EL CALDERO MÁGICO

Eric S. Raymond

El caldero mágico es un ensayo de Eric S. Raymond sobre el modelo económico que sustenta el desarrollo de software de código abierto. Su título en inglés es The Magic Cauldron. Fue publicado por O'Reilly en 1999 en su libro The Cathedral and the Bazaar. El ensayo analiza los modelos económicos que sustentan los proyectos libres en cuatro pasos: Comienza criticando algunos mitos sobre el retorno de la inversión en desarrollo de software libre y presenta un modelo basado en la teoría de juegos sobre la estabilidad y escalabilidad de la cooperación en el software abierto. En segundo lugar, recopila nueve modelos de desarrollo abierto sostenible: dos para actividades sin ánimo de lucro y otros siete para iniciativas orientadas al lucro. A continuación describe una teoría para decidir cuando es económicamente rentable desarrollar un programa en modelo abierto o cerrado. Por último, examina algunos mecanismos (entonces recientes) compatibles con el mercado para apoyar el desarrollo de software de código abierto (como el patronazgo o los mercados de tareas a contratar)

Eric S. Raymond

El Caldero Mágico

ePUB v1.0
ysidoro 11.10.11

Autor: Eric Steven Raymond

Versión del documento: 1.14
Traducido por: Diego Rodrigo
el 9 de Julio de 1999
Versión de la traducción: 1.3

Resumen

Este artículo analiza el sustrato económico evolucionante del fenómeno open-source. Al principio derribaremos algunos mitos prevalescientes acerca del soporte económico para el desarrollo de programas y de la estructura de precios del software. Presentaremos un análisis a través de teoría de juegos de la estabilidad de la cooperación open-source. Presentaremos nueve modelos de desarrollo open-source sustentables económicamente; dos sin fines de lucro, y siete con fines de lucro. Continuaremos desarrollando una teoría cualitativa acerca de cuando es económicamente razonable ser cerrado. Luego examinaremos algunos mecanismos nuevos que el mercado está inventando para sustentar económicamente el desarrollo open-source con fines de lucro, incluyendo la reinvención del sistema de mecenazgo o patrocinio y las tareas de mercado. Concluiremos con algunas predicciones tentativas del futuro.

Indistinguible de la Magia

En la mitología galesa, la diosa Ceridwen poseía un gran caldero, el cual magicamente producía alimentos - cuando recibía las palabras mágicas que sólo la diosa conocía. En la ciencia moderna, Buckminster Fuller nos dió el concepto de "efemeralización" ("ephemeralization"), la tecnología cada vez es más efectiva y más barata, mientras que los recursos físicos invertidos en diseños previos son reemplazados por más y más información. Arthur C. Clarke conectó ambas afirmaciones observando que "Cualquier tecnología suficientemente avanzada es indistinguible de la magia".

Para mucha gente, el éxito de la comunidad open-source aparece como una forma poco creible de magia. Software de alta calidad se materializa "gratis", lo cual es muy lindo mientras dure pero parece dificilmente sostenible en el mundo real de la competitividad y de los recursos escasos. ¿Cuál es la idea? ¿El caldero de Ceridwen es solo un truco? y si no lo es, ¿cómo trabaja la efemeralización en este contexto - qué conjuro está utilizando la diosa?

Mas Allá de Geeks Trayendo Regalos

La experiencia de la cultura open-source ciertamente ha confundido muchas de las afirmaciones que la gente aprendió acerca del software fuera de esta cultura. "La Catedral y el Bazaar" ("The Cathedral and the Bazaar") [#!CatB!#] describió los modos en los cuales el desarrollo descentralizado y cooperativo de software quebranta la ley de Brooks, llevando a niveles de calidad y confiabilidad sin precedentes en proyectos individuales. "Homesteading the Noosphere" [#!HtN!#] examinó la dinamica social dentro de la cual está situado este estilo de desarrollo tipo "bazaar", fundamentando que no es efectivamente entendido en términos de una economía de intercambio convencional, sino en lo que los antropólogos llaman una "cultura del regalo" en la cual los miembros compiten por status donando o regalando cosas. En este artículo comenzaremos derribando algunos mitos comunes acerca de la economía de la producción de software; luego continuaremos el análisis de [#!CatB!#] y [#!HtN!#] dentro del dominio de la economía, teoría de juegos y modelos de negocios, desarrollando nuevas herramientas conceptuales necesarias pasa comprender la manera en que la cultura del regalo de los desarrolladores open-source puede sustentarse por si misma dentro de una economía del intercambio.

Para continuar esta línea del análisis sin distracción, necesitaremos abandonar (o por lo menos ignorar temporalmente) el nivel de explicación de "cultura del regalo". [#!HtN!#] sugiere que el comportamiento de la cultura del regalo surge en situaciones en las cuales los bienes necesarios para la supervivencia son lo suficientemente abundantes como para transformar el juego del cambio en algo no muy interesante; pero mientras esto parece suficientemente poderoso como explicacion psicológica del comportamiento, carece de suficiencia como explicación del contenido económico mezclado en el cual la mayoría de los desarrolladores open-source operan actualmente. Para la mayoría, el juego del intercambio ha perdido su gracia pero no su poder para contener. Su comportamiento tiene que generar suficiente sentido de economía de escasez de materiales para mantenerlos en la zona de la cultura del regalo.

Por lo tanto ahora consideraremos (completamente desde adentro del dominio de la economía de la escasez) los modos de cooperación e intercambio que sustentan el desarrollo open-source. Mientras hacemos esto, nos haremos la pregunta pragmática: "Como hago dinero con esto?", en detalle y con ejemplos. Primero, sin embargo, mostraremos que mucha de la tensión detrás de esta pregunta deriva de modelos tradicionales de economía de produccion de software que son falsos.

(Una nota final antes de la exposición: la discusión y defensa del desarrollo open-source en este artículo no debe ser entendida como una afirmación que el desarrollo closed-source es intrínsecamente malo, ni como un caso contra los derechos de

propiedad intelectual en el software, ni como una apariencia altruista de "compartir". Aunque estos argumentos todavía son queridos por una minoría en la comunidad de desarrollo open-source, la experiencia desde [#!CatB!#] ha hecho claro que no son necesarios. Un caso completamente suficiente para el desarrollo open-source descansa en sus salidas tanto en lo ingenieril como en lo económico - mejor calidad, alta confiabilidad, y gran posibilidad de elección.

La Ilusión de la Manufactura

Necesitamos comenzar notando que los programas de computacion como cualquier otra herramienta o bien de capital, tiene dos clases distintas de valor económico. Tienen el valor de uso y el valor de venta.

El valor de uso de un programa es su valor económico como herramienta. El valor de venta de un programa es su valor como una mercancia vendible (commodity). (En lenguaje económico profesional, el valor de venta es el valor como bien final, y el valor de uso es el valor como bien intermedio.)

Cuando la mayoría de la gente trata de razonar acerca de la economía de producción de software, trata de asumir un "modelo de fábrica" que esta basado en las siguientes premisas fundamentales.

1. La mayor parte del tiempo del desarrollador se paga por el valor de venta.

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

En otras palabras, la gente tiene una fuerte tendencia a asumir que el software tiene las características de valor de un bien manufacturado típico. Pero ambas suposiciones son demostrablemente falsas.

Primero, el código escrito para vender es sólo la punta del iceberg de la programación. En la era previa a la microcomputadora solía ser un hecho conocido por todos que el 90% de todo el código en el mundo era escrito artesanalmente (en casa 1) en bancos y compañias de seguros. Este probablemente ya no es el caso - otras industrias producen software de forma mas intensiva hoy, y el porcentaje de la industria de las finanzas ha caido - pero veremos brevemente que hay evidencia empírica que alrededor del 95% del código todavía es escrito en casa.

Este código incluye la mayoría de las cosas de MIS, la adecuación del software financiero y de bases de datos que toda mediana y gran compañia necesita hacer. Incluye código técnico especializado, como device drivers (casi nadie hace dinero vendiendo device drivers, un punto al que volveremos mas adelante). Incluye toda clase de código embebido (o empotrado) (embedded) para nuestras (cada vez más) máquinas controladas por microchips - desde herramientas y aviones hasta autos, hornos microondas y tostadoras.

La mayoría de este codigo hecho en casa esta integrado con su ambiente en modos que hacen que su reuso o copia sea muy dificultosa. (Esto es tan cierto si el ambiente es un conjunto de procedimientos de oficina, como si es el sistema de inyección de una cosechadora) Entonces, cuando el ambiente cambia, hay mucho trabajo que se necesita continuamente para mantener el software funcionando.

Esto se llama "mantenimiento", y cualquier ingeniero de software o analista de

sistemas dirá que compone la gran mayoría (mas del 75%) del trabajo por el que se le paga a un programador. De acuerdo a esto, la mayoría de las horas de programación se gastan en (y la mayoría de los salarios de los programadores se pagan por) escribir y mantener código "hecho en casa" que no tiene ningun valor de venta - un hecho que el lector puede confirmar examinando el listado de empleos pedidos para trabajos de programación en los avisos clasificados del periódico.

Buscar en la sección de empleos de su periódico local es una experiencia iluminante, por eso pido al lector que la realize. Examine el listado de trabajos en posiciones de programación, procesamiento de datos, e ingeniería de software que involucren desarrollo de software. Clasifique cada trabajo de acuerdo a si el software es desarrollado para uso o para venta.

Rapidamente quedara claro que, aunque se de la definicion mas amplia de "para venta", por lo menos 19 de 20 salarios son soportados estrictamente por el valor de uso (esto es, el valor de un bien intermedio). Esta es nuestra razon para creer que sólo el 5% de la industria es movida por el valor de venta. Nótese, sin embargo, que el resto del análisis en este artículo es relativamente intenso respecto a este número; si fuese 15% o 20%, las consecuencias económicas serían esencialmente las mismas.

(Cuando hablo en conferencias técnicas, generalmente comienzo mi charla haciendo dos preguntas: cuantos en la audiencia reciben pago por desarrollar software, y para cuantos su salario depende del valor de venta del software. Generalmente obtengo un bosque de manos para la primera pregunta, pero muy pocas o ninguna para la segunda, y hasta la misma audiencia se sorprende de la proporción.)

Segundo, la teoría que dice que el valor del software está acoplado a su costo de desarrollo o de reemplazo es mas facilmente demolida examinando el comportamiento real de los consumidores. Hay muchos bienes para los cuales una relación de este tipo les cabe (antes de la depreciación) - comida, autos, herramientas. Hay tambien muchos bienes intangibles para los cuales el valor de venta se acopla fuertemente al costo de desarrollo y de reemplazo - derechos para reproducir musica o mapas o bases de datos, por ejemplo. Estos bienes pueden retener o incluso aumentar su valor de venta despues que su productor se ha retirado del mercado.

En contraste, cuando el productor de software sale del mercado (o meramente si el producto es discontinuado), el mayor precio que los consumidores pagarán por él rapidamente cae casi hasta cero sin importar su valor teórico de uso o el costo de desarrollo de un bien funcionalmente equivalente. (Para confirmar esta afirmación, examine los contenedores de productos discontinuados 2 en cualquier casa de venta de software.)

El comortamiento de los vendedores a consumidor final cuando un productor se retira (fold) es muy revelador. Nos dice que ellos saben algo que los productores no saben. Lo que saben es esto: el precio que un consumidor pagará se ve efectivamente

disminuido por el valor futuro esperado del servicio prestado por el productor (donde "servicio" aquí se entiende ampliamente para incluir mejoras, actualizaciones y proyectos siguientes).

En otras palabras, el software es una industria de servicios operando bajo la persistente pero infundada ilusión que es una industria de manufacturas.

Vale la pena examinarlo porque normalmente tendemos a creer otra cosa. Simplemente puede ser porque la pequeña porción de la industria del software que vende software es tambien la única que promociona y publicita su producto. Ademas, algunos de los productos mas visibles y mas altamente promocionados son emíferos.

También vale la pena notar que la ilusión de la manufactura alienta estructuras de precios que estan patológicamente fuera de linea con la actual falla (breakdown) en los costos de desarrollo. Si (como se acepta generalmente) mas del 75% del costo del ciclo de vida de un proyecto de software típico se utilizará en mantenimiento, depuración y extensiones, entonces la política común de precios de cargar un alto precio de adquisición y una cuota de soporte relativamente baja o nula nos lleva a resultados que no conforman a ninguna de las partes.

Los consumidores pierden porque, aunque el software es una industria de servicios, los incentivos en el modelo de fabrica se vuelven en contra del servicio competente ofrecido por un productor. Si el dinero del productor viene de vender retazos, más esfuerzo hara para construir retazos y mostrarlos en la puerta; la mesa de ayuda (help desk), no sera un centro de ganancias, se convertira en un campo de volcado de desperdicios de lo menos efectivo y tomará suficientes recursos sólo para evitar activamente alienar un numero crítico de clientes.

La otra cara de la moneda es que los productores que compren este modelo de fábrica también caerán en el largo plazo. Mantener un soporte indefinidamente continuo con un precio fijo solo es posible en un mercado que se esta expandiendo lo suficientemente rápido como para cubrir el soporte y los costos del ciclo de vida en los que se incurrió por el software vendido ayer con las entradas del software que se venderá mañana. Una vez que el mercado madura y las ventas decaen, los productores no tendrán otra opción que cortar los gastos dejando huérfano al producto.

Si esto se hace explícitamente (discontinuando el producto) o implícitamente (haciendo que el soporte sea dificil de obtener), tiene el efecto de llevar los clientes hacia la competencia (porque destruye el valor futuro esperado, el cual es contingente en ese servicio). En el corto plazo, uno puede escapar de esta trampa haciendo que las versiones que reparan errores (bug-fix) aparezcan como nuevos productos con un nuevo precio, pero los consumidores rapidamente se cansan de esto. En el largo plazo, el unico modo de escapar es no tener competidores - esto es, tener un monopolio efectivo en el mercado en el cual uno esta inserto. Al final, solo podrá

existir uno solo.

Y ciertamente, hemos visto este modo de falta de soporte matar inclusive a competidores fuertes de segundo lugar en algun nicho del mercado. (Este escenario debería ser particularmente claro para cualquiera que haya sobrevivido la historia de un sistema operativo propietario para PC, procesadores de texto, programas de contabilidad o software de negocios en general.) Los incentivos perversos establecidos por el modelo de fábrica llevan a una dinámica de mercado llamada "el ganador toma todo" en la cual hasta los clientes del ganador terminan perdiendo.

Si no es el modelo de fábrica, entonces que? Para manejar la estructura de costo real de un ciclo de vida de software eficientemente (en el sentido informal y económico de "eficiencia"), necesitamos una estructura de precios fundada en los contratos de servicio, subscripciones, y un intercambio continuo de valor entre el productor y el cliente. Bajo las condiciones buscadoras de eficiencia dl mercado libre, podemos predecir que esta es la clase de estructura de precios que la mayor parte de la industria madura de software seguira al fin.

Todo lo dicho comienza a darnos una cierta vista en el porque el software open-source establece no solamente un desafío tecnológico sino también un reto econiómico al orden previamente establecido. Los efectos de construir software "libre", parece que serian los de forzarnos a entrar en ese mundo de servicios dominado por la cuota - y a exponer que la venta de retazos closed-source era una proposición relativamente debil.

La palabra "libre" es confusa en cierto sentido. Bajar el costo de un bien tiende a subir, más que a bajar, la inversion total en la infraestructura que lo sustenta. Cuando el precio de un auto baja, la demanda de meciánicos de autos sube - por esto es que hasta ese 5% de programadores que ahora estan compensados por el valor de venta no sufrirían en un mundo open-source. La gente que pierda en la transicion no seran los programadores, seran los inversores que han apostado a estrategias closed-source cuando no son apropiadas.

El Mito: "la información quiere ser libre"

Hay otro mito, igual y opuesto a la ilusión del modelo de fábrica, el que frecuentemente confunde los pensamientos de la gente acerca de la economía del software open-source. El mito es que "la información quiere ser libre". Esto generalmente lleva a decir que el costo marginal nulo asociado a la reproduccción digital de información implica que su precio deberia ser cero.

La forma mas general de este mito explota cuando consideramos el valor de la información que constituye un reclamo sobre un bien disputado - un mapa del tesoro, el numero de cuenta de un banco suizo, o el password de una cuenta de computadora.

Mencionamos este mito principalmente para afirmar que no está relacionado de modo alguno con los argumentos de utilidad económica del desarrollo open-source; como veremos mas adelante. Entonces por ahora no abordaremos la pregunta acerca de si el software "debería" ser libre o no.

Los Campos Inversos

Habiéndole echado un vistazo escéptico al modelo reinante, veamos si podemos construir otro modelo - una explicación altamente económica de que es lo que hace que la cooperación open-source sea sustentable.

Esta es una pregunta que merece examinarse en un par de niveles distintos. En un nivel, necesitamos explicar el comportamiento de individuos que contribuyen a proyectos open-source; en otro nivel, necesitamos entender las fuerzas económicas que sustentan la cooperación en proyectos open-source como Linux o Apache.

Nuevamente, primero debemos demoler un modelo tradicional bastante extendido que interfiere con el entendimiento. Sobre cada intento de explicar el comportamiento cooperativo aparece la sombra de la Tragedia del Campo Abierto ((Commons)) de Garret Hardin.

Hardin nos pide que imaginemos una extensión verde que es utilizada en común por una villa de granjeros que alimentan su ganado allí. El hecho de pastar degrada al campo, debilitando el pasto y dejando zonas enlodadas, en las cuales sólo vuelve a crecer pasto lentamente. Si no existen políticas acordadas (y respetadas!) para establecer zonas de pastoreo que prevengan la sobreutilización del campo, cada granjero tratará de colocar la mayor cantidad de ganado lo más rapido que pueda, tratando de extraer lo máximo del campo antes que se degrade en un mar de barro.

Mucha gente tiene un modelo intuitivo de cooperación muy parecido a esto. No es realmente un buen diagnóstico de los problemas económicos de open-source, los cuales son libres (por debajo de la provisión) mas que bienes públicos congestionados (sobreuso).

La Tragedia del Campo predice sólo tres resultados posibles. Uno es el mar de barro. Otro es cuando algún actor con el suficiente poder cohersitivo establece una política de pastoreo para la villa (la solución comunista). El tercer resultado es que se rompe la villa, y los granjeros alambran el campo en sectores que puedan administrar y defender adecuadamente (la solución de los derechos de propiedad).

Cuando se aplica este modelo a la cooperacion open-source, se espera que será inestable con una vida media corta. Como no hay un medio razonable de forzar una politica para el tiempo de los programadores en internet, este modelo lleva directamente a predecir que los programadores se dividirán, quedando muchos pedazos de software que se volveran closed-source y rapidamente decrecerá la cantidad de trabajo que se "tire a la pileta común".

De hecho, empíricamente es claro que la tendencia es totalmente opuesta a esto. La cantidad y volumen del desarrollo open-source (medido por ejemplo en envíos diarios a Metalab o anuncios por día en freshmeat.net) esta creciendo continuamente. Claramente existe algun modo crítico en el cual el modelo de la "Tragedia del

Campo" falla.

Parte de la respuesta ciertamente yace en que usar el software no hace bajar su valor (no se deprecia por el uso). De hecho, el uso masivo de software open-source tiende a aumentar su valor, porque los usuarios remiten sus propios arreglos y caracteristicas (patches). En este "campo inverso", mientras mas se pastorea mas crece el pasto.

Otra parte de la respuesta se basa en el hecho que el valor de mercado de un parche (patch) para una fuente común es dificil de capturar. Suponiendo que yo escriba un arreglo para un error irritante, y suponiendo que mucha gente se da cuenta que el arreglo tiene un valor monetario; ¿como recolecto el dinero de toda esa gente? Los sistemas convencionales de pago tienen altos costos como para adecuarse a la clase de micropagos que sería apropiada.

No solo es difícil capturar el valor, sino que también es difícil asignar un valor. Supongamos que internet viniera equipada con el sistema de pago teórico ideal para micropagos - seguro, universalmente accesible, y con costo nulo. Digamos que alguien escribio un parche llamado "Arreglos Varios al Kernel Linux". ¿Cómo sabe el autor que precio pedir? Cómo sabe un comprador potencial, sin haber visto el parche, cuanto vale la pena pagar por el?

Aqui tenemos como una imagen un tanto distorsionada del "problema del calculo" de F.A. Hayek's - sólo un ser superior podría ser capaz de evaluar el valor funcional de los parches y establecer los precios de acuerdo a esto.

Desafortunadamente, hay una gran escasez de seres superiores, entonces el autor del parche J. Random Hacker tiene sólo dos alternativas: quedarse con el parche, o tirarlo a la pileta común libremente (gratis). Con la primera opción no ganará nada. Con la segunda opción puede no ganar nada, o puede favorecer el mismo comportamiento recíproco de otros que tendrán el mismo problema del señor J. Random Hacker. La segunda opción, aunque aparentemente altruista, es en realidad óptimamente egoista en un sentido de teoría de juegos.

Al analizar esta clase de cooperación, se debe notar que no es un problema que escala con el número de usuarios finales. Los costos de complejidad y comunicación de un proyecto open-source están enteramente en función del numero de desarrolladores involucrados; tener más usuarios finales que nunca miran el código fuente no agrega ningun costo adicional. Puede aumentar la cantidad de preguntas tontas en las listas de correo del proyecto, pero esto puede solucionarse publicando una lista de Preguntas Frecuentes (FAQ) y simplemente ignorando a las personas que obviamente no la han leido (de hecho estas prácticas son típicas).

Los problemas de los desarrolladores open-source estan más ralacionados con los costos de fricción en enviar parches que con otra cosa. Un contribuyente potencial con poca fama en el juego de la reputación cultural (ver [#!HtN!#]) puede pensar, en

ausencia de una compensación monetaria, "No vale la pena enviar este arreglo porque tengo que limpiar el patch, escribir un archivo de cambios, y firmar los papeles de la FSF". Por esta razón el numero de contribuyentes (y el orden de exito) de los proyectos es inversamente proporcional a la cantidad de vueltas que le hacen dar a un usuario para transformarse en contribuyente. Esos costos de fricción pueden ser politícos o mecánicos. Entonces se puede explicar como la cultura Linux, amorfa y suelta, ha atraido varios ordenes de magnitud más energía cooperativa que los esfuerzos de BSD, mas centralizados y fuertemente organizados; y porque la FSF (Free Software Foundation = Fundacion Software Libre) ha decaido en importancia mientras que Linux ha subido.

Hasta aquí esta todo bien. Pero esta es una explicación "despues del hecho", lo que J. Random Hacker decidió hacer despues de escribir su patch. La otra mitad que necesitamos es una explicación económica de como J. Random Hacker fue capaz de escribir el patch en primer lugar, en lugar de trabajar en un programa closed-source que podria haberle retornado algo como valor de venta. ¿Qué modelos de mercado crean nichos en los cuales el desarrollo open-source puede florecer?

Razones para Cerrar las Fuentes

¿Qué es lo que protegemos exactamente cuando cerramos las fuentes (closed-source)?

Digamos que el lector contrata a alguien para que escriba un paquete de contabilidad para su negocio. El problema no será resuelto mejor si las fuentes estan abiertas que cerradas; las únicas razones lógicas para utilizar un esquema cerrado serían si se trata de vender el paquete, o negar su uso a competidores.

La respuesta obvia es que se está protegiendo el valor de venta, pero para el 95% del software escrito para uso interno esta razon no se aplica. Entonces que otras ganancias existen en ser cerrado?

El segundo caso (proteger la ventaja competitiva) merece un poco de examen. Supongamos que el lector decide cambiarse al modelo open-source y abrir las fuentes del paquete. Éste se convierte en popular, y el lector se beneficia por las mejoras hechas por la comunidad. Ahora, el competidor comienza a usarlo. El competidor obtiene el beneficio sin incurrir en gastos de desarrollo. ¿Es éste un argumento en contra de open-source?

Quizás - y quizás no. La pregunta real es si las ganancias obtenidas por liberar el paquete exeden las pérdidas por una aumentada competencia. Por hipótesis, los costos de desarrollo deben pagasre de una manera u otra, entonces es erroneo contarlos como costos de open-source.

Hay otras razones para cerrar las fuentes que son realmente irracionales. Se podría estar trabajando bajo la ilusión que cerrar las fuentes convertirá a los sistemas de negocios en sistemas más seguros contra crackers e intrusos. Si el lector piensa esto, le recomiendo unas sesiones de conversación terapéutica con un experto en criptografía inmediatamente. Los paranoicos profesionales han aprendido con la experiencia a no confiar en la seguridad de programas cerrados. La seguridad es un aspecto de la confiabilidad; sólo los algoritmos e implementaciones que han sido minusciosamente revisados entre colegas pueden ser confiados como seguros.

Modelos de Sustentación del Valor de Uso

La distinción entre valor de venta y valor de uso nos permite notar que solamente el valor de venta es amenazado por un cambio desde closed-source hacia open-source; mientras que el valor de uso no lo es.

Si el valor de uso es el mayor motor del desarrollo de software, y (como se argumento en [#!CatB!#]) el desarrollo open-source es realmente mas efectivo y eficiente que el desarrollo cerrado, entonces deberiamos encontrar circunstancias en las cuales el valor esperado de uso sostenga por si mismo el desarrollo open-source.

Y de hecho no es difícil identificar al menos dos modelos importantes en los cuales los salarios de desarrolladores full-time en proyectos open-source son sostenidos únicamente por el valor de uso.

El caso Apache: compartir costos

Digamos que el lector trabaja en una firma que tiene requerimientos críticos de negocios para un servidor web de alto volumen, y alta confiabilidad. Quizás es para comercio electrónico, quizás la empresa es un medio masivo de alta visibilidad que vende publicidad. Se necesita disponibilidad 24/7 (24 horas al día, 7 días a la semana), se necesita velocidad y posibilidad de adecuación.

Cómo va a obtener estas cosas? Hay tres estrategias básicas que pueden seguirse:

Comprar un servidor web propietario.

En este caso el lector apuesta a que los diseños del productor concuerden con las necesidades de la empresa, y que tenga la competencia técnica como para implementarlo. Suponiendo que ambas cosas pueden cumplirse, el producto muy posiblemente tenga poco espacio para adecuaci'on; sólo podrá ser modificado a través de los mecanismos provistos por el productor. Este camino del servidor web propietario no es muy popular.

Construir un servidor web

Construir un servidor web propio no es una opción como para descartar inmediatamente; los servidores no son muy complejos, ciertamente menos que los navegadores. En este camino se puede obtener las características y la capacidad de adaptacion deseadas, aunque se deberá pagar con tiempo de desarrollo. La empresa encontrará que tendrá problemas cuando el desarrollador se retire o deje la compañía.

Unirse al grupo Apache.

El servidor web Apache fue construido por un grupo de webmasters conectados por internet quienes se dieron cuenta que era mucho mas inteligente juntar sus esfuerzos en mejorar un solo código base que conducir esfuerzos de desarrollo paralelo por separado. De este modo, obtuvieron lo mejor de los dos métodos, construir el servidor web propio y el efecto poderoso de revisiones paralelas masivas.

La ventaja de la elección de Apache es muy fuerte. En la encuesta mensual de Netcraft de Junio de 1996, se muestra que Apache y sus derivados tienen una porción del mercado de 61% (http://www.netcraft.com/survey/), sin un dueño legal, sin promoción, y sin ninguna organización de servicios contratada detrás.

La historia de Apache se generaliza en un modelo en el cual los usuarios de software descubren que es ventajoso soportar el desarrollo open-source, porque haciéndolo obtienen un mejor producto a un menor costo.

