CULTIVANDO LA NOOSFERA

Autor: Eric S. Raymond
Traducción: Javier Gemignani
Versión: 1.22 (24 agosto 2000)

Después de observar la contradicción entre la ideología oficial definida por las licencias de código abierto y el actual comportamiