrección del diseño y su implementación no puede revisarse fácilmente por otros medios diferentes de la revisión independiente entre pares.

(c) El software es crítico para el control del negocio del usuario.

(d) El software establece o habilita una infraestructura de computación y comunicaciones

común.

(e) Los métodos clave (o sus equivalentes funcionales) son parte de un conocimiento de ingeniería común.

Doom: Un caso de estudio

La historia del juego de éxito Doom de id software ilustra vías en las que la presión del mercado y la evolución del producto pueden cambiar de manera crítica las magnitudes del beneficio para el software abierto o cerrado.

Cuando Doom se lanzó por primera vez a finales de 1.993, su animación de primera persona en tiempo real lo hizo único (la antítesis del criterio (e)). No sólo era impresionante el impacto visual de sus técnicas (sobrepasando el mundo de animación plana de su predecesor Wolfenstein 3D), sino que durante meses nadie pudo averiguar cómo se había conseguido en los limitados procesadores de la época. Estos bits secretos suponían un valor muy importante. Además, el beneficio potencial de abrir el fuente era bajo. Como juego individual, el software (a) tenía costes bajos de fallo, (b) no era tremendamente difícil de verificar, (c) no era crítico para el negocio de ningún cliente, (d) no se beneficiaba de efectos de red. Era económicamente racional para Doom ser cerrado.

Sin embargo, el Mercado construido alrededor de Doom no permaneció quieto. Los competidores potenciales inventaron equivalentes funcionales de sus técnicas de animación, y otros juegos de "disparo en primera persona" como Duke Nukem empezaron a aparecer. Según quitaban cuota de mercado a Doom estos juegos, el valor de la renta de los bits secretos disminuía.

Por otro lado, los esfuerzos para expandir esa cuota trajeron nuevos desafíos técnicos: mejor fiabilidad, más funciones, mayor base de usuarios y múltiples plataformas. Con la llegada de juegos multijugador y servicios de juegos de Doom, el mercado empezó a mostrar efectos de red sustanciales. Todo esto requería horas de programador que id hubiera preferido invertir en el siguiente juego.

Desde el momento en que el juego se liberó por primera vez, id había visto con benevolencia la publicación de especificaciones técnicas que ayudaran a la gente a crear objetos de datos para el juego, y ocasionalmente cooperaba directamente con los hackers respondiendo a preguntas concretas o publicando un documento propio de especificaciones, También fomentaban la distribución por Internet de nuevos datos para Doom.

Las tendencias técnicas y de mercado elevaron el beneficio de la liberación del código; la apertura de especificaciones y la potenciación de los añadidos de terceros incrementaron el valor percibido del juego y crearon un mercado secundario a explotar. En algún momento las curvas de beneficios se cruzaron y se hizo económicamente racional para id pasar a hacer dinero en ese mercado secundario (con productos como antologías de escenarios para el juego) y después abrir el código fuente de Doom. Algún tiempo después de este punto, ocurrió. El código fuente completo de Doom se liberó a finales de 1.997.

Sabiendo cuándo liberar

Doom es un caso de estudio interesante porque no es ni un sistema operativo ni un software de comunicaciones o redes; por tanto está lejos de los ejemplos habituales y obvios de éxito en open source. Sin duda, el ciclo de vida de Doom, junto a su punto de cruce, puede convertirse en el modelo para el software de aplicación en la ecología del código actual; una en la que las comunicaciones y la computación distribuida crea tanto problemas serios de robustez/fiabilidad/escalabilidad sólo resolubles mediante la revisión entre pares, y a menudo cruza fronteras entre entornos técnicos y entre agentes competidores (con todas las cuestiones de confianza y simetría que esto implica).

Doom evolucionó del juego solitario al juego en red. Cada vez más, el efecto de red es la computación. Tendencias similares son visibles incluso en las aplicaciones de negocio más pesadas, como los sistemas ERP, según los negocios se relacionan más intensamente con suministradores y clientes; y, por supuesto, están implícitas en toda la arquitectura del World Wide Web. Se sigue que prácticamente en todas partes, el beneficio del open source está incrementándose continuamente.

Si la tendencia actual continúa, el desafío central de la tecnología del software y la gestión de productos en el próximo siglo será conocer cuándo liberar: cuándo permitir al código cerrado pasar a la infraestructura open source para explotar el efecto de la revisión entre pares y capturar mayores ingresos en servicios y otros mercados secundarios.

Hay incentivos obvios para no errar en el punto de cruce demasiado lejos en cualquier dirección. Más allá, hay un riesgo de oportunidad serio al esperar demasiado: podría quedar desplazado por un competidor que pasara a ser open source en el mismo nicho de mercado.

La razón de que esto sea un asunto serio es que tanto el pool de usuarios como el pool de talento disponible para ser reclutado para la cooperación open source en cualquier categoría de producto dada es limitado, y el reclutamiento tiende a ser adhesivo. Si dos productores son el primero y el segundo en liberar código de funcionalidad prácticamente equivalente, el primero es probable que el primero atraiga la mayoría de usuarios y la mayoría de los desarrolladores más motivados; el segundo tendrá que coger las sobras. El reclutamiento tiende a ganar adhesiones, según los usuarios ganan familiaridad y los desarrolladores invierten tiempo en el código.

Open Source como arma estratégica

A veces, liberar el código puede ser eficaz no sólo como vía para incrementar el mercado sino como maniobra estratégica contra la competencia de una compañía. Puede ser fructífero revisar algunas de las tácticas de negocio descritas antes desde este ángulo; no directamente como generadores de ingresos sino como maneras de introducirse y reformar mercados.

Compartir costes como arma competitiva

Antes consideramos Apache como ejemplo de mejor y más barato desarrollo de infraestructuras mediante la compartición de costos en un proyecto open source. Para los fabricantes de software y sistemas que compiten con Microsoft y su servidor web IIS, el proyecto Apache es también un arma competitiva. Sería difícil, tal vez imposible, para cualquier otro proveedor de servidores web desbancar completamente las ventajas del enorme equipo de guerra de Microsoft y el poder de su monopolio en el mercado de desktop. Pero Apache permite a cada empresa que participa en el proyecto ofrecer un servidor web que es técnicamente superior a IIS y reafirma a los clientes con una cuota de mercado mayoritaria; a un coste muy inferior. Esto mejora la posición de mercado y el coste de producción para los productos de comercio electrónico de valor añadido (como WebSphere de IBM).

Esto es generalizable. La infraestructura abierta y compartida proporciona a sus participantes ventajas competitivas. Una es menor coste por participante para producir productos y servicios vendibles. Otra es una posición de mercado que asegura a los clientes que es mucho menos probable que quede atados a una tecnología huérfana como resultado de una cambio de estrategia o táctica de un proveedor.

Reajustar la competencia

Cuando el desarrollo del sistema open source de ventanas X windows fue financiado por DEC en los 80, su objetivo explícito era “reajustar la competencia”. En aquella época había varios entornos

gráficos para Unix que competían, incluyendo especialmente el sistema NeWS de Sun Microsystems. Los estrategas de DEC creyeron (probablemente de manera correcta) que si Sun era capaz de establecer un estándar de gráficos propietario podría cerrar el floreciente mercado de estaciones de trabajo Unix. Financiando X y cediéndole ingenieros, y aliándose con muchos fabricantes más pequeños para establecer X como estándar de facto, DEC fue capaz de neutralizar las ventajas que tenían Sun y otros competidores con mayor experiencia interna en gráficos. Esto desplazó el foco de la competencia en el mercado de estaciones de trabajo hacia el hardware, donde DEC era históricamente fuerte.

Esto también es generalizable. El Open Source es atractivo para los clientes inteligentes, y para aliados potenciales que no son suficientemente grandes como para financiar un desarrollo competitivo por sí mismos. Un proyecto open source, lanzado en el momento adecuado, puede ser mejor que simplemente competir con éxito contra alternativas cerradas; en realidad puede evitar que consigan agarre en el mercado, reajustando la competencia y redirigiéndola desde un área donde la compañía que lo inicia es débil a una donde es fuerte.

Aumentar el estanque

Red Hat Software financió el desarrollo del sistema de paquetes RPM para proporcionar al mundo Linux un instalador de paquetes binarios estándar. Haciendo esto, apostaron por que el incremento de confianza que tal instalador estándar proporcionaría a los clientes potenciales valdría más en ingresos futuros que el coste de desarrollo del software o los ingresos potencialmente perdidos frente a los competidores que también serían capaces de usarlo.

A veces la mejor manera de ser una rana más grande es hacer que la charca crezca más rápido. Esto, por supuesto, es la razón económica de que las empresas de tecnología hayan participado en estándares públicos: y es útil pensar en el software open source como en un estándar ejecutable. Además de ser un excelente creador de mercados, esta estrategia puede ser un arma competitiva directa cuando una compañía pequeña la usa para desplazar la masa y el poder en el mercado de una compañía mucho más grande que no participa en la alianza del estándar. En el caso de Red Hat, el gran competidor obvio y reconocido es Microsoft; la estandarización de RPM en la mayoría de distribuciones Linux aportó mucho para la neutralización de las ventajas que Microsoft había establecido previamente en la sencillez de la administración de sistemas en sus máquinas Windows.

Evitando un chokehold

Al explicar antes el modelo de negocio líder en pérdidas / posicionador de mercado, describí cómo la apertura del código del navegador Mozilla por parte de Netscape fue una maniobra (exitosa) dirigida a evitar que Microsoft bloqueara HTML y el protocolo HTTP.

A menudo, es más importante evitar que su competencia consiga un chokehold en una tecnología particular que controlar la tecnología usted mismo. Abriendo el código, incrementa enormemente el tamaño potencial de su coalición bloqueante.

Open Source y Riesgo Estratégico de Negocio

Al final, las razones por las que el open source parece destinado a convertirse en una práctica extendida tienen más que ver con la demanda de los clientes y las presiones del mercado que con eficiencias del lado de la oferta para los proveedores. Ya he discutido, desde el punto de vista del proveedor, los efectos de la demanda de los clientes de fiabilidad y de infraestructuras sin un único jugador dominante, y cómo éstos han jugado históricamente en la evolución de las redes. Hay más que decir, sin embargo, sobre el comportamiento de los clientes en un mercado en el que el open source es un factor.

Colóquese por un momento en la situación de un Director de Tecnología en una compañía del Fortune 500 que piensa si construir o actualizar una de las infraestructuras IT de su firma. Quizá necesita elegir un sistema operativo de red para desplegarlo en toda la empresa; quizá su preocupación implica un servicio web y de comercio electrónico 24x7; quizá su negocio dependa de que sea capaz de soportar transacciones de bases de datos de alto volumen y alta fiabilidad.

Suponga que sigue el camino tradicional del código cerrado. Si lo hace, coloca a su firma a la merced del monopolio de un suministrador: porque por definición, hay sólo un lugar al que ir para obtener soporte, corrección de errores y mejoras. Si el suministrador no responde, no tendrá ningún recurso eficaz porque se encontrará atrapado por su inversión inicial y los costes de formación. Su suministrador lo sabe. Bajo estas circunstancias, ¿supone que el software cambiará para ajustarse a *sus* necesidades y *su* plan de negocios... o las necesidades de *su proveedor* y el plan de negocios de *su proveedor*?

La brutal verdad es esta: cuando sus procesos de negocio clave se ejecuten mediante bloques opacos de bits que ni siquiera puede ver (ni mucho menos modificar) *habrá perdido el control de su negocio*. Necesitará a su proveedor más que su proveedor le necesita a usted; y pagará, y pagará, y pagará otra vez por este desequilibrio de poder. Pagará en precios más altos, pagará en oportunidades perdidas y pagará en trampas que se hacen cada vez peores según el proveedor (que ha refinado su juego en un montón de víctimas anteriores) aprieta el lazo.

Compare esto con la opción open source. Si sigue esta ruta, *usted tiene el código fuente*, y nadie puede quitárselo. En lugar del monopolio de un proveedor con un chokehold en su negocio, ahora tiene varias compañías de servicios que ofertan por su negocio; y no sólo consigue que compitan entre ellas, sino que tiene la opción de construir su propia organización de soporte interna si eso parece menos caro que contratar a terceros. El mercado trabaja para *usted*.

La lógica es convincente; depender de código cerrado es una riesgo estratégico inaceptable para el negocio. Tanto que creo que no pasará mucho hasta que la compra de código cerrado a un único proveedor cuando haya una alternativa open source disponible se verá como una irresponsabilidad fiduciaria real, y una base para un litigio por parte de los accionistas.

La Ecología de Negocios del Open Source

La comunidad open source se ha organizado a sí misma de una manera que tiende a amplificar los efectos de productividad del open source. En el mundo Linux, en particular, es un hecho económicamente significativo que hay varios distribuidores de Linux que forman una capa distinta de los desarrolladores.

Los desarrolladores escriben código, y hacen el código disponible por Internet. Cada distribuidor selecciona algún subconjunto del código disponible, lo integra, lo empaqueta, pone su marca y lo vende a los clientes. Los usuarios eligen entre distribuciones, y pueden complementar una distribución descargando código directamente del sitio del desarrollador.

El efecto de esta separación entre capas es crear un mercado interno de mejoras muy fluido. Los desarrolladores compiten entre sí por la atención de los distribuidores y los usuarios sobre la calidad de su software. Los distribuidores compiten por los dólares de los usuarios sobre lo adecuado de sus políticas de selección, y sobre el valor que pueden añadir al software.

Un efecto de primer orden de la estructura interna de este Mercado es que ningún nodo de la red es indispensable. Los desarrolladores pueden abandonar; incluso si su parte del código base no es recogida directamente por algún otro desarrollador, la competencia por la atención tenderá a generar rápidamente alternativas funcionales. Los distribuidores pueden fallar sin dañar o comprometer el código base común. La ecología como un todo tiene una respuesta más rápida a las demandas del mercado, y más capacidad para resistir shocks y regenerarse, que la que cualquier fabricante monolítico de un sistema operativo cerrado puede posiblemente conseguir.

Otro efecto importante es reducir el sobrecosto e incrementar la eficiencia mediante la especialización. Los desarrolladores no experimentan las presiones que habitualmente comprometen los proyectos cerrados convencionales y los convierten en pozos de alquitrán; no hay listas de funciones inútiles y molestas de Marketing, ningún gestor obliga a usar lenguajes o entornos de desarrollo inadecuados y obsoletos, no hay requerimientos para reinventar la rueda de una manera nueva e incompatible en nombre de la diferenciación del producto o de la protección de la propiedad intelectual y (lo más importante) *no hay fechas límite*. No se corre para sacar una versión 1.0 antes de que esté bien hecha. De Marco y Lister observaron en su discusión del estilo de dirección “despiértame cuando esté acabado” en “Peopleware: Proyectos y equipos productivos” que esto normalmente conduce no sólo a más calidad sino a la entrega más rápida de un resultado operativo.

Los distribuidores, por otro lado, consiguen especializarse en las cosas que los distribuidores pueden hacer más eficazmente. Liberados de la necesidad de financiar desarrollos de software masivos y continuados sólo para permanecer competitivos, pueden concentrarse en la integración de sistemas, el empaquetamiento, el aseguramiento de la calidad y el servicio.

Tanto los distribuidores como los desarrolladores se mantienen honestos gracias al feedback constante y la monitorización de los usuarios, que es una parte integral del método open source.

Lidiando con el Éxito

La Tragedia de los Comunes puede no ser aplicable al desarrollo open source tal como sucede hoy, pero eso no significa que no haya razones para considerar si el momento actual de la comunidad open source es sostenible. ¿Dejarán de cooperar los jugadores clave cuando las apuestas se hagan mayores?

Hay varios niveles en los que se puede plantear esta pregunta. Nuestra historia “Comedia de los comunes” está basada en el argumento de que el valor de las contribuciones individuales al open source es difícil de convertir a dinero. Pero este argumento tiene mucha menos fuerza para las empresas (como, por ejemplo, los distribuidores de Linux) que ya tiene una fuente de ingresos asociada al open source. Su contribución ya se está traduciendo en dinero cada día. ¿Es estable su papel colaborativo actual?

Examinar esta cuestión nos conducirá a algunas perspectivas interesantes sobre la economía del software open source en el mundo real de nuestro tiempo; y sobre lo que un verdadero paradigma de industria de servicios implica para la industria de software en el futuro.

En el nivel práctico, aplicado a la comunidad open source tal como existe hoy, esta cuestión normalmente se plantea de dos maneras diferentes. Una: ¿Se fragmentará Linux? Dos: por el contrario, ¿desarrollará Linux un jugador dominante, casi monopolista?

La analogía histórica a la que mucha gente acude cuando se plantea si Linux se fragmentará es el comportamiento de los fabricantes de Unix propietarios en los 80. A pesar de la incesante charla sobre estándares abiertos, a pesar de las numerosas alianzas y consorcios y acuerdos, los Unix propietarios de separaron. El deseo de los vendedores de diferenciar sus productos añadiendo y modificando funciones del sistema operativo se demostró más fuerte que su interés en incrementar el tamaño total del mercado Unix manteniendo la compatibilidad (y como consecuencia bajando las barreras de entrada para los desarrolladores independientes de software y el coste total de propiedad para los clientes).

Es bastante improbable que esto suceda con Linux, por la simple razón de que todos los distribuidores están limitados a trabajar partiendo de una base común de código abierto. No es posible en la práctica para cualquiera de ellos mantener la diferenciación, porque las licencias bajo las que ha sido desarrollado el código de Linux requieren que compartan el código con el resto. En el momento que cualquier distribuidor desarrolla una función, todos los competidores son libres de clonarla.

Como todas las partes entienden esto, nadie piensa siquiera en hacer el tipo de maniobras que fragmentaron los Unix propietarios. Al revés, los distribuidores de Linux están obligados a competir de modos que benefician en la práctica al cliente y al mercado en general. Esto es, deben competir en servicio, en soporte, y su diseño se apoya en qué interfaces llevan a mayor sencillez de instalación y uso.

La base de código común también cierra la posibilidad de monopolización. Cuando la gente de Linux se preocupa por esto, el nombre que normalmente se murmura es “Red Hat”, que es el mayor y más exitoso de los distribuidores (con una cuota de mercado estimada de alrededor del 90% en los Estados Unidos). Pero es destacable que unos días después del anuncio en Mayo de 1.999 del lanzamiento de la esperada versión 6.0 de Red Hat, antes incluso de que los CD-ROM de Red Hat llegaran a distribuirse en cantidades significativas, imágenes de CD-ROM de la versión construidas a partir del propio servidor FTP de Red Hat estaban siendo anunciadas por un editor de libros y varios distribuidores de CD-ROM a precios incluso menores que los precios de lista de Red Hat.

A la propia Red Hat ni siquiera se le movió un pelo por esto, porque sus fundadores entienden muy claramente que ellos si poseen ni pueden poseer los bits de su producto; las normas sociales de la comunidad Linux lo prohíben. En una reinterpretación de la famosa observación de John Gilmore de que Internet interpreta la censura como daño, y la evita con un rodeo, se ha dicho acertadamente que la comunidad hacker responsable de Linux interpreta los intentos de control como daño y los evita con un rodeo. Para Red Hat, haber protestado contra las copias de su producto más novedoso habría comprometido seriamente su capacidad para obtener la cooperación futura de su comunidad de desarrolladores.

Quizá más importante en el momento actual, las licencias de software que expresan estas normas de la comunidad en una forma legal prohíben activamente a Red Hat monopolizar los fuentes del código en el que su producto está basado. La única cosa que pueden vender es una relación de marca/servicio/soporte con gente que desea libremente pagar por eso. Este no es un contexto en el que la posibilidad de un monopolio predador suponga una gran amenaza.

I+D abierta y la reinvención del patronazgo

Hay otro aspecto en el que la infusión de dinero real en el mundo open source lo está cambiando. Las estrellas de la comunidad están encontrándose cada vez más que pueden conseguir ser pagadas por lo que hacen, en lugar de seguir al open source como un hobby pagado por otro trabajo de día. Empresas como Red Hat, O'Reilly & Associates, y VA Linux Systems están construyendo el equivalente a brazos de investigación semi-independientes con charters para contratar y mantienen establos de talento open source.

Esto tiene sentido desde el punto de vista económico sólo si el coste per cápita del mantenimiento de este laboratorio puede pagarse fácilmente con las ganancias esperadas que se conseguirán incrementando más rápidamente el mercado de la compañía. O'Reilly puede permitirse pagar a los líderes de Perl y Apache para que hagan sus cosas porque espera que sus esfuerzos le permitirán vender más libros sobre Perl y Apache y llevar a más gente a sus conferencias. VA Linux Systems puede financiar su laboratorio porque mejorar Linux incrementa el valor de uso de las estaciones de trabajo y servidores que vende. Y Red Hat financia los Laboratorios de Desarrollo Avanzado Red Hat para incrementar el valor de su oferta Linux y atraer a más clientes.

Para los estrategas de los sectores más tradicionales de la industria del software, formados en culturas que consideran a las patentes o la propiedad intelectual protegida-por-secretos-de-negocio como las joyas de la corona de una organización, este comportamiento puede (a pesar de sus efectos en el incremento del mercado) parecer inexplicable. ¿Por qué financiar una investigación de la que cada uno de sus competidores (por definición) es libre de apropiarse sin coste?

Parecen existir dos rezones de control. Una es que mientras estas compañías sigan siendo jugadores dominantes en sus nichos de mercado, pueden esperar capturar la parte del león de los beneficios de la investigación y desarrollo open source. Usar el I+D para comprar futuros beneficios no es una idea nueva; lo que es interesante es que el cálculo implícito de que las ganancias futuras esperadas son suficientemente grandes como para que estas compañías puedan tolerar sin problemas a los aprovechados con tal de conseguir el efecto de revisión entre pares.

Aunque este análisis obvio del valor futuro esperado es necesario en un mundo de capitalistas duros que mantienen sus ojos en el retorno de la inversión, no es en realidad la manera más interesante de explicar la contratación de estrellas porque las propias empresas aportan una más difusa. Ellas le dirán, si les pregunta, que están haciendo simplemente lo correcto para la comunidad de la que provienen. Su humilde autor está suficientemente bien relacionado con directivos de las tres firmas antes mencionadas como para testificar que estas afirmaciones no pueden ser consideradas bobadas. Sin duda, yo fui personalmente reclutado en el board de VA Linux Systems a finales de 1.998 explícitamente para que estuviera disponible para aconsejarles sobre "lo adecuado", y he encontrado que han estado muy dispuestos a escucharme cuando lo he hecho.

Un economista está autorizado para preguntar qué beneficio está implicado aquí. Si aceptamos que la charla sobre hacer lo adecuado no es una pose vacía, deberíamos preguntar a continuación que a qué interés de la compañía sirve "hacer lo adecuado". La respuesta no es, en sí misma, ni sorprendente ni difícil de verificar si planteamos las preguntas adecuadas. Como con el comportamiento superficialmente altruista de otras industrias, lo que estas compañías creen que en realidad que están comprando es renombre.

Trabajar para ganar renombre, y valorarlo como un activo que predice futuras ganancias en el Mercado, tampoco es ninguna novedad. Lo que es interesante es la valoración extremadamente alta que el comportamiento de estas empresas sugiere que asignan a este renombre. Están dispuestas a contratar talento caro para proyectos que no son generadores directos de ingresos incluso durante las fases más hambrientas de capital de la carrera hacia la IPO. Y, al menos hasta aquí, el mercado ha premiado con generosidad este comportamiento.

Los propios directivos de estas empresas son bastante claros sobre las razones de que el renombre sea especialmente valorable para ellos. Se apoyan fuertemente en voluntarios entre su base de clientes para el desarrollo de productos y como fuerza de marketing informal. Su relación con su base de clientes es íntima, y a menudo se apoya en lazos de confianza personales entre individuos internos y externos a la firma. No se limitan a usar la comunidad hacker; se identifican con ella.

Estas observaciones refuerzan una lección que aprendimos antes de una línea de razonamiento distinta. La relación íntima entre Red Hat/VA/O'Reilly y sus clientes/desarrolladores no es la típica de las empresas fabricantes. En lugar de eso, lleva a un extremo interesante los patrones característicos de las industrias de servicios altamente profesionalizadas y de conocimiento intenso. Observando desde el exterior de la industria de la tecnología, podemos ver estos patrones (por ejemplo) en firmas de abogados, prácticas médicas y universidades.

Podemos observar, de hecho, que las empresas de open source contratan hackers "estrella" por las mismas razones que las universidades contratan académicos "estrella". En ambos casos, la práctica es parecida en su mecanismo y en su efecto en el sistema de patronazgo aristocrático que financió la mayor parte de las bellas artes hasta después de la Revolución Industrial; una semejanza de la que algunas partes son plenamente conscientes.

Llegando allí desde aquí

Los mecanismos del mercado para financiar (¡y obtener beneficio!) del desarrollo open source están evolucionando todavía rápidamente. Los modelos de negocio que hemos revisado en este ensayo no serán probablemente los últimos en ser inventados. Los inversores están todavía pensando en las consecuencias de reinventar la industria del software como una con un foco explícito en el servicio mas que en la propiedad intelectual cerrada, y lo seguirán haciendo durante algún tiempo.

Esta revolución conceptual tendrá algún coste en pérdidas de beneficio para la gente que invierta en el valor de venta del 5% de la industria; históricamente, los negocios de servicio no son tan lucrativos como los negocios de fabricación (aunque como cualquier médico o abogado puede decirle, el retorno para los profesionales es a menudo mayor). Cualquier pérdida de beneficio, sin embargo, será más que compensada por los beneficios en el lado del coste, según los consumidores de software alcancen enormes ahorros y eficiencias con los productos open source. (Hay un paralelismo aquí con los efectos que el desplazamiento de la red tradicional de voz y teléfono por Internet está teniendo en todas partes).

La promesa de estos ahorros y eficiencias está creando una oportunidad de mercado que los emprendedores y sociedades de capital riesgo se están moviendo para explotar. Cuando el primer borrador de este ensayo estaba en preparación, la firma de capital riesgo más prestigiosa de Silicon Valley tomó una participación importante en la primera compañía emergente que se especializó en soporte técnico 24x7 de Linux (Linuxcare). En agosto de 1.999 la IPO de Red Hat fue (a pesar de un trasfondo de caída en acciones de Internet y tecnología) un gran éxito. Se espera que varias IPOs de empresas relacionadas con Linux y open source se lancen antes del final de 1.999; y que sean también bastante exitosas. (Actualización del año 2.000: ¡lo fueron!)

Otro desarrollo muy interesante es el comienzo de intentos sistemáticos de crear mercados de tareas en los proyectos de desarrollo open source. SourceXchange y CoSource representan maneras ligeramente diferentes de intentar aplicar un modelo de subasta inversa para financiar el desarrollo open source.

Las tendencias generales son claras. Mencionamos antes la proyección de IDC de que Linux crecerá más rápido que todos los otros sistemas operativos juntos hasta 2.003. Apache tiene una cuota del mercado del 61% y crece continuamente. El uso de Internet está explotando, y los muestreos como el Internet Operating System Counter muestran que Linux y otros sistemas operativos open source

son ya multitud entre los hosts de Internet y ganan cuota continuamente frente a los sistemas cerrados. La necesidad de explotar la infraestructura open source de Internet condiciona cada vez más no sólo el diseño de otro software sino las prácticas de negocio y los patrones de uso/compra de software de todas las empresas. Estas tendencias, en todo caso, parece probable que se aceleren.

Conclusión: Vida después de la revolución

¿Cómo será el mundo del software cuando la transición al open source sea completa?

Algunos programadores temen que la transición al open source pueda eliminar o devaluar sus trabajos. La pesadilla estándar es lo que llamo el escenario "Apocalipsis Open Source". Comienza con el valor del mercado del software aproximándose a cero por todo el código gratis que está ahí fuera. El valor de uso por sí mismo no atrae a los suficientes clientes como para soportar el desarrollo de software. La industria del software comercial se derrumba. Los programadores se mueren de hambre o dejan el sector. El apocalipsis llega cuando la propia cultura open source (que depende del tiempo libre de estos profesionales) se derrumba, sin dejar a nadie que sepa programar. Todos mueren. ¡Qué vergüenza!

Ya hemos visto bastantes razones por las que esto no ocurrirá, comenzando por el hecho de que los sueldos de la mayoría de los programadores no dependen del valor de venta del software. Pero la mejor, y que merece la pena resaltar aquí, es esta: ¿cuándo fue la última vez que vio a un grupo de desarrollo de software que no tuviera por delante un montón de trabajo? En un mundo que cambia sutilmente, en una economía que se hace cada vez más compleja y más centrada en la información, habrá siempre abundancia de trabajo y una demanda saludable de gente que pueda conseguir que los ordenadores hagan cosas; sin importar cuánto tiempo y cuántos secretos se publiquen.

Para el propósito de examinar el propio mercado del software, sería útil clasificar los tipos de software según la capacidad para describir el servicio que ofrecen mediante estándares técnicos abiertos, lo que se correlaciona bien con cómo de comoditizado se ha hecho el servicio subyacente.

Este eje se corresponde razonablemente bien con lo que la gente normalmente piensa cuando habla de "aplicaciones" (no comoditizados, estándares técnicos abiertos débiles o inexistentes), "infraestructura" (servicios comoditizados, estándares fuertes) y "middleware" (parcialmente comoditizado, estándares técnicos efectivos pero incompletos). Los casos paradigmáticos en la actualidad serían un procesador de textos (aplicación), una pila TCP/IP (infraestructura) y un motor de bases de datos (middleware).

El análisis de beneficio que hicimos antes sugiere que las infraestructuras, aplicaciones y middleware se transformarán de manera diferente y presentarán diferentes equilibrios entre código abierto y cerrado. También sugirió que la prevalencia del open source en un área particular de software sería una función de si hay efectos de red sustantivos que operen ahí, cuáles son los costes del fallo, y hasta qué punto el software es un bien crítico para el negocio.

Podemos aventurar algunas predicciones si aplicamos estas heurísticas no a productos individuales sino al segmento completo del mercado de software. Vamos allá:

La infraestructura (Internet, la Web, los sistemas operativos y los niveles inferiores del software de comunicaciones que tiene que traspasar fronteras entre agentes en competencia) será casi todos open source, mantenidos cooperativamente por consorcios de usuarios y por organizaciones de distribución y servicios que actúan por beneficio, con un papel semejante al de Red Hat hoy.

Las aplicaciones, por otro lado, tendrán la mayor tendencia a permanecer cerradas. Habrá circunstancias en las que el valor de uso de un algoritmo o tecnología secretos serán lo suficientemente altos (y los costes asociados con la falta de fiabilidad suficientemente bajos, y los riesgos asociados con un monopolio del fabricante suficientemente tolerables) como para que los

consumidores sigan pagando por software cerrado. Esto es más probable que siga siendo cierto en aplicaciones individuales para mercados verticales donde los efectos de red son débiles. Nuestro ejemplo anterior del aserradero es uno de estos; el software de identificación biométrico parece, de entre los candidatos de 1.999, ser otro.

El middleware (como bases de datos, herramientas de desarrollo, o los extremos personalizados de las pilas de protocolos de las aplicaciones) estará mas mezclado. Si las categoría de middleware tienden a ser cerradas o abiertas parece probable que dependa del coste de los fallos, donde los mayores costes crean presión del mercado hacia una mayor apertura.

Para completar la imagen, sin embargo, necesitamos observar que ni las “aplicaciones” ni el “middleware” son en realidad categorías estables. Anteriormente vimos que las tecnologías de software parecen atravesar un ciclo de vida natural de racionalmente cerrado a racionalmente abierto. La misma lógica se aplica en general.

Las aplicaciones tienden a convertirse en middleware según se desarrollan técnicas estandarizadas y porciones del servicio de comoditizan. (Las bases de datos, por ejemplo, se convirtieron en middleware después de que el SQL desacopló los front end de los motores). Según se comoditicen los servicios middleware, tenderán a su vez a convertirse en infraestructuras open source; una transición que estamos viendo en los sistemas operativos ahora mismo.

En un futuro que incluye la competición del open source, podemos esperar que el destino final de toda tecnología de software sea o morir o formar parte de la propia infraestructura abierta. Aunque esto no sea muy buena noticia para los emprendedores a los que les gustaría obtener rentas del software cerrado para siempre, sugiere que la industria del software en general seguirá siendo empresarial, con nuevos nichos abriéndose constantemente por arriba (aplicaciones) y un ciclo de vida limitado para los monopolios cerrados según sus categorías de productos se conviertan en infraestructura.

Por ultimo, por supuesto, este equilibrio será estupendo para los consumidores de software que están dirigiendo el proceso. Más y más software de calidad estará permanentemente disponible para ser usado y construir sobre él, en lugar de ser discontinuado o cerrado en el cofre de alguien. El caldero mágico de Ceridwen es, al final, una metáfora demasiado débil; porque la comida se consume o caduca, mientras que el software puede durar para siempre. El mercado libre, incluyendo en su más amplio sentido libertario todas las actividades no forzadas, sean negocios o regalos, puede producir para todos una riqueza de software que crezca incesantemente.

Epílogo: Por qué cerrar un Driver hace perder dinero a su fabricante

Los fabricantes de hardware de periféricos (tarjetas ethernet, controladores de disco, tarjetas de video y similares) han sido históricamente reacios a abrirse. Esto está cambiando ahora, con jugadores como Adaptec y Cyclades que comienzan a liberar de manera rutinaria especificaciones y código fuente de drivers para sus tarjetas. Sin embargo, todavía hay resistencia ahí fuera. En este apéndice intentaré disipar varios de los malentendidos económicos que la sostienen.

Si eres un fabricante de hardware, puedes temer que al liberar los fuentes estés revelando cosas importantes acerca de cómo trabaja tu hardware que los competidores podrían copiar, ganando así una ventaja competitiva injusta. En los días de los productos con ciclos de vida de tres a cinco años esto era un argumento válido. Hoy, el tiempo que los ingenieros de tu competencia necesitaría invertir en copiar y entender la copia es una porción dañinamente grande del ciclo de producto, un tiempo que no están invirtiendo en innovar o diferenciar su propio producto.

Esta no es una visión nueva. El antiguo jefe de la KGB Oleg Kalugin plantea el caso bien:

Por ejemplo, cuando robamos IBM en nuestros proyectos, o algunas otras áreas en la electrónica donde el Oeste hacía grandes avances y nosotros íbamos por detrás, nos llevaba años implementar los resultados de nuestros esfuerzos de inteligencia. Para entonces, en cinco o siete años, el Oeste había avanzado, y nosotros teníamos que robar una y otra vez, y quedábamos más y más atrás.

Pero Rudyard Kipling lo dijo mejor en su poema El Mary Gloster, hace casi un siglo. Escribió:

Y me preguntaron cómo lo hice, y yo les di el texto de la Escritura,"Mantén tu luz de manera que brille un poco por delante de lo siguiente" Ellos copiaron todo lo que pudieron seguir, pero no pudieron copiar mi mente,y les dejé sudando y robando un año y medio por detrás,

La aceleración del tiempo en Internet hace que este efecto muerda más fuerte. Si estas de verdad por delante del juego, ¡el plagio es una trampa en la que tú quieres que caigan tus competidores!

En cualquier caso, estos detalles no permanecen ocultos mucho tiempo en estos días. Los drivers de hardware no son como los sistemas operativos y las aplicaciones; son pequeños, fáciles de desensamblar y fáciles de clonar. Incluso un programador novato adolescente puede hacer esto; y a menudo lo hacen.

Hay literalmente miles de programadores de Linux y FreeBSD ahí fuera con la capacidad y la motivación para construir drivers para una tarjeta nueva. Para muchas clases de dispositivo que tenga interfaces relativamente simples y estándares bien conocidos (como controladores de discos y tarjetas de red) estos hackers ansiosos pueden a menudo prototipar un driver casi tan rápido como podría hacerlo tu propia empresa, incluso sin documentación y sin desensamblar un driver existente.

Incluso para dispositivos más complejos como tarjetas de video o de sonido, no hay mucho que puedas hacer para frustrar a un programador listo armado con un desensamblador. Los costes son bajos y las barreras legales porosas; Linux es un esfuerzo internacional y siempre hay una jurisdicción en la que la ingeniería inversa sea legal.

Para tener mayor evidencia de que todo esto es cierto, examina la lista de dispositivos soportados en el kernel de Linux y observa el ritmo al que se añaden otros nuevos, incluso sin el soporte del fabricante.

Otra buena razón para abrir tus drivers es que puedes concentrarte en la innovación. Imagina que ya no tienes que gastar el tiempo y el sueldo de tu equipo interno en reescribir, probar y distribuir nuevos binarios para cada nuevo kernel según aparece. Seguramente tendrás cosas mejores que hacer con todo ese talento.

Otra buena razón más: nadie quiere esperar seis meses para que salga un parche. Si tienes alguna competencia open source, es probable que te entierren sólo por esta razón.

Por supuesto, está el efecto de protección frente al futuro al que ya nos hemos referido. Los clientes quieren open source porque saben que extenderá el ciclo de vida del hardware más allá del punto hasta el que es efectivo desde el punto de vista del coste para ti soportarlo.

La mejor razón, sin embargo, es porque vender hardware es lo que produce dinero para ti. No hay ninguna demanda en el mercado para tu secreto; de hecho, es más bien al contrario. Si tus drivers son difíciles de encontrar, si tienen que ser actualizados frecuentemente, o si (lo peor de todo) se ejecutan mal, esto se refleja mal en tu hardware y venderás menos. Abrir los fuentes puede resolver estos problemas e incrementar tus ingresos.

¿El mensaje? Mantener tu driver secreto parece atractivo a corto plazo, pero es probablemente una mala estrategia a largo plazo (y más aún si estás compitiendo con otros fabricantes que ya son abiertos). Pero si debes hacerlo, graba el código en una ROM en la placa. Después publica el interfaz de la ROM. Libera tanto como sea posible para construir tu mercado y demostrar a los clientes potenciales que crees en tu capacidad para pensar e innovar más que tus competidores allí donde importa.

Si permaneces cerrado normalmente tendrás lo peor de los dos mundos: tus secretos podrán ser

revelados, no tendrás ayuda para el desarrollo gratuita, y no habrás malgastado el tiempo de tus competidores más estúpidos en la clonación. Pero lo más importante, perderás una vía para extender la adopción temprana. Un mercado grande e influyente (la gente que gestiona los servidores que ejecutan Internet y muchos CPDs de empresas) identificarán a tu compañía como estúpida y defensiva, porque no eres consciente de estas cosas. Entonces comprarán sus placas a alguien que lo sea.

Notas

[SC] El problema de la infraprovisión en realidad escalaría linealmente con el número de usuarios si suponemos que el talento de programación está distribuido uniformemente en la población de usuarios del proyecto cuando este se expande en el tiempo. Este no es, sin embargo, el caso.

Los incentivos discutidos en [HtN] (y algunos otros más convencionales desde el punto de vista económico) implican que la gente cualificada tiende a buscar proyectos que satisfagan sus intereses, además de los proyectos que les buscan a ellos. Según esto, la teoría sugiere (y la experiencia tiende a confirmar) que las personas más valiosas (más cualificadas y motivadas) tenderán a descubrir proyectos para los que encajen bien relativamente pronto en el ciclo de vida del proyecto, con una caída correspondiente más adelante.

Faltan datos empíricos, pero en base a la experiencia sospecho fuertemente que la asimilación de talento sobre la vida de un proyecto en crecimiento tiende a seguir una clásica curva logística.

[SH] Shawn Hargreves ha escrito un buen análisis de la aplicabilidad de los métodos open source a los juegos: Jugando al juego Open Source.

[DPV] Nota para contables: el argumento de que los costes del servicio al final inundan el precio fijo inicial todavía funciona si pasamos de dólares constantes al valor presente descontado, porque el futuro ingreso por ventas se descuenta en paralelo con los futuros costes de servicio.

Una replica similar pero más sofisticada a este argumento es observar que, por copia, los costes de servicio pasan a ser cero cuando el comprador deja de usar el software: por tanto, todavía puedes ganar si el usuario se para antes de que haya generado demasiado coste de servicio. Esta es básicamente otra forma del argumento de que la asignación de precios según el modelo de fábrica recompensa la producción de shelfware. Quizá una manera más instructiva de expresarlo sería que el riesgo de que sus costes de servicio inunden los ingresos por compra aumenta con el periodo esperado de utilidad del software. Así, el modelo de fábrica penaliza la calidad.

[WG] Wayne Gramlich Wayne@Gramlich.net ha propuesto que la persistencia del modelo de fábrica se debe parcialmente a reglas de contabilidad anticuadas, formuladas cuando las máquinas y los edificios eran más importantes y las personas menos. Los libros de contabilidad de las compañías de software muestran los ordenadores, el mobiliario de oficina y los edificios como activos y los programadores como costes. Por supuesto, en realidad, los programadores son los verdaderos activos y los ordenadores, equipos de oficina y edificios no tienen la menor importancia. Esta valoración perversa se sostiene por la presión de los IRS y el mercado de valores hacia reglas de contabilidad estables y uniformes que reduzcan la complejidad de asignar una cifra en dólares al valor de la compañía. El arrastre resultante ha provocado que las reglas se alejen de la realidad.

Desde este punto de vista, asignar un precio alto a los bits del producto (independiente del valor del servicio futuro) es parcialmente una forma de mecanismo de defensa, una manera de acordar entre todas las partes implicadas la pretensión de que los cimientos ontológicos no se han derrumbado bajo las reglas estándar de contabilidad.

(Gramlich también señala que estas reglas subyacen en la extraña y a menudo autodestructiva manía de adquisición en la que caen muchas compañías de software después de salir a bolsa. “Normalmente las compañías de software lanzan algunas acciones adicionales para construir un equipo de guerra. Pero no pueden gastar este dinero para mejorar su plantilla de programadores,

porque las reglas de contabilidad mostrarían que están incrementando los gastos. En lugar de eso, la compañía de software tiene que crecer comprando otras compañías de software, porque las reglas de contabilidad le permiten tratar la adquisición como una inversión.")

Para ver un ejemplo paradigmático de bifurcación que sigue a la defección, consulte la historia de OpenSSH. Este proyecto fue tardíamente bifurcado de una versión temprana de SSH (Secure Shell) después de que ésta pasara a una licencia cerrada.

Bibliografías

[CatB] La Catedral y el Bazar.

[HtN] Homesteading the Noosphere.

[DL] De Marco y Lister, Peopleware: Productive Projects and Teams (New York; Dorset House, 1987; ISBN 0-932633-05-6)

Agradecimientos

Varias conversaciones estimulantes con David D. Friedman me ayudaron a refinar el modelo de "comunes inverso" de la cooperación open source. También estoy en deuda con Marshall van Alstyne por señalar la importancia conceptual de los bienes de información rivales. Ray Ontko del Indiana Group proporciono crítica constructiva. Muchas personas en auditorios ante los que he dado charlas en el año que va hasta Junio de 1.999 también ayudaron: si es una de ellas, usted sabe quién es.

Todavía es otro testimonio del modelo open source que este ensayo haya sido mejorado sustancialmente por el feedback del email que he recibido en los días siguientes al lanzamiento inicial. Lloyd Word señaló la importancia de que el open source estuviera "a prueba de futuro", y Doug Dante me recordó el modelo de negocio "Libera el Futuro". Una pregunta de Adam Moorhouse condujo a la discusión de los beneficios de la exclusión. Lionel Oliveira Gresse me dio un nombre mejor para uno de los modelos de negocio. Stephen Turnbull me llamó la atención sobre el descuidado tratamiento de los efectos de los aprovechados. Anthony Bailey y Warren Young corrigieron algunos datos en el caso de estudio de Doom. Eric W. Sink contribuyó con la visión de que el modelo de fábrica recompensa al shelfware.