El caso Cisco: disminuir el riesgo

Hace algunos aõs en Cisco, se asignó a dos programadores la tarea de escribir un sistema de colas de impresión distribuido para ser usado en la red corporativa de Cisco. Este trabajo era un gran reto. Ademas de la hablidad de permitir que un usuario arbitrario A imprimiera en una impresora arbitraria B (la cual podría estar en la habitacion siguiente o a mil millas de distancia), el sistema debía asegurar que cuando existiesen condiciones de falta de papel o bajo nivel de toner los trabajos deberían ser encaminados hacia una impresora alternativa cercana a la original. El sistema tambien necesitaba ser capaz de reportar todo tipo de problemas a u administrador de impresión.

El dúo terminó con una serie inteligente de modificaciones al software standard de impresión de Unix, ademas de algunos scripts, que hacian el trabajo pedido. Entonces se dieron cuenta que Cisco tenía un problema.

El problema era que ninguno de ellos iría a estar en Cisco para siempre. Eventualmente, ambos prgramadores podrían irse, y el software quedaría falto de mantenimiento y podría comenzar a degradarse (dejar de estar "en sintonía" con las condiciones del mundo real). Ningún desarrollador desea ver que esto le ocurra a su trabajo, y el dúo intrépido sentía que Cisco había pagado por una solución bajo la no poco razonable expectativa que el trabajo duraría sin importar la permanencia o no del dúo en Cisco.

De acuerdo a este razonamiento, pidieron a su jefe que autorice el lanzamiento del software de impresión como open-source. Su argumento era que Cisco no tenía ningún valor de venta que perder y mucho por ganar. Favoreciendo el crecimiento de una comunidad de usuarios y co-desarrolladores, Cisco podía luchar contra la pérdida de los desarrolladores originales del software.

La historia de Cisco se generaliza en un modelo en el cual la idea de código abierto funciona no tanto para bajar costos sino más bien para disminuir el riesgo. Todas las partes encuentran que la apertura del código , y la presencia de una comunidad cooperativa, provee un seguro contra fallas que es económicamente valuable - suficientemente valuable como para destinar recursos a él.

Porque el Valor de Venta es Problemático

Open-source hace que sea bastante difícil capturar el valor de venta del software. La dificultad no es técnica; el código fuente no es mas ni menos copiable que los binarios, y las regulaciones establecidas por las leyes de licencias y copyright que permiten capturar el valor de venta no necesariamente hacen que la captura sea mas difícil para productos código abierto que para cerrados.

La dificultad yace principalmente en la naturaleza del contrato social que soporta el desarrollo open-source. Pro tres razones, las mayores licencias open-source prohiben la mayoría de las restricciones sobre el uso, redistribución y modificación que facilitarían la captura de dinero proveniente de la venta directa. Para entender estas razones, debemos examinar el contexto social dentro del cual evolucionaron las licencias: la cultura hacker (http://www.tuxedo.org/ esr/faqs/hacker-howto.html) de internet.

A pesar de los mitos acerca de la cultura hacker, los cuales todavía (1999) son ampliamente difundidos fuera de ella, ninguna de esas razones tiene que ver con una hostilidad hacia el mercado. Mientras una minoría de los hackers permanece hostil al motivo de ganar dinero, la voluntad general de cooperar con distribuidores Linux con fines de lucro como Red-Hat, SUSE y Caldera demuestra que la mayoría de los hackers trabajarían felizmente para el mundo corporativo si éste sirve a sus fines.

Una razón tiene que ver con la simetría. Mientras que la mayoría de los desarrolladores open-source no objetan que otros ganen dinero con sus regalos, la mayoría también demanda que nadie (con la posible excepción del originador de una porción de código) esté en una posición privilegiada como para obtener ganancias. J. Random Hacker deja que Furbaco obtenga ganancias vendiendo su software o parches, pero sólo y mientras él también pueda hacer lo mismo.

Otra tiene que ver con consecuencias no intencionales. Los hackers han observado que las licencias que incluyen restricciones y coutas para uso "comercial" o venta (la forma mas común de tratar de recapturar el valor de venta, y que a primera vista parece razonable) tienen efectos desalentadores. Uno es el de colocar una sombra legal sobre actividades como la distribucion barata en CD-ROM, que idealmente tratamos de fomentar. Más generalmente, las restricciones sobre el uso, modificación y distribución (y otras complicaciones en las licencias) implican un costo adicional y (cuando aumenta el numero de paquetes con los que se trata) una explosión combinatoria de incertidumbre y riesgo legal potencial. Este resultado es considerado peligroso, por lo tanto hay una presión social para mantener las licencias lo mas simple posibles y libres de restricciones.

La razón final, y la más crítica tiene que ver con preservar la dinámica de revisión entre colegas y cultura del regalo descrita en [#!HtN!#]. Las restricciones en las

licencias diseñadas para proteger la propiedad intelectual o capturar l valor de venta directo suelen tener el efecto de hacer que la bifurcación de los proyectos sea legelmente imposible (este es el caso de, por ejemplo, la licencia llamada "Community Source" para Jini y Java, de Sun). Mientras que la bifurcación está desaprobada y considerada como un último recurso (por razones largamente discutidas en [#!HtN!#]), se considera críticamente importante que ese último reurso esté presente en caso de incompetencia de la persona que mantiene el proyecto, o el cambio de éste a una licencia más restrictiva.

La comunidad hacker tiene algo de tolerancia a la razón de simetría; por eso tolera licencias como NPL de Netscape, que otorgan priviliegios de ganancias a sus originadores (especificamente, en el caso de NPL, el derecho exclusivo a utilizar el código open-source de Mozilla en productos derivados, incluso closed-source). La comunidad tiene menos tolerancia a la razón de las consecuencias no intencionales, y ninguna tolerancia al hecho de preservar la opción a bifurcar (por esto es que los esquemas "Community License" de Java y Jini han sido largamente rechazados por la comunidad).

Estas razones explican las cláusulas de la Deficición de Open Source, la cual fue escrita para expresar el consenso de la comunidad hacker sobre las características críticas de las licencias standard (licencias : GPL, BSD, MIT y Artistic). Estas cláusulas tienen el efecto (aunque no la intención) de hacer que el valor de venta directa sea muy difícil de capturar.

Modelos de Valor de Venta Indirecto

Hay modos de crear mercados de servicios relacionados al software que capturen algo como el valor indirecto de venta. Hay cinco modelos conocidos y dos modelos especulativos (más podrían ser desarrollados en el futuro).

Posicionamiento en el Mercado

En este modelo, se usa software open-source para crear o mantener una posición de mercado para un software propietario que genera un flujo directo de ingresos. En la variante más común, el uso de software cliente open-source genera ventas de software servidor, o ingresos por subscripciones, o publicidad asociadas a un portal.

Netscape Communications Inc., estaba siguiendo esta estrategia cuando abrió las fuentes del browser Mozilla al principio de 1998. El lado del cliente de su negocio estaba al 13% y cayendo cuando Microsoft lanzó Internet Explorer. La intensidad de la campaña publicitaria sobre IE (y algunas oscuras prácticas de distribución que más tarde se convertirían en el tema central de un juicio antitrust) rápidamente se devorarían la participación de Netscape en el mercado de los navegadores, creando una conciencia que Microsoft trató de monopolizar el mercado de los browsers y luego usar un control de-facto sobre HTML para dejar a Netscape fuera del mercado de los servidores.

Abriendo las fuentes del todavía ampliamente popular navegador Netscape, Netscape le negó a Microsoft la posibilidad de un monopolio de browsers. Esperaban que la colaboración open-source aceleraría el desarrollo y depuración del browser, y esperaban que Internet Explorer estaría confinado a jugar un rol secundario, y prevendría que definieran exclusivamente HTML.

La estrategia funcionó. En noviembre de 1998 Netscape comenzó a re-ganar participación de mercado. Al tiempo que Netscape fue adquirido por AOL al principio de 1999, la ventaja competitiva de mantener Mozilla era suficientemente clara como para que uno de los primeros compromisos públicos de AOL fue continuar soportando el proyecto Mozilla, aunque todavía estaba en estado alfa.

Asegurar el Futuro

Este modelo es para fabricantes de hardware (hardware, en este contexto, incluye cualquier cosa desde un adaptador Ethernet o alguna placa periférica hasta un sistema completo de computación). Las presiones del mercado han forzado a las compañías de hardware a escribir y mantener software (desde device drivers, hasta herramientas de configuración, hasta el nivel de sistemas operativos completos), pero ese software no es un centro de ganancias. Es un costo - a menudo un costo sustancial.

En esta situación, abrir las fuentes es una opción clara. No hay ningúna ganancia que perder. Lo que el fabricante gana es una cantidad muy grande de desarrolladores, una respuesta más rápida y flexible a las necesidades del usuario, y una mayor confiabilidad a través de las revisiones. Probablemente también ganará una mayor lealtad de parte de sus clientes, al realizar las adecuaciones de software que ellos necesitan.

(Una objeción que suele presentarse a abrir el código de manejadores de hardware es que éstos pueden revelar datos importantes acerca de la operación del hardware que los competidores podrían copiar, ganando una ventaja competitiva injusta. En la época de los ciclos de productos de tres a cinco aõn este era un argumento válido. Hoy, el tiempo que los ingenieros de la competencia tardan en copiar y entender es una parte importante del ciclo de vida del producto que no están invirtiendo en investigación o diferenciación de su propio producto. El plagio es una trampa en la que todo fabricante quiere que sus competidores caigan.)

El efecto "a prueba de futuro" de open-source es particularmente fuerte en este caso. Los productos de hardware tienen una vida finita de producción y soporte; despues de eso, los clientes estan por su propia cuenta. Pero si tienen acceso al código de los drivers y pueden parcharlos de acuerdo a sus necesidades, habrá mas clientes felices que repetiran sus compras en la misma compañía. Un ejemplo muy dramático de adoptar este modelo fué la decisión de Apple Computer de abrir el código de "Darwin" , la base de su sistema operativo MacOSX, a mediados de marzo de 1999.

Regale la Receta, y Abra un Restaurant

En este modelo, uno abre el código del software para crear una posición de mercado, no para software cerrado (como en el modelo de Posicionamiento en el Mercado), sino para servicios.

Esto es lo que hacen Red Hat y otros distribuidores de Linux. Lo que realmente están vendiendo no es el software, sino el valor agregado por el ensamblaje y pruebas de un sistema operativo que está garantizado. Otros elementos de su proposición de valor includen soporte de instalación gratuito y la provisión de opciones para contratos de soporte continuo.

El efecto constructor de mercado de open-source puede ser extremadamente poderoso, especialmente para compañías que están inevitablemente en una posición de servicios. Un caso instructivo reciente es Digital Creations, una casa de diseño de sitios web que comenzó en 1998 y se especializa en bases de datos complejas y sitios transaccionales. Su mayor herramienta, la joya de la corona, es objeto de publicidad que ha pasado por varios nombres y encarnaciones pero ahora se llama Zope.

Cuando Digital Creations buscaba capital, los consultores que evaluaron cuidadosamente el nicho prospectivo de mercado, su gente y sus herramientas recomendaron que la empresa abra el código de Zope.

Desde el punto de vista de los standards de la industria del software, esta decisión se ve como una jugada absolutamente loca. La sabiduría convencional opina que la joya de la corona de la compañía no debe ser regalada bajo ninguna circunstancia. Pero los consultores tenían dos razones. Una es que la base de Aope refleja el cerebro y las habilidades de la gente de Digital Creations. La segunda es que Zope generaráa más valor como constructor de mercado que como herramienta secreta.

Para ver esto, compare estos dos escenarios. En el convencional, Zope permanece como el arma secreta de Digital Creations. Estipulemos que es un arma muy efectiva. Como resultado, la firma será capaz de despachar alta calidad en cortos tiempos - pero nadie lo sabe. Será facil satisfacer a los clientes, pero será dificil construir una base de clientes para comenzar.

En cambio, los consultores dijeron que abriendo el código de Zope sería publicidad crítica para Digital Creations. Se esperaba que los clientes que evaluaran Zope considerarían más eficiente contratar a los expertos que desarrollar experiencia Zope en casa.

Un directivo de Zope ha confirmado que la estrategia oppen-source ha "abierto muchas puertas que no se hubieran abierto de otra manera". Los clientes potenciales respondieron a la lógica de la situación - y Digital Creations está prosperando.

Otro ejemplo de último momento es e-smith (http://www.e-smith.net/) Esta compañía vende contratos de soporte para software servidor de Internet open-source,

una versión adaptada de Linux. Uno de los responsables, describiendo la cantidad de downloads libres del software de e-smith dijo: "La mayoría de las compañías considerarían a esto piratería de software; nosotros lo consideramos marketing libre".

Accesorizar

En este modelo, se venden accesorios para software open-source. En un bajo nivel, se venden tazas y remeras; en un alto nivel, documentación de calidad profesional.

O'Reilly Associates, publicadores de muchas excelentes volúmenes de referencia sobre software open-source, es un buen ejemplo de una compañía de accesorios. O'Reilly contrata y soporta a hackers conocidos de la cultura open-source (como Larry Wall y Brian Behlendorf) como un modo de construir su reputación en el mercado elegido.

Libere el Futuro, Venda el Presente

En este modelo, se distribuye software como binarios y fuentes con una licencia cerrada, pero que incluye una fecha de expiración en las previsiones de cierre. Por ejemplo, se podría escribir una licencia que permita la libre redistribución, prohiba el uso comercial sin una cuota, y garantiza que el software se regirá por licencia GPL un año despues de su fecha de lanzamiento, o si el productor desaparece.

Bajo este modelo, los clientes se aseguran que el producto es adaptable a sus necesidades, porque tienen las fuentes. El producto es a prueba de futuro - la licencia garantiza que una comunidad open-source puede retomar el producto si la compa~ía original muere.

Como el precio de venta y el volumen están basados en estas expectativas de los clientes, la compañía gozaraá de ingresos que no hubiese obtenido si hubiera distribuido su producto exclusivamente en forma cerrada. Más aún, mientras el viejo código el llevado a licencia GPL, éste recibirá una gran cantidad de revisión, arreglos, y características menores, lo cual quita una parte de ese 75% de mantenimiento por parte de sus creadores.

Este modelo ha sido seguido exitosamente por Aladdin Enterprises, los creadores del programa Ghostscript (un intérprete Postscript que puede traducir al lenguaje nativo de muchas impresoras).

La principal desventaja de este modelo es que su naturaleza cerrada tiende a inhibir la revisión y participación en las primeras fases del ciclo de vida, que es precisamente cuando más se las necesita.

Libere el Software, Venda la Marca

Este es un modelo de negocios especulativo. Se abre el código de una tecnología de software, se retiene una suite de pruebas o un conjunto de criterios de compatibilidad, y se le vende a los usuarios una marca certificando que su implementación de tecnología es compatible con todas las demás que lleven esa marca.

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

Libere el Software, Venda el Contenido

Este es otro modelo de negocios especulativo. Imagine algo parecido a un servicio de subscripción. El valor no está en el software cliente ni en el servidor, sino en proveer información confiable. Entonces se abre el código de todo el software, y se venden subscripciones al contenido. Cuando los hackers porten el cliente a nuevas plataformas y lo mejoren de diversas formas, el mercado automáticamente se expande.

(Por esto AOL debería abrir el código de su software cliente.)

Cuando Ser Abierto, Cuando Ser Cerrado

Habiendo hecho una revisión de los modelos de mercado que soportan el desarrollo open-source, ahora podemos aproximarnos a la pregunta general acerca de cuando tiene sentido (económicamente) usar un modelo abierto y cuando conviene ser cerrado. Primero debemos aclarar cuales son las ganancias de cada estrategia.

¿Cúales son las ganancias?

La aproximación cerrada permite recolectar rentas de sus secretos; desde otro punto de vista, cierra las puertas a la posibilidad de una verdadera revisión independiente. La aproximación abierta establece las condiciones para una revisión independiente, pero no deja tomar ganancias de los secretos.

La ganancia de tener secretos está bien entendida; tradicionalmente, los modelos de negocios de software se han construido a su alrededor. Hasta hace poco tiempo, la ganancia de las revisiones independientes no estaban bien entendidas. Sin embargo, el sistema operativo Linux nos da una lección que deberíamos haber aprendido hace varios años del software base de internet y de otras ramas de la ingeniería - que la revisión independiente tipo open-source es el único método escalable de obtener alta calidad y confiabilidad.

Entonces, en un mercado competitivo, los clientes que busquen alta confiabilidad y calidad recompensarán a los productores de software que se dirijan hacia implementar open-source y que descubran como mantener un flujo de entradas de dinero en los mercados de servicios, valor agregado, y soporte asociados al software. Este es el fenómeno detrás del sorprendente éxito de Linux, el cual vino desde la nada en 1996 hasta pasar el 17% del mercado de servidores al fin de 1998 , y parece que dominará el mercado dentro de dos años (al principio de 1999, IDC proyectó que Linux crecería más rápido que todos los otros sistemas operativos juntos hacia el 2003).

Una ganancia igualmente importante de abrir el código es su utilidad como un medio de propagar standards abiertos y construir mercados alrededor de ellos. El crecimiento dramático de internet se debe en gran parte a que nadie es el poseedor de TCP/IP; nadie ha cerrado propietariamente los protocolos base de internet.

Los efectos de red detrás del éxito de TCP/IP y Linux están bastante claros, y se reducen finalmente a dos elementos: confianza y simetría - las partes que conforman una infraestructura compartida pueden confiar (racionalmente) más si pueden ver como funciona, y van a preferir una infraestructura en la cual todas las partes tienen derechos simétricos a una en la cual una sola parte está en una posición privilegiada para extraer ganancias o ejercer el control.

Sin embargo, no es realmente necesario asumir efectos de red para lograr que la simetría sea un tema importante para los consumidores de software. Ningún consumidor de software elegiría racionalmente encerrarse en un monomolio controlado por el productor dependiendo de fuentes cerradas si existe una alternativa open-source de calidad aceptable. Este argumento toma más fuerza cuando el software se torna más crítico para el negocio del consumidor - mientras más vital es, menos tolerará el consumidor ser controlado por una organización externa.

Finalmente, una ganancia importante para el consumidor del software abierto relacionada con la confianza es que este tipo de software es a prueba de futuro. Si las fuentes son abiertas, el consumidor tiene algunos recursos si el productor desaparece. Esto puede ser particularmente importante para asuntos relacionados con el hardware, ya que éste tiende a tener ciclos de vida cortos, pero el efecto es más general y se traduce en un aumento de valor para el software de fuentes abiertas.

¿Cómo interactuan?

Cuando las ganancias de los secretos son más altas que las de abrir las fuentes, es económicamente sensato mantener las fuentes cerradas. Cuando los retornos de abrir las fuentes son más altos que las ganancias proporcionadas por los secretos, es sensato moverse hacia la apertura de las fuentes.

En si misma, esta es una observación trivial. Se convierte en no trivial cuando notamos que las ganancias provenientes de las fuentes abiertas son más difíciles de medir y predecir que la renta proveniente de los secretos - y que dicha ganancia es generalmente muy subestimada. De hecho, hasta que el mundo de los negocios comenzó a repensar sus premisas siguiendo la apretura del código de Mozilla en 1998, se asumía generalmente que las ganancias de open-source eran nulas.

Entonces cómo podemos evaluar las ganancias de open-source? En general es una pregunta difícil, pero podemos aproximarlas como lo haríamos con cualquier otro problema de predicción. Podemos comenzar desde casos donde la apertura de fuentes ha sido exitosa o ha fracasado. Podemos tratar de generalizar un modelo, el cual nos de por lo menos una impresión cualitativa de los contextos en los cuales open-source es una ganancia neta para el inversor o el negocio tratando de maximizar beneficios. Entonces podemos volver a los datos y tratar de refinar el modelo.

Del análisis presentado en [#!CatB!#], podemos esperar que abrir las fuentes tenga altas ganancias cuando (a) la confiabilidad/estabilidad/escalabilidad son críticas, y (b) la corrección del diseño y la implementación no puede ser efectivamente verificada por otros medios distintos a la revisión independiente. (El segundo criterio es cierto en la práctica para casi todos los programas no triviales.)

El deseo racional de un consumidor de evitar quedar atado a un productor monopólico incrementará su interés en la apertura de las fuentes (y, por lo tanto el valor competitivo de mercado de los productores que implementen open-source) cuando el software es más crítico para el consumidor. Entonces, otro criterio (c) empuja hacia la apertura de las fuentes cuando el software es un bien de capital crítico del negocio.

Para el área de las aplicaciones, observamos más arriba que la infraestructura open-source crea efectos de confianza y simetría que, a través del tiempo, tienden a atraer más consumidores; y frecuentemente es mejor tener una pequeña parte de ese mercado que se expande rápidamente, que una parte mayor de un mercado cerrado y estancado. De acuerdo a esto, y para software de infraestructura, es muy probable que una jugada de apertura de fuentes tenga una mayor ganancia a largo plazo que una jugada de fuentes cerradas (para lucrar con la propiedad intelectual).

De hecho, la capacidad de los potenciales consumidores de razonar acerca de las consecuancias futuras de las estrategias del productor y su rechazo a aceptar un

monopolio de suministradores implica una restricción más fuerte; sin tener un devastador poder de mercado, usted puede elegir o bien una jugada open-source o una jugada de ganancias directas de fuentes cerradas - pero no ambas. (Analogías a este principio pueden ser encontradas en cualquier otro campo, por ejemplo en el mercado electrónico, donde los clientes suelen rehusarse a comprar diseños que provengan de una sola fuente.) El caso puede ser expuesto menos negativamente: donde dominan los efectos de red, abrir las fuentes es la decisión correcta.

Podemos completar esta lógica observando que abrir las fuentes parece ser más exitoso generando mayores retornos que fuentes cerradas cuando se trata de software que (d) establece una infraestructura común de comunicaciónes y cómputos.

Finalmente, debemos notar que los productores de servicios únicos o altamente diferenciados tienen mucho más temor que sus métodos sean copiados por competidores que los productores de servicios para los cuales los algoritmos críticos y las bases de conocimiento están ampliamente entendidas. Por esto, la apertura de fuentes dominará cuando (e) los métodos clave (o equivalentes funcionales) forman parte de un conocimiento común de ingeniería.

El software base de internet, Apache, y la implementación de la API ANSI-standard de Unix por parte de Linux son los mejores ejemplos de los cinco criterios. El camino hacia las fuentes abiertas en la evolución de esos mercados está muy bien ilustrada por la reconvergencia de las redes de datos en TCP/IP a mediados de los 90s, después de quince años de intentos fallidos de construir imperios sobre protocolos cerrados como DECNET, XNS, IPX, etc.

Desde la otra cara de la moneda, la apertura de fuentes parece ser la última opción para compañías que tienen una posesión única de una tecnología de software generadora de valor (altamente cumpliendo con el criterio (a)) la cual es (a) relativamente poco sensible a las fallas, la cual puede ser (b) verificada por medios distintos a la revisión independiante, la cual no es (c) crítica para el negocio, y la cual no tendría su valor incrementado substancialmente por (d) efectos de red.

Como un ejemplo de este caso extremo, una compañía que escribe software que calcula patrones de corte para aserraderos que quieren maximizar los tablones extraidos de troncos me preguntó: "Deberíamos pasarnos a apertura de fuentes?". Mi conclusión fué "No". El único criterio al que parece acercarse es a (c); pero, si hay problemas, un operario experimentado podría generar los patrones de corte a mano.

Un punto importante es que el lugar donde encaja un determinado producto o tecnologí en estas escalas puede cambiar con el tiempo, como veremos en el caso de estudio siguiente.

En resumen, los siguientes items empujan hacia el desarrollo open-source:

(a) la confiabilidad/estabilidad/escalabilidad son críticas

(b) la corrección del diseño y la implementación sólo pueden ser verificadas por

medio de revisiones independientes entre colegas

(c) el software es crítico para el control del negocio por parte del consumidor

(d) el software establece una infraestructura común de comunicaciones y cómputos

(e) los métodos clave (o equivalentes funcionales de ellos) forman parte de conocimientos comunes de ingeniería.

Doom: Un Caso de Estudio

La historia del juego Doom de id software ilustra los modos en los cuales la presión del mercado y la evolución del producto pueden cambiar críticamente las magnitudes de ganancias entre fuentes abiertas y cerradas.

Cuando Doom fue lanzado a fines de 1993, su modo de animación de tiempo real, en primera persona lo hacían algo totalmente único (la antítesis del criterio (e)). No sólo era por el impacto visual de la técnica, sino que por muchos meses nadie podia darse cuenta como podía ser logrado en los microprocesadores de baja potencia de la época. Esos secretos valían una seria ganancia. Además, la potencial ganancia de abrir las fuentes eran bajas. Como un juego en solitario, el software (a) incurría en costos de fallas tolerablemente bajos, (b) no era terriblemente difícil de verificar, (c) no era crítico para el negocio de ningún cliente, (d) no se beneficiaba de los efectos de red. Era economicamente racional mantener a Doom en una estrategia de fuentes cerradas.

Sin embargo, el mercado alrededor de Doom no se mantenía quieto. Competidores inventaron equivalentes funcionales de sus técnicas de animación, y otros juegos de "pistolero en primera persona" (como Duke Nukem) comenzaron a aparecer. Cuando estos juegos se comenzaron a devorar la porción de mercado de Doom, el valor de ganancia de los secretos se desvaneció.

Además, los esfuerzos pasa expandir esa porción de mercado trajeron nuevos desafíos tecnológicos - mejor confiabilidad, mas características de juego, una mayor base de usuarios, y múltiples plataformas. Con el advenimiento de los partidos multijugador "deathmatch" y los servicios de juegos Doom, el mercado comenzó a desplegar ciertos efectos de red substanciales. Todo esto estaba demandando horas de programación que id habría preferido gastar en el próximo juego.

todas estas tendencias aumentaron las ganancias de abrir las fuentes. En algún punto las curvas de ganancias se cruzaron y fué economicamente razonable abrir el código de Doom y cambiar el rumbo hacia hacer dinero en negocios secundarios, como antologías de escenarios de juegos. Y en un tiempo después de ese punto, la trancisión ocurrió. El código completo de Doom fue abierto a fines de 1997.

Hay Que Saber Cuando Liberar

Doom es un caso interesante de estudio porque no se trata ni de un sistema operativo, ni de un software de comunicaciones o red; por lo tanto, está bastante lejos de los ejemplos comunes y obvios del éxito open-source. De hecho, el ciclo de vida de Doom, completo con punto de cruce de curvas, puede comenzar a tipificar la ecología de código del software de aplicaciones de hoy - una en la cual las comunicaciones y la computación distribuida crean serios problemas de robustez/confiabilidad/escalabilidad que sólo son accesibles a través de la revisión entre colegas, y frecuentemente cruzan los límites entre ambientes técnicos y entre actores que compiten (con todos los temas de confianza y simetría que implican).

Doom evolucionó desde un juego en solitario hacia partidos a muerte. Cada vez más, el efecto de red es la computación. Tendencias similares son visibles incluso en las aplicaciones de negocios más pesadas, como ERPs, y redes de negocios que cada vez interactúan más con suministradores y clientes - y, por supuesto, están implícitos en toda la arquitectura de la World Wide Web. Se deduce que casi en todos lados, las ganancias de open-source estan creciendo continuamente.

Si las tendencias actuales continuan, el desafío central de la tecnología de software y de la administración de productos será saber cuando liberar - cuando permitir que el código cerrado pase a infraestructura de código abierto, para explotar el efecto de revisión entre colegas y capturar altos retornos en servicios y otros mercados secundarios.

Existen incentivos de ingresos demasiado obvios como para dejar pasar el punto de cruce de curvas con mucho error en cualquier dirección. Detrás de eso, hay un riesgo de oportunidad muy serio en esperar demasiado - se podría quedar retrasado por un competidor que decida abrir las fuentes en el mismo nicho de mercado.

La razón para decir que este es un tema serio es que tanto el grupo de usuarios como el grupo de talentos disponibles para ser recrutados en cooperación open-source para una determinada categoría de producto son limitados. Si dos productores son el primero y segundo en abrir las fuentes de codigo que compite en una función relativamente parecida, el primero atraerá la mayor cantidad de usuarios y la mayor cantidad de codesarrolladores (y los más motivados); el segundo tomará lo que quede.

La Ecología de Negocios de Open Source

La comunidad open-source se ha organizado de un modo tal que tiende a amplificar los efectos productivos de abrir las fuentes. En el mundo Linux, en particular, es un hecho economicamente significante que existen múltiples distribuidores que compiten, los cuales formanuna instancia separada de los desarrolladores.

Los desarrolladores escriben código, y hacen que el código esté disponible en internet. Cada distribuidor selecciona un subconjunto del código disponible, lo integra, lo empaqueta y le pone un nombre, y lo vende a los clientes. Los usuarios eligen entre las distribuciones, y pueden complementar una distribución bajando código directamente de sitios de desarrolladores.

El efecto de esta separación de instancias es el de crear un mercado interno muy fluido para las mejoras. Los desarrolladores compiten entre ellos por la atención de los distribuidores y usuarios, con la calidad de su software. Los distribuidores compiten por los dólares de los usuarios con la adecuación de sus políticas de selección, y con el valor que le agregan al software.

Un efecto de primer órden de esta estructura interna de mercado es que ningún nodo de la red es indispensable. Los desarrolladores pueden retirarse; aunque su porción de código no sea tomado directamente por ningún otro desarrollador, la competencia por atención tenderá a generar alternativas funcionales rapidamente. Los distribuidores pueden fracasar sin dañar o comprometer el código base open-source común. La ecología como un todo tiene una respuesta más rápida a las demandas del mercado, y más capacidad de resistir choques y regenerarse a si misma, que la que cualquier productor monolítico de un sistema operativo pueda reunir.

Otro efecto importante es el de disminuir el costo e imcrementar la eficiencia a través de la especialización. Los desarrolladores no experimentan las presiones que rutinariamente forman parte de los proyectos cerrados convencionales - no hay listas de características sin sentido y distrayentes de Marketing, no hay órdenes de la Administración de usar lenguajes o ambientes de derrollo inapropiados, no hay requisitos de reinventar la rueda de un modo nuevo e incompatible en el nombre de la diferenciación del producto o de la protección de la propiedad intelectual, y (más importantemente) no hay fechas límite. No hay que apurar una versión 1.0, y sacarla al mercado antes que realmente esté completada - la cual (como observaron DeMarco y Lister en su discusión del estilo de administración "despiértenme cuando esté listo" en [#!DL!#]) generalmente conduce no sólo a una más alta calidad sino también a la más rápida entrega de un resultado que realmente funciona.

Los distribuidores, por otra parte, llegan a especializarse en las cosas que los distribuidores saben hacer mas efectivamente. Liberados de la necesidad de soportar

económicamente el desarrollo masivo para mantenerse competitivos, pueden concentrarse en la integración de sistemas, empaquetado, aseguración de calidad y servicio.

Los distribuidores y los desarrolladores son mantenidos al tanto ((honest)) por la constante realimentación y monitoreo por parte de los usuarios, quienes son una parte integral del método open-source.

Manejando el Éxito

La Tragedia del Campo puede no ser aplicable al desarrollo open-source como sucede hoy en día, pero esto no implica que no existan razones para preguntarse si el momento ((momentum)) actual de la comunidad open-source es sustentable. ¿Los jugadores clave desistirán de la cooperación cuando la recompensa se vuelva más alta?

Hay varios niveles en los cuales se puede hacer esta pregunta. Nuestra contra-historia de la "Comedia del Campo" está basada en que el valor de las contribuciones individuales open-source es difícil de monetizar. Pero este argumento tiene mucha menos fuerza para firmas (como, digamos, los distribuidores Linux) las cuales ya tienen un flujo monetario asociado con open-source. Su contribución está siendo monetizada todos los días. ¿Su rol cooperativo actual es estable?

Examinar esta pregunta nos llevará a algunas vistas interesantes sobre la economía del software open-source del mundo real de hoy - y acerca de lo que un verdadero paradigma de industria de servicios implica para la industria del software en el futuro.

En el nivel práctico, aplicada a la comunidad open-source tal como existe hoy, esta pregunta se divide de dos modos totalmente distintos. Uno: ¿Se fragmenterá Linux? Dos: ¿Linux se convertirá en un jugador dominante, casi monopólico?

La analogía histórica en la que mucha gente piensa cuando se sugiere que Linux se fragmentará es la del comportamiento de los productores de Unix propietario en los 80s. A pesar de interminables charlas sobre standards abiertos, a pesar de numerosas alianzas y consorcios, los Unix propietarios se dividieron. El deseo de los productores de diferenciar sus productos agregando y modificando características del sistema operativo fueron más fuertes que sus intereses en hacer crecer el tamaño total del mercado Unix manteniendo la compatibilidad (y consecuentemente bajando las barreras de entrada para desarrolladores de software independientes, y bajando el costo total de posesión para los clientes).

Esto es muy difícil que le suceda a Linux, por la simple razón que todos los distribuidores están restringidos a operar dentro de una base común de código fuente. No es realmente posible para ninguno de ellos mantener una diferenciación, oprque las licencias bajo las cuales el código Linux se desarrolla requieren que compartan el código con todas las partes. Desde el momento en el cual un distribuidor desarrolla una característica, todos los competidores pueden clonarla.

Como todas las partes entienden esto, nadie ni siquiera piensa en hacer el tipo de maniobras que fragmentaron a los Unix propietarios. En cambio, los distribuidores Linux son forzados a competir de maneras que realmente benefician al consumidor y al mercado global. Esto es que, deben competir en el servicio, soporte, y sus apuestas

de diseño en lo que a interfaces concierne para conducir a una mayor facilidad de uso e instalación.

La fuente común también cierra las puertas a la monopolización. Cuando la gente de Linux comienza a preocuparse por esto, el nombre que usualmente se murmura es "Red Hat". el del más grande y más exitoso de los distribuidores (con una participación estimada en alrededor del 90% en Estados Unidos). Pero es notable que en días después de anuncio de la largamente esperada versión 6.0 - antes que los CD-ROMs se enviaran a destino en cantidad - comenzaron a ofrecerse imágenes del CD-ROM construidas a partir del sitio FTP de Red Hat por editores de libros y por otros distribuidores de CD-ROM a precios menores que los de lista esperados por Red Hat.

A Red Hat no se le movió un pelo por esto, porque sus fundadores entienden muy claramente que ellos no poseen ni pueden poseer los pedazos de su producto; las normas sociales de la comunidad Linux lo prohiben. Si Red Hat hubiese protestado por clonarle el pre-lanzamiento de su nuevo producto, habría comprometido seriamente su capacidad para contar con cooperación futura de parte de la comunidad de desarrolladores.

Quizás más importante en este tiempo, las licencias de software que expresan estas normas de la comunidad con cierta forma legal prohiben activamente que Red Hat monopolice el código fuente en el cual está basado su producto. Lo único que puede vender es una relación de marca/servicio/soporte con gente que desea pagar por ella. Este no es un contexto en el cual la posibilidad de un monopolio depredador dure mucho.

Investigación y Desarrollo Abiertos, y la Reinvención del Mecenazgo

Hay otro aspecto en el cual la inserción de dinero real dentro del mundo open-source lo está cambiando. Las estrellas de la comunidad estan comenzando a darse cuenta que pueden recibir dinero por hacer lo que quieren hacer, en lugar de seguir el movimiento open-source como un hobby pero financiados por otro trabajo de día. Corporaciones como Red Hat, O'Reilly Associates, y VA Linux Systems están construyendo departamentos de investigación semi independiente contratando y manteniendo estables a talentos del mundo open-source.

Esto tiene sentido económico sólo si el costo por cabeza de mantener semejante laboratorio puede ser facilmente pagado con las ganancias esperadas por hacer crecer el mercado de la firma más rapidamente. O'Reilly puede pagarles a los principales autores de Perl y Apache por hacer sus trabajos porque espera que sus esfuerzos harán posible una mayor venta de libros relacionados con Apache y Perl. VA Linux Systems puede sostener su rama de laboratorio porque mejorar Linux mejora el valor de uso de las estaciones de trabajo y de los servidores que vende. Y Red Hat sostiene Red Hat Advanced Development Labs para incrementar el valor de su oferta Linux y para atraer mas clientes.

Para los estrategas de los sectores más tradicionales de la industria del software, crecidos ed culturas que tratan a las patentes- o a los secretos de propiedad intelectual como las joyas de la corona de la corporación, este comportamiento puede parecer inexplicable. ¿Por qué financiar desarrollo que cualquiera de sus competidores puede (por definición) apropiarse sin ningún costo?

Parece que hay dos razones importantes. Una es que mientra estas compañías se mantengan como jugadores dominantes en su nicho de mercado, pueden esperar capturar una gran proporción de participación en el mercado como resultado de su Investigación y Desarrollo abiertos. Usar Investigación y Desarrollo para comprar beneficios futuros no es una idea nueva; lo que es interesante es el cálculo implícito que las ganancias futuras esperadas son lo suficientemente grandes como para que estas compañías toleren la copia.

Mientras que esta análisis obvio de valor futuro esperado es necesario en un mundo de capitalistas, no es realmente el modo mas interesante de explicación para el hecho de contratar estrellas. Las empresas dirán que simplemente están haciendo las cosas bien por la comunidad de la cual vienen. Su humilde autor está suficientemente provisto de información de las tres firmas citadas anteriormente como para testificar que estas declaraciones no deben ser tomadas como falsedades. De hecho, yo fuí personalmente contratado por VA Linux Systems a fines de 1998 para aconsejarles

acerca de "hacer las cosas bien" , y han sido muy voluntariosos a la hora de escucharme.

Un economista preguntaría que ganancia hay aquí. Si aceptamos que la charla de hacer las cosas bien no es una postura vacía, deberíamos preguntarnos a que auto-interés de la firma sirve el "hacer las cosas bien". Como con el comportamiento superficialmente altruista en otras industrias, lo que estas firmas creen realmete es que están comprando buena voluntad.

Trabajar para ganar buena voluntad, y valuarlo como un bien predictivo de ganancias futuras de mercado, tampoco es nuevo. Lo que es interesante es la extrema alta valuación que el comportamiento de estas empresas sugiere que otorgan en esa buena voluntad. Y hasta ahora, el mercado ha recompensado este comportamiento.

Los directivos de estas compañías tienen bastante claras las razones por las cuales la buena voluntad es tan valiosa para ellos. Ellos confían mucho en voluntarios entre sus consumidores como desarrolladores de productos y como arma informal de marketing. Las relaciónes con sus clientes son íntimas, a menudo confían en lazos de confianza personal entre individuos fuera y dentro de la compañía.

Estas observaciones refuerzan una lección que aprendimos antes sobre las diferentes formas de razonar. La relación entre Red Hat/VA/O'Reilly y sus clientes/desarrolladores no es la típica de las firmas de manufactura. Todo lo contrario, llevan a patrones característicos de industrias de servicios basadas intensivamente en el conocimiento. Mirando fuera de la industria de la tecnología, podemos encontrar estos patrones en (por ejemplo) empresas de derecho, de medicina, y en universidades.

Podemos observar, de hecho, que las firmas open-source contratan hackers estrellas por la misma razón por la cual las universidades contratan a académicos estrellas. En ambos casos, la práctica es similar en su mecanismo y en su efecto al sistema de mecenazgo aristocrático, que financiaba la mayoría del arte fino hasta despues de la Revolución Industrial - una similaridad tal, que algunas partes ya se han dado cuenta.

Llegando Desde Aquí Hacia Allá

Los mecanismos de mercado para financiar (y para obtener ganancias de!) el desarrollo open-source todavía están evolucionando rapidamente. Los modelos de negocios que hemoss revisad en este artículo probablemente no serán los últimos en ser inventados. Los inversores todavía están pensando las consecuencias de reinventar la industria del software como una industria con un foco explícito en el servicio, no tanto en la propiedad intelectual cerrada.

Esta revolución conceptual tendrá algún costo (en beneficios no percibidos) para la gente que invierta en el 5% de la industria del valor de venta; historicamente, los negocios de servicios no son tan lucrativos como los negocios de manufactura (pero, como algún médico o abogado podrá decir, el retorno de los consumidores es a menudo alto). Cualquier beneficio no percibido, será más que igualado por beneficios en el lado de los costos, cuando los consumidores cosechen grandes ahorros y eficiencia de los productos open-source. (Hay un paralelo aquí con los efectos que está teniendo el reemplazo de la red tradicional de teléfonos por internet.)

La promesa de estos ahorros y eficiencia estácreando una oportunidad de mercado a la cual los capitalistas se están volcando para explotar. Cuando el primer borrador de este artículo estaba en preparación , la más prestigiosa firma de capital de Silicon Valley tomó una participación de liderazgo en la primera compañía que se especializará en soporte técnico 24/7 para Linux. Se espera generalmente que varias empresas relacionadas con Linux y con open-source aparecerán antes del fin de 1999, y que serán bastante exitosas.

Otro desarrollo muy interesante es el comienzo de intentos sistemáticos de realizar tareas de mercado en desarrollo open-source. SourceXchange (http://www.sourcexchange.com/process.html) y CoSource (http://www.cosource.com/) representan modos ligeramente distintos de tratar de aplicar un modelo de remate inverso para financiar desarrollo open-source.

Las grandes tendencias están claras. Mencionamos antes las proyecciones de IDC que Linux crecerá más rápido que todos los demás sistemas operativos juntos hacia el 2003. Apache está en el 61% de participación en el mercado y creciendo continuamente. El uso de internet está explotando, y las encuastas como el Contador de Sistemas Operativos de Internet (Internet Operating System Counter) muestran que Linux y otros sistemas operativos open-source ya son una pluralidad como hosts de internet, y están continuamente ganando mercado a los sistemas cerrados. La necesidad de explotar infraestructuras de internet open-source condicionan no solo el diseño de otro software, sino también las prácticas de mercado y los patrones de uso/obtención de uso de cada compañía. Y parece que estas tendencias están acelerándose.

Conclusión: La Vida Despues de la Revolución

Cómo se verá el mundo del software una vez que la trancisión hacia open-source se complete?

Para examinar la pregunta, sería conveniente clasificar el software de acuerdo al grado de completud que el servicio que ofrecen puede ser descrito por standards técnicos abiertos.

Este eje corresponde razonablemente bien con lo que la gente piensa cuando habla de "aplicaciones" (standards técnicos abiertos muy débiles o no existentes), "infraestructura" (standards abiertos fuertes), y "middleware" (standards técnicos efectivos pero incompletos). Los casos paradigmáticos hoy, en 1999, serían: un procesador de textos (aplicación), una pila TCP/IP (infraestructura), y un motor de base de datos (middleware).

El análisis de ganancias que hicimos anteriormente sugiere que aplicaciones, infraestructura y middleware serán transformados de diferentes maneras, y exhibirán distintas mezclas de fuentes abiertas y cerradas. Recordamos que sigerimos la prevelescencia de open-source en una determinada área de software sería una función de cuanto efecto substancial de red opera alii, cuales son los costos de las fallas, y a que extremo el software es un bien de capital crítico para el negocio.

Podemos aventirar algunas predicciones si aplicamos estas heurísticas, no a productos individuales, sino a segmentos enteros del mercado del software. Aquí vamos:

La Infraestructura (internet, la Web, los sistemas operativos, y los niveles bajos del software de comunicaciones que tiene que cruzar límites entre partes competidoras) será casi enteramente open-source , mantenido cooperativamnte por consorcios de usuarios y por empresas de distribución/servicio con fines de lucro, con un rol como el que hoy tiene Red Hat.

Las aplicaciones, por otra parte, tendrán la mayor tendencia a permanecer cerradas. Habrá circunstancias bajo las cuales el valor de uso de un algoritmo no revelado será suficientemente alto (y los costos asociados a la falta de confiabilidad serán suficientemente bajos, y los riesgos asociados con un suministrador monopólico serán lo suficientemente tolerables) como para que los consumidores continúen pagando por software cerrado. Esto permanecerá así en mercados verticales de aplicaciones solitarias, donde los efectos de red son débiles.

El middleware (como bases de datos, herramientas de desarrollo, o los top-ends adaptados de pilas de protocolos) estarán más mezclados. Esta categoría tenderá a ir haci al model cerrado o abierto de acuero al costo de las fallas, con un alto costo

creando presiones de mercado para una mayor apertura.

Para completar el cuadro, necesitamos notar que ni "aplicaciones" ni "middleware" son realmente categorías estables. En "Hay Que Saber Cuando Liberar" vimos que las tecnologías individuales de software pasan por un ciclo natural de vida desde racionalmente cerrado hacia racionalmente abierto. La misma lógica se aplica a este conjunto.

Las aplicaciones tienden a caer en el terreno de middleware cuando se utilizan técnicas standarizadas. (Las bases de datos, por ejemplo se volvieron middleware despues que SQL desacopló los clientes de los motores.) A su turno, cuando el middleware se convierta en algo más standard, caerá dentro de la categoría de infraestructura open-source - una transición que actualmente estamos viendo en sistemas operativos.

En un futuro que incluye la competencia de parte de open-source, podemos esperar que el destino eventual de cualquier tecnología de software será o bien morir o convertirse en parte de la infraestructura abierta. Mientras esto no es una noticia muy feliz para los empresarios que querrían recolectar rentas basadas en software cerrado para siempre, sugiere que la industria del software permanecerá empresarial, con nuevos nichos constantemente abriéndose en la parte alta (aplicaciones), y una corta extensión de vida para los monopolios cerrados a medida que sus productos caen dentro de la categoría de infraestructura.

Finalmente, por supuesto, este equilibrio será muy importante para el consumidor de software manejando el proceso. Más y más software de alta calidad se hará permanentemente disponible para usar en lugar de ser discontinuado o ser encerrado en la bóveda de alguien. El caldero mágico de Ceridwen es finalmente, una metáfora demasiado débil - porque los alimentos se consumen o perecen, mientras que el software vive (potencialmente) para siempre. El mercado libre, en su sentido de libertad más amplio incluyendo todas las actividades no cohersitivas sean de intercambio o de regalo, podrá producir riqueza de software para todos perpetuamente.