Por qué el Software debería ser libre

Introducción

La existencia de software provoca inevitablemente que nos preguntemos sobre qué decisiones concernientes a él deberían tomarse. Por ejemplo, supongamos una persona que teniendo una copia de un programa se encuentra con otra que desearía tener una copia. La posibilidad de copiar el programa existe; ¿quién debería decidir si esto se lleva a cabo o no? ?¿Las personas involucradas? ¿U otro sujeto, llamado ``dueño"?

Los desarrolladores de Software generalmente consideran estos problemas basándose en que el criterio para resolverlos es maximizar los beneficios del desarrollador. El poder político de la empresa ha llevado al gobierno a la adopción de este último criterio así como el propuesto por los desarrolladores: que el programa tiene un dueño, generalmente una compañía asociada a su desarrollo.

Me gustaría considerar el mismo problema pero usando un criterio diferente: la prosperidad y la libertad del público en general.

La respuesta no puede provenir de la ley vigente --la ley debería amoldarse a la ética y no al revés. Tampoco el día a día resuelve este problema, a pesar de que puede sugerir algunas soluciones posibles. La única forma de juzgar es viendo quién se ve ayudado y quién se ve perjudicado mediante el reconocimiento de dueños de software, por qué, y cuánto. En otras palabras, deberíamos realizar un análisis del tipo costo-beneficio en nombre de la sociedad como un todo, teniendo en cuenta la libertad individual así como la producción de bienes materiales.

En este ensayo, describiré los efectos provocados por el hecho de tener dueños, y mostraré que los resultados son perjudiciales. Mi conclusión es que los programadores debemos dedicarnos a animar a otros a compartir, redistribuir, estudiar y mejorar el software que escribimos: en otras palabras, escribir software ``libre". (1)

Cómo los Dueños Justifican Su Poder

Aquellos que se benefician del sistema actual en donde los programas se entienden como propiedad esgrimen dos argumentos en favor de su derecho de ser dueños de programas: el argumento emocional y el argumento económico.

El argumento emocional es del tipo: ``Pongo mi cariño, mi corazón, mi alma en este programa. Proviene de *mí*, es *mío*!"

Este argumento no necesita de una refutación seria. El sentimiento de cercanía es uno que los programadores pueden cultivar cuando les viene bien; no es inevitable. Considérese, por ejemplo, cuán deseosos esos mismos programadores firman y ceden sus derechos sobre el programa a una gran compañía a cambio de recibir un salario; el apego emocional se desvanece misteriosamente. Por el contrario, considérense a los grandes artistas y artesanos de la época Medieval, que ni siquiera firmaban sus trabajos. Para ellos, el nombre del artista no era importante. Lo que importaba era que el trabajo se había hecho --y el propósito al que servía. Esta visión prevaleció durante cientos de años.

El argumento económico es del tipo: ``Quiero ser rico (normalmente expresado de manera poco precisa como `vivir de algo'), y si no me dejas llegar a rico programando, entonces no programaré. Todo el mundo es como yo, de manera que nadie programará jamás. ¡Y te encontrarás conque no tienes programas!" Esta amenaza suele estar disfrazada de 'consejo de amigo que viene de un sabio'.

Explicaré más tarde por qué esta amenaza es algo completamente absurdo. Antes me gustaría presentar una suposición implícita que es más evidente en otra formulación del mismo argumento.

Esta formulación empieza comparando la utilidad social del software propietario con la utilidad sin ese software, y entonces llega a la conclusión de que el software propietario es, en general, beneficioso, y debería ser promovido. La falacia aquí se encuentra en comparar solamente dos posibilidades --software propietario vs. ausencia de software-- y suponiendo que no existen otras posibilidades.

Dado un sistema en el que impera la propiedad intelectual, el desarrollo del software se encuentra generalmente vinculado a la existencia de un dueño que controla el uso de ese software. Mientras existe este vínculo, estamos continuamente frente a la elección entre software propietario o nada. Sin embargo, esta unión no es ni inherente ni inevitable; es más bien una consecuencia de la decisión sociolegal específica sobre lo que estamos considerando: La decisión de tener dueños. Formular la elección entre software propietario y ausencia de software está pidiendo a gritos este planteamiento.

El Argumento en contra de Tener Dueños

La pregunta que se nos plantea es, "¿Debería el software estar vinculado a la existencia de dueños para, de esa manera, restringir su uso?"

Para poder resolver este problema, tenemos que juzgar el efecto en la sociedad de cada de las dos opciones

independientemente: el efecto de desarrollar el software (sin tener en cuenta la manera en que se redistribuye), y el efecto de restringir su uso (suponiendo que el software ha sido desarrollado). Si una de estas actividades es beneficiosa y la otra es perjudicial, deberíamos deshacernos de la doble actividad y usar sólo la beneficiosa.
En otras palabras, si restringir la distribución de un programa ya desarrollado es perjudicial para la sociedad en su conjunto, entonces un desarrollador de software que se considere ético debería rechazar esta opción.
Para determinar el efecto de restringir el poder compartir, necesitamos comparar el valor, para la sociedad, de un programa restringido (propietario) con ese mismo programa, pero accesible a todo el mundo. Esto nos lleva a comparar dos mundos posibles.
Este análisis también tiene en cuenta el, a veces defendido, contra-argumento de que ``el beneficio que se le proporciona al vecino al recibir una copia de un programa se cancela con el perjuicio provocado al dueño." Este contra-argumento presupone que el perjuicio y el beneficio son iguales en magnitud. El análisis llevado a cabo tiene en cuenta el comparar estas magnitudes, y el resultado muestra que el beneficio es mucho mayor que el perjuicio.
Para clarificar todo esto, vamos a aplicarlo a otra área: la construcción de carreteras.
La financiación para construir todas las carreteras podría provenir de peajes. Como consecuencia nos encontraríamos puntos de peaje en cada esquina. Un sistema de este tipo generaría incentivo a la hora de mejorar las carreteras. También tendría la virtud de causar que los usuarios de una determinada carretera pagasen por ella. Sin embargo, un punto de peaje es un obstáculo artificial para una conducción sin cortes -- artificial, porque no es una consecuencia derivada de cómo los coches o las carreteras funcionan.
Si comparamos carreteras libres y carreteras con peaje por su utilidad, encontramos que (siendo iguales), las carreteras sin puntos de peaje son más baratas de construir, más baratas para administrar y más eficientes.(2)
En un país pobre, el peaje podría provocar que algunas carreteras estuviesen inaccesibles a muchos ciudadanos. De manera que las carreteras sin puntos de peaje ofrecen mayor beneficio a la sociedad a menor costo; son preferibles por la sociedad. Luego la sociedad debería elegir financiar las carreteras de otro modo, no mediante puntos de peaje. El uso de las carreteras, una vez construidas, debería ser libre [free].
Cuando los defensores de los puntos de peaje los presentan como *simples* recaudadores de fondos, distorsionan la elección que de verdad existe. Los puntos de peaje incrementan los presupuestos, pero hacen algo además de eso: De hecho, degradan la carretera. La carretera con peajes no es tan buena como la carretera libre; el hecho de que se nos de más carreteras o carreteras técnicamente superiores puede muy bien no ser una mejora si ello implica sustituir carreteras libres por carreteras de peaje.
Por supuesto, la construcción de una carretera gratuita cuesta dinero, que de alguna manera la gente tiene que pagar. Sin embargo, esto no implica la inevitabilidad de los puntos de peaje. Nosotros, que en ambos casos pagamos, sacaremos mayor beneficio de nuestro dinero si compramos una carretera gratuita.
No es estoy queriendo decir que una carretera con peaje sea peor que no tener carreteras. Eso sería verdad si el peaje fuese tan grande que casi nadie pudiese usarla --pero no es esta la intención para un recaudador de impuestos. Sin embargo, debido a que los puntos de peaje causan pérdida de tiempo y molestia considerables, es mejor conseguir el dinero de una manera menos obstaculizadora.
Para aplicar este mismo argumento al desarrollo del software, mostraré ahora que el tener "puntos de peaje" en programas útiles le cuesta a la sociedad una barbaridad: provoca que los programas sean más caros a la hora de construirlos, más caros para distribuir, y menos satisfactorios y eficientes al usarlos. Se seguirá que la construcción de programas debería ser promovida de alguna otra forma. Más tarde, continuaré explicando otros métodos que promuevan y (hasta donde sea de verdad necesario) financien el desarrollo de software.

El Perjuicio Ocasionado por Obstaculizar el Software

Considérese por un momento que un programa ha sido desarrollado, y que cualesquiera pagos necesarios para su desarrollo se llevaron a cabo; ahora la sociedad debe decidir entre convertirlo en propietario o dejar que se use y comparta libremente. Supóngase que la existencia del programa y su disponibilidad se desean. (3)
Las restricciones sobre la distribución y modificación del programa no pueden facilitar su uso. Sólo pueden interferir con él. Así que el efecto solamente puede ser negativo. ¿Pero cuánto? ¿Y de qué tipo?
Existen tres niveles diferentes de daño efectivo que provienen de esta interferencia:

- Un menor número de personas usa el programa.
- Ninguno de los usuarios puede adaptar o arreglar el programa.
- Otros desarrolladores no pueden aprender del programa, o basar un trabajo nuevo en él.

Cada nivel de perjuicio efectivo lleva asociado un perjuicio psicológico. Me refiero al efecto que las

decisiones de la gente tiene en sus sentimientos, actitudes y predisposiciones posteriores. Estos cambios en la manera de pensar de la gente tendrán un efecto posterior en sus relaciones con sus vecinos, y pueden acarrear consecuencias efectivas.
Los tres niveles de perjuicio efectivo desaprovechan parte del valor que el programa podría proporcionar, pero no lo pueden reducir a cero. Si desaprovechan casi todo el valor del programa, entonces el hecho de escribir el programa perjudica a la sociedad en tanto se dedicó esfuerzo en escribir el programa. Se podría decir que aquel programa que produce beneficios al venderse debe proporcionar algún tipo de beneficio material directo.
Sin embargo, teniendo en cuenta el perjuicio psicológico asociado, no existe límite al perjuicio que el desarrollo de software propietario puede llegar a ocasionar.

Obstaculizar el Uso de Programas

El primer nivel de perjuicio impide el simple uso del programa. Una copia del programa tiene un costo marginal nulo (y se puede pagar este costo realizando esta copia personalmente), de manera que en un mercado libre, el programa tendría un precio casi nulo. El pago por una licencia es un desincentivo significativo a la hora de usar el programa. Si un programa de gran utilidad es propietario, mayor será la cantidad de gente que no lo use.
Es fácil mostrar que la contribución total que un programa proporciona a la sociedad se reduce al asignársele un dueño. Cada usuario potencial del programa, enfrentado al hecho de tener que pagar para usarlo, puede escoger entre pagar o renunciar a usar el programa. Cuando un usuario escoge pagar, esto es en realidad una transferencia nula de riqueza entre las dos partes. Pero cada vez que alguien elige no usar el programa, se provoca un perjuicio a esa persona sin que nadie salga beneficiada. La suma entre números negativos y ceros es siempre negativa.
Pero esto no reduce la cantidad de trabajo que lleva el *desarrollar* el programa. Como resultado, la eficiencia del proceso entero, medida en satisfacción del usuario final por hora de trabajo, se reduce.
Esto muestra la diferencia crucial entre copias de programas y coches, sillas o sandwiches. No existe una copiadora de objetos materiales fuera de la ciencia ficción. Pero los programas son fáciles de copiar; cualquiera puede producir tantas copias como desee, con muy poco esfuerzo. Esto no es cierto para objetos materiales porque la materia se conserva: cada copia nueva tiene que generarse con la misma materia prima de la misma forma que la primera copia que se construyó.
Con objetos materiales, un desincentivo a la hora de usarlos tiene cierto sentido, porque un menor número de objetos comprados implica menos materia prima y menos trabajo para producirlos. Es cierto que generalmente existe un costo inicial, un costo de desarrollo, que se extiende sobre el proceso de producción. Pero mientras el costo marginal de producción puede ser significativo, añadir una COMPARTICIÓN en el costo de desarrollo no produce una diferencia cualitativa. Y no requiere restricciones sobre la libertad de los usuarios normales.
Sin embargo, imponer un precio en algo que, de otra manera, podría ser gratuito, es un cambio cualitativo. Un pago impuesto unilateralmente sobre la distribución del software provoca un gran desincentivo.
Lo que es más, la producción centralizada como se practica en nuestros días es ineficiente incluso en términos de hacer llegar las copias del software. Este sistema incluye enviar discos o cintas magnéticas en embalajes superfluos, mandar grandes cantidades de ellos a lo largo del mundo, y almacenarlos para venderlos. Este costo se presenta como derivado del hacer negocios; en realidad, es una parte del gasto inútil causado por el hecho de tener dueños.

Perjuicio sobre la Cohesión Social

Suponga que tanto usted como su vecino vieran útil el correr un cierto programa. En un pacto ético con su vecino, entenderían que un uso apropiado de la situación podría hacer posible que los dos usasen ese programa. Una propuesta tal que sólo a uno se le permitiese usar el programa, restringiendo al otro, es divisora; ninguno de los dos, usted o su vecino, la encontraría aceptable.
El hecho de firmar una licencia típica de software implica traicionar a su vecino: ``Prometo privar a mi vecino de este programa de forma que yo sea capaz de tener una copia para mí." Las personas que toman estas decisiones sienten presión psicológica interior que los justifica, a cambio de degradar la importancia de ayudar a su vecino-- así que el espíritu público sale perjudicado. Se trata de un daño psicosocial asociado con el daño material provocado por el desincentivo de usar el programa.
Muchos usuarios inconscientemente admiten lo érroneo de la negativa a compartir, así que deciden ignorar las licencias y las leyes, y comparten el programa de todas formas. Pero a menudo se sienten culpables haciendo eso. Saben que deben infringir las leyes para poder ser buenos vecinos, pero siguen considerando

las leyes autorativas, y concluyen que el ser un buen vecino (que lo son) es malo o de lo que sentirse avergonzado. También se trata de un tipo de daño psicosocial, pero uno puede escapar de ello decidiendo que las licencias y las leyes no tiene suficiente fuerza moral.
Los programadores también sufren ese daño psicosocial al saber que a muchos usuarios se les impedirá usar su trabajo. Esto conduce a una actitud de cinismo o negativa. Un programador puede describir de manera entusiasta un trabajo que el considera técnicamente excitante; y cuando se le pregunta: ``¿Se me dejará usar el programa?", se vuelve cabizbajo y admite que la respuesta es no. Para poder no sentirse desalentado, o bien ignora este hecho la mayor parte del tiempo o adopta una postura cínica pensada para minimizar su importancia.
Desde la era de Reagan, la mayor escasez en los Estados Unidos no es innovación técnica sino más bien el deseo de trabajar juntos por el bien público. No tiene sentido alentar lo anterior a expensas de lo primero.

Obstruir la Adaptación Propia de Programas

El segundo nivel de perjuicio material es la imposibilidad para adaptar los programas. La posibilidad de modificar el software es una de las grandes ventajas frente la tecnología más antigua. Sin embargo, la mayoría del software comercial disponible no lo está en términos de modificabilidad, ni siquiera después de comprarlo. Puedes decidir cogerlo o no cogerlo, como una caja negra --eso es todo.
Un programa que usted ejecute consiste en una serie de números cuyo significado es oscuro. Nadie, ni siquiera un buen programador, puede cambiar fácilmente esos números para hacer que el programa haga algo diferente.
Los programadores trabajan normalmente con el ``código fuente" del programa, que se encuentra escrito en un lenguaje de programación como Fortran o C. Usa nombres que designan a los datos que se usan y a las partes del programa, y representa operaciones con símbolos tales como `+' para la suma y `-' para la resta. Está diseñado para ayudar a los programadores a leer y modificar los programas. He aquí un ejemplo; un programa que calcula la distancia entre dos puntos en un plano:

```
float
distance (p0, p1)
    struct point p0, p1;
{
  float xdist = p1.x - p0.x;
  float ydist = p1.y - p0.y;
  return sqrt (xdist * xdist + ydist * ydist);
}
```

Aquí está ese mismo programa en formato ejecutable, en el ordenador que suelo utilizar:

```
1314258944    -232267772    -231844864     1634862
1411907592    -231844736     2159150      1420296208
-234880989    -234879837    -234879966    -232295424
1644167167     -3214848     1090581031    1962942495
572518958     -803143692    1314803317
```

El código fuente es útil (al menos potencialmente) para cualquier usuario de un programa. Pero a la mayoría de los usuarios no se les deja tener copias del código fuente. Generalmente el código fuente de un programa propietario se guarda en secreto por el dueño, por miedo a que cualquier otro pueda aprender algo de él. Los usuarios reciben solamente los ficheros de números incomprensibles que el ordenador se encargará de ejecutar. Esto quiere decir que sólo el dueño del programa puede cambiar el programa.
Una amiga me habló una vez sobre el hecho de trabajar como programador en un banco por unos seis meses, escribiendo un programa similar a algo que se podía obtener comercialmente. Pensaba que si hubiese tenido acceso al código fuente de ese programa comercial lo podría haber adaptado fácilmente a las necesidades del banco. El banco esta dispuesto a pagar por ello, pero no le estaba permitido hacerlo-- el código fuente era un secreto. De manera que tuvo que hacer seis meses de trabajo de construcción, trabajo que aparece en el GNP pero que realmente fue desperdiciado.
El laboratorio de Inteligencia Artificial del MIT (AI lab) recibió, como regalo, una impresora gráfica de Xerox alrededor de 1977. Corría bajo software libre al que añadimos bastantes mejoras útiles. Por ejemplo, el software notificaba inmediatamente al usuario cuando el trabajo de imprimir se había realizado. Cuando la impresora tenía un problema, como una abstrucción de papel o falta de papel, el software notificaba inmediatamente a todos los usuarios que tuviesen trabajos pendientes. Estás mejoras facilitaban un trabajo más suave [smooth].

Más tarde Xerox donó al Laboratorio de IA una impresora nueva, más rápida, una de las primeras impresoras láser. Funcionaba con software propietario que corría en un ordenador dedicado en exclusiva y separado, de manera que no pudimos añadir ninguna de nuestras mejoras favoritas. Pudimos hacer que mandase una notificación cuando un trabajo de impresión había sido mandado al ordenador dedicado a la impresora, pero no cuando el trabajo había sido impreso (y generalmente el retraso era considerable). No había forma de saber cuándo el trabajo había sido impreso; lo único que podías hacer era adivinarlo. Y nadie sabía nunca cuando había un atasco de papel, así que la impresora se quedaba a menudo sin arreglar por espacio de una hora.

Los programadores de sistema del laboratorio de IA estaban capacitados para arreglar aquellos problemas, probablemente tan capacitados como los autores originales del programa. Xerox no mostró interés en arreglar aquellos fallos y eligió advertirnos de ellos, de manera que nos vimos forzados a aceptar los problemas. Nunca se arreglaron.

La mayoría de los buenos programdores han experimentado esta frustración. El banco podía permitirse resolver un problema escribiendo un programa nuevo partiendo de cero, pero un usuario típico, no importa la cualificación, sólo puede arrojar la toalla.

Arrojar la toalla provoca un daño psicosocial --al espíritu de independencia. Es desmoralizante vivir en una casa que no puedes arreglar para adecuarla a tus necesidades. Conduce a la resignación y al retraimiento, que pueden extenderse a otros aspectos de la vida de uno mismo. La gente que siente de esta manera no se encuentran a gusto y no realizan un buen trabajo.

Imagínese cómo sería si las recetas de cocina se guardasen de la misma manera que el software. Uno se podría preguntar, ``¿Cómo cambio esta receta de manera que no tenga sal?", y el gran chef respondiese, ``¡Cómo se atreve a insultar mi receta, mi creación y mi paladar, manoseándola? No tiene usted el juicio necesario para cambiar mi receta y hacer que salga bien!"

``¡Pero mi doctor me ha prohibido tomar sal! ¿Qué puedo hacer? ¿Va a quitar usted la sal por mí?"

``Me encantaría hacer eso; mis honorarios son sólo 50.000 dólares." Las tasas suelen ser grandes debido a la posición de monopolio sobre los cambios. ``De todas formas, ahora mismo no tengo tiempo. Estoy ocupado con una comisión para diseñar una nueva receta de galleta de barco para el departamento de Marina. Estaré contigo más o menos en dos años."

Impedir el Desarrollo del Software

El tercer nivel de daño material afecta al desarrollo de software. El desarrollo del software solía ser un proceso de evolución, en donde una persona cogía un programa existente y reescribía algunas partes de él para alguna función nueva, y entonces otra persona reescribía algunas partes de él para añadir alguna función nueva más; en algunos casos, esto continuaba durante un periodo de veinte años. Mientras tanto, algunas partes de ese programa eran ``canibalizadas" para formar el inicio de otros programas.

La existencia de dueños impide este tipo de evolución, haciendo que sea necesario empezar desde cero cuando se quiere desarrollar un programa. También impide a los aprendices estudiar los programas existentes que les enseñen técnicas útiles o incluso cómo están estructurados los programas grandes.

Los dueños también dificultan la educación. He conocido estudiantes brillantes en ciencia de computadores que nunca han visto el código fuente de un programa extenso.. Puede que fueran buenos escribiendo pequeños programas, pero no pueden empezar a aprender las diferentes habilidades para escribir programas extensos si no pueden ver como lo han hecho otros.

En cualquier campo intelectual, uno puede conseguir posiciones más elevadas sosteniéndose en los hombros de otros. Pero esto no está permitido generalmente en el campo del software --tu sólo puedes alzarte sobre los hombros de otras personas *en tu empresa*.

El daño psicosocial asociado afecta al espíritu de cooperación científica, que solía ser tan intensa que los científicos seguían cooperando incluso cuando sus países entraban en guerra. Siguiendo este espíritu, los oceanógrafos japoneses que abandonaban su laboratorio en una isla del Pacífico presrvaron cuidadosamente su trabajo para la invasión de los marines de los EE.UU. y dejaron una nota pidiendo que lo guardaran bien.

El conflicto por el beneficio ha ocupado lo que le quedaba de conflicto internacional. Hoy en día los científicos de bastantes campos no publican lo suficiente en sus trabajos que puedan permitir a otros repetir el experimento. Publican solamente aquello que permita a los lectores maravillarse por lo mucho que saben hacer. Esto es así, desde luego, en ciencia de computación, en donde el código fuente de los programas es generalmente secreto.

No Importa el Cómo se Restringe el Compartir

He estado hablando sobre los efectos de no dejar a la gente copiar, cambiar o basarse en un programa. No he

especificado el cómo esta obstrucción se lleva a cabo, porque no afecta a la conclusión. Como quiera que se haga, protección contra copia, o copyright, o licencias, o encriptación, o tarjetas ROM, o números de serie en el hardware, si tiene *éxito* impidiendo el uso, perjudica.

Los usuarios consideran algunos de estos métodos más repugnantes que otros. Creo que los métodos más odiados son aquellos que cumplen su objetivo.

El Software Debería ser Libre

He argumentado cómo la pertenencia de un programa-- el poder de restringir los cambios o las copias-- es obstruyente. Sus efectos negativos son amplios e importantes. Se sigue pues que en la sociedad no debería haber dueños de programas.

Otra manera de entender esto es que lo que la sociedad necesita es software libre, y el software propietario es un sustituto pobre. Promover el sustituto no es una manera lógica de conseguir lo que necesitamos.

Vaclav Havel nos aconsejó ``Trabajar por algo porque es bueno, no simplemente porque tiene probabilidades de ser un éxito." Un negocio que produce software propietario tiene probabilidades de éxito en sus propios y estrechos términos, pero no es lo que beneficia a la sociedad.

El por qué la Gente Desarrollará Software

Si eliminamos la propiedad intelectual como forma de animar a la gente a desarrollar software, al principio se desarrollará menor cantidad de software, pero ese software será más útil. No está claro si la satisfacción total que el usuario recibirá será menor; pero si esto es así, o si queremos aumentarla de todas formas, existen otras maneras de promover el desarrollo, exactamente igual que hay formas alternativas a los puestos de peaje para conseguir dinero de las carreteras. Antes de que empiece a hablar sobre cómo se puede hacer esto, primero quiero preguntar cuánta promoción artificial es verdaderamente necesaria.

Programar es Divertido

Existen algunas líneas de trabajo en las que pocos entrarán si no es por dinero; construcción de carreteras, por ejemplo. Hay otros campos de estudio y arte en las que existe escasa probabilidad de enriquecerse, en los que la gente entra por su fascinación o por el valor hacia la sociedad que han percibido en ellos. Algunos ejemplos son la lógica matemática, música clásica y arqueología; y organización política entre trabajadores. La gente compite, más tristemente que amargamente, por las pocas posiciones remuneradas existentes, ninguna de las cuales lo está abundantemente. Quizá tengan que pagar por la posibilidad de trabajar en ese campo, si pueden permitírselo.

Un campo así puede transformarse a sí mismo de la noche a la mañana si empieza a ofrecer la posibilidad de enriquecer. Cuando un trabajador se convierte en rico, otros piden la misma oportunidad. Pronto todos pedirán grandes sumas de dinero por aquello que hacían antes por placer. Después de un par de años, todo el mundo relacionado con ese campo se burlará de la posibilidad de realizar el trabajo en ese campo sin grandes sumas de dinero a cambio. Aconsejarán a planificadores sociales el asegurarse que estas devoluciones de capital sean posibles, creando privilegios especiales, poderes y monopolios alegando que son necesarios para lograrlo.

Este cambio sucedió en el campo de la programación computacional de la década pasada. Hace quince años uno podía encontrarse con artículos sobre ``adicción a los ordenadores": los usuarios estaban ``conectados" y tenían hábitos que les costaban cien dólares por semana. Generalmente se aceptaba que la gente amase tanto el programar como para acabar con sus matrimonios. Hoy en día, se entiende que nadie programe sin recibir una excelente paga por ello. La gente ha olvidado lo que sabía hace quince años.

Cuando en cierto momento ocurre que la mayoría de la gente únicamente trabajará en un determinado campo por una buena paga, ya no es necesario que se mantenga así. La dinámica del cambio puede efectuarse al revés si la sociedad proporciona el empuje inicial. Si anulamos la posibilidad de grandes riquezas, entonces, después de un tiempo, cuando la gente haya reajustado sus actitudes, volverán una vez más a trabajar en ese campo por el placer de hacerlo.

La pregunta, ``¿Cómo podemos pagar a los programadores?", resulta ser una pregunta más fácil cuando nos damos cuenta de que no es verdaderamente un problema acerca de pagarles una fortuna. Es más fácil mejorar una forma simple de ganarse la vida.

Financiar Software Libre

Las instituciones que pagan a los programadores no tienes por qué ser necesariamente empresas de software. Otras muchas instituciones que ya existen se pueden encargar de ello.

Los fabricantes de hardware saben que es esencial colaborar en el desarrollo de software incluso aunque no puedan controlar el uso de ese software. En 1970, la mayoría del software era libre porque no habían considerado la posibilidad de restringirlo. Hoy en día, su deseo creciente de entrar en consorcios muestra su

realidad y es que ser dueños de software no es lo que les importa verdaderamente.
Las universidades producen bastantes proyectos de programación. Hoy en día, generalmente venden los resultados, cuando en la década de los 70 no lo hacían. ¿Hay alguna duda de que la universidades desarrollarían software libre si les estuviese prohibida la venta de software? Estos proyectos podrían estar respaldados por los mismos contratos y subvenciones gubernamentales que ahora respaldan al desarrollo de software propietario.
Es común hoy en día que los investigadores universitarios obtengan subvenciones para desarrollar un sistema, desarrollarlo casi hasta el punto de completarlo y llamar a eso ``acabado", y luego que las empresas lo retomen allí dnde se dejó y lo conviertan en útil. A veces etiquetan a la versión sin acabar ``libre"; si están corruptos de forma meticulosa entonces consiguen una licencia de exclusividad por la universidad. Esto no es un secreto; se admite abiertamente por todo el mundo involucrado. Sin embargo, si los investigaodres no estuviesen tentados de hacer estas cosas, seguirían investigando de todas formas.
Los programadores que escriban software libre pueden vivir de vender servicios rlacionados con el software. He sido contratado para portar el Compilador GNU de C para hardware nuevo y para construir interfaces de usuario para GNU Emacs. (Ofrezco estas mejoras al público una vez que las he hecho.) También doy clases por las que me pagan.
No estoy solo trabajando de esta manera; ahora existe una corporación que está creciendo de forma exitosa que no realiza ningún otro tipo de trabajo. Algunas otras compañías proporcionan soporte comercial por el software libre del sistema GNU. Esto es el comienzo de una industria independiente de soporte de software--una industria que podría convertirse en algo grande si el software libre se impone. Proporciona a los usuarios una opción generalmente inaccesible a través del software propietario, excepto a los más ricos.
Las nuevas instituciones como Fundación para el Software Libre pueden también subvencionar a los programadores. La mayoría de los fondos de la fundación provienen de los usuarios que compran cintas magnéticas a través del correo. El software en las cintas magnéticas es libre, lo que quiere decir que cualquier usuario tiene la libertad de copiarlo y cambiarlo, pero muchos a pesar de ello pagan por conseguir copias. (Recuérdese que ``software libre" se refiere a la libertad, no al precio.) Algunos usuarios encargan cintas magnéticas de las que ya tienen una copia como una forma de contribución que ellos piensan que merecemos. La Fundación también recibe importantes donaciones de fabricantes de ordenadores.
La Fundación para el Software Libre es una sociedad sin ánimo de lucro, y sus ingresos se gastan en contratar tantos programadores como se puedan. Si se hubiese planteado como una empresa, distribuir el mismo software libre al público por el mismo precio, proporcionaría ahora una buena vida a su fundador.
Debido a que la Fundación es una sociedad sin ánimo de lucro, los programadores trabajan generalmente por la mitad por lo que podían estar haciéndolo en cualquier otro sitio. Hacen esto porque estamos libres de burocracia, y porque encuentran satisfacción sabiendo que su trabajo no encontrará obastáculos en su uso. Y lo que es más importante, lo hacen porque sienten que programar es divertido. Además, los voluntarios han escrito muchos programas útiles para nosotros. (desde hace poco, incluso escritores técnicos han empezado a colaborar.)
Esto confirma que el programar se encuentra entre los campos más fascinantes, junto con la música y el arte. No debemos temer que nadie quiera programar.

¿Qué deben los usuarios a los desarrolladores?

Los usuarios de software tienen una buena razón para sentirse moralmente obligados a contribuir a su soporte. Los desarrolladores de software libre están contribuyendo a las actividades de los usuarios, y es justo a la vez que beneficioso para los usuarios a largo plazo el proporcionar fondos para que esto continúe.
Sin embargo, esto no se aplica a los desarrolladores de software propietario, ya que la obstrucción conlleva un castigo más que una recompensa.
De manera que tenemos una paradoja: el desarrollador de software útil tiene el derecho a recibir el soporte de los usuarios, pero cualquier intento que convierta esta obligación moral en una petición destruye la base para la obligación. Un desarrollador puede o bien merecer una recompensa o pedirla, pero no las dos cosas a la vez.
Creo que un desarrollador ético enfrentado con esta paradoja debe actuar de modo que merezca la recompensa, pero debería asimismo animar a los usuarios a que realicen donaciones. Puede que los usuarios aprendan así a ayudar a los desarrolladores sin coacción, como han aprendido a ayudar a la radio pública o a las cadenas de televisión.

¿Qué es la Productividad del Software?

Si el software fuese libre seguirían habiendo programadores, pero quizá menos. ¿Sería esto malo para la

sociedad?
No necesariamente. Hoy en día las naciones desarrolladas tienen menos granjeros que en 1900, pero no creemos que esto sea malo para la sociedad porque ese menor número distribuye más comida a los consumidores que lo que solían hacer muchos más antes. Llamamos a esto Mejora de la productividad. El software libre requeriría bastantes menos programadores para satisfacer la demanda, debido al aumento en la productividad del software en todos los niveles:

- Uso más extendido de cada programa que se desarrolla.
- La posibilidad de adaptar programas existentes para configuraciones especiales en vez de tener que empezar los programas de cero.
- Mejor eduación de los programadores.
- La eliminación del esfuerzo doble en el desarrollo.

Aquellos que se opnen a la cooperación porque resultaría en el empleo de menos programdores están, en realidad, oponiéndose al aumento de productividad. Y además estas personas aceptan generalmente la creencia universal de que la industria del software necesita un incremento en su producitividad. ¿Cómo es posible esto?
``La Productividad del Software" puede significar dos cosas diferentes: la productividad general de todo el desarrollo del software, o la productividad de proyectos individuales. La productividad general es lo que a la sociedad le gustaría mejorar, y la forma más directa de lograr esto es eliminar los obstáculos artificiales a la cooperación que la reducen. Pero los investigadores que estudian el campo de ``La Productividad del Software" se centran sólo en el segundo y limitado, sentido del término, en donde la mejora precisa de avances tecnológicos difíciles.

Es la Competencia inevitable?

¿Es inevitable que la gente trate de competir y superar a sus rivales en la sociedad? Puede que así sea. Pero la competencia en sí misma no es dañina; lo dañino es el *combate*.
Existen muchas formas de competir. La Competición puede consistir en tratar de conseguir siempre más, en mejorar lo que otros han hecho. Por ejemplo, en el pasado, había una competición entre los gurús de la programación--competición que consistía en quién era capaz de producir el ordenador que realizase la cosa más fascinante, o quién era capaz de escribir el programa más corto o más rápido para una determinada tarea. Este tipo de competencia puede beneficiar a todos, *mientras* el espíritu de buen juego se mantenga.
Una competencia constructiva es suficiente para motivar a la gente a realizar grandes esfuerzos. Hay personas que compiten por ver quién es el primero en visitar todos los países de la Tierra; algunos de ellos incluso se gastan una fortuna intentándolo. Pero no sobornan a los capitanes de barcos para que dejen desamparados a sus rivales en islas desiertas. No tienen ningún problema para dejar al mejor ganar.
La competición se convierte en combate cuando los competidores intentan impedir el uno al otro en lugar de avanzar por sí mismos-- cuando ``Que gane el mejor" se convierte en ``Déjame ganar, sea el mejor o no." El software propietario es perjudicial, no porque sea una forma de competición, sino porque es una forma de combate alrededor de los ciudadanos de nuestra sociedad.
La competición en el negocio no es necesariamente un combate. Por ejemplo, cuando dos tiendas de abarrotes compiten, todo su esfuerzo se emplea en mejorar sus operaciones, no en sabotear al rival. Pero esto no demuestra un compromiso especial hacia la ética en el negocio; por el contrario, existe un pequeño margen de libertad en esta rama de los negocios, falta de violencia física. No todas las áreas en los negocios comparten ésta misma característica. Preservar información que podría ayudar al avance de todos es una forma de combate.
La ideología del negocio no prepara a la gente para resistir la tentación de combatir a la competencia. Algunas formas de combate han sido prohibidas con leyes antimonopolio, leyes sobre sinceridad en publicidad y otras más, pero lejos de generalizar esto mediante una repulsa de principios hacia el combate en general, los ejecutivos inventan otras formas de combate que no están específicamente prohibidas. Los recursos de la sociedad se despilfarran en el equivalente económico de una guerra civil.

``¿Por qué no nos vamos a Rusia?''

En los Estados Unidos de América, cualquier partidario de otra cosa que no sea la forma más extrema de *laissez-faire* ha oído a menudo esta acusación. Por ejemplo, es usada contra los defensores de un sistema de sanidad pública, como aquel que se encuentra en todas las demás naciones industrializadas del mundo libre. Es usada contra los que desean una ayuda pública al mundo de las artes, también universal en las naciones avanzadas. La idea de que los ciudadanos tienen una obligación para con el bien común se identifica en Estados Unidos con Comunismo. ¿Pero de qué modo son semejantes estas ideas?

El comunismo tal como se practicó en la Unión Soviética era un sistema de control central en donde toda la actividad era regida, supuestamente por el bien común, pero en realidad en beneficio de los miembros del partido comunista. Y donde los equipos de copia estaban estrechamente vigilados para prevenir posibles copias ilegales.
El sistema de propiedad intelectual de Estados Unidos ejerce un control central sobre la distribución de un programa, y se guarda de los equipos de copia con sistemas de protección contra copia automáticos de forma que pueda evitarse la copia ilegal.
En contraste con ello, yo estoy trabajando para construir un sistema en donde la gente es libre para decidir sus propias acciones; en particular, libre para ayudar a sus vecinos, y libre para alterar y mejorar las herramientas con las que trabajan en su vida cotidiana. Un sistema basado en la cooperación voluntaria y en la descentralización.
Así, si fuésemos a juzgar posturas por su parecido al comunismo ruso, son los dueños del software quienes son comunistas.

La Pregunta de las Premisas

Hago la suposición en este texto de que un usuario de software no es menos importante que un autor, o incluso un jefe del autor. En otras palabras, sus intereses y necesidad tienen igual peso, cuando se trata de decidir qué toma de decisión es mejor.
Esta premisa no es aceptada universalmente. Muchos mantienen que la persona que contrata al autor es fundamentalmente más importante que ninguno otro. Dicen, por ejemplo, que el propósito de tener dueños de software es el dar al que contrata al autor la ventaja que se merece-- independientemente de como puede afectar esto al público.
No tiene sentido el tratar de probar o anular estas premisas. La prueba necesita de premisas prestadas. Así que la mayoría de lo que digo está destinado sólo a aquellos que comparten las premisas que yo uso, o al menos están interesados en cuáles son las consecuencias. Para aquellos que crean que los dueños son más importantes que nadie, este documento es simplemente irrelevante.
¿Pero por qué aceptaría un gran número de estadounidenses una premisa que eleva en importancia a algunas personas sobre todo el resto del mundo? En parte debido a la creencia de que esta premisa forma parte de las tradiciones legales de la sociedad estadounidense. Algunas personas sienten que poniendo en duda la premisa implica cuestionar la base de la sociedad.
Es importante para esas personas el saber que esta premisa no forma parte de nuestra tradición legal. Nunca lo ha sido.
Así, la Constitución dice que el propósito del copyright es ``promover el progreso de la ciencia y de las artes útiles." La Corte Suprema ha discutido sobre esto, diciendo en el caso ``Fox Film contra Doyal" que `` el único interés del los Estados Unidos y el objetivo prncipal por el que se otorga el monopolio [del copyright] descansa en los beneficios generales obtenidos por el público gracias al trabajo de los autores."
No estamos obligados a estar de acuerdo con la Constitución o la Corte Suprema. (En un momento dado, los dos perdonaron el esclavismo.) Así que sus posiciones no rechazan la premisa de la supremacia del dueño. Pero yo espero que el conocimiento de que ésta suposición es una radicalmente conservadora más que una reconocida tradicionalmente debilitará su apelación.

Conclusión

Nos gusta pensar que nuestra sociedad promueve el ayudar al vecino; pero a cada rato estamos recompensando a alguien por obstaculización, o admirándolos por la riqueza que estan obteniendo de esta forma, estamos enviando el mensaje opuesto.
La acumulación de software es una expresión de nuestra disposición general a ser indeferentes sobre el bienestar de la sociedad y a favor del bien personal. Podemos conservar esta indiferencia desde Ronald Reagen a Jim Bakker, desde Ivan Boesky a Exxon, desde falta de bancos a falta de colegios. Podemos medirla con el número de personas sin hogar y la gente encarcelada. El espíritu antisocial se nutre de sí mismo, porque cada vez que vemos que la gente no nos ayudará más vano nos parece ayudarlos. Y así la sociedad degenera en una jungla.
Si no queremos vivir en una jungla, debemos cambiar nuestras actitudes. Debemos empezar enviando el mensaje de que un buen ciudadano es aquel que colabora cuando es apropiado, no aquel que es exitoso cuando toma de otros. Espero que el movimiento por el software libre pueda contribuir a esto: al menos en un área, reemplazaremos la jungla con un sistema más eficiente que anime y se base en la cooperación voluntaria.

Notas al pie

1. La palabra ``libre" en ``software libre" se refiere a libertad, no al precio; el precio pagado por una copia de un programa libre puede ser cero, bajo o (raramente) bastante alto.
2. Los asuntos de contaminación y atascos no alteran esta conclusión. Si queremos hacer que el conducir sea más caro para desanimar a conducir en general, no es ventajoso hacer esto con puestos de peaje, que contribuyen tanto a contaminar como a los atascos. Un impuesto sobre la gasolina es mucho mejor. Igualmente, el creer que la seguridad mejorará poniendo límites a la velocidad es equivocado; un acceso libre a la carretera mejora la velocidad media gracias a la ausencia de paradas y retrasos, dado cualquier límite de velocidad.
3. Uno puede ver un programa de ordenador en particular como algo dañino que no debería estar disponible en absoluto, como la base de datos de información personal de Lotus Marketplace, que fue retirado del mercado debido al rechazo público. La mayoría de lo que he dicho no se aplica a este caso, pero tiene poco sentido el discutir el querer tener un popietario con la base de que ese propietario hará que el programa esté menos accesible por la gente. El propietario no lo hará *completamente* inaccesible, como uno podría desear en el caso de un programa cuyo uso se considere destructivo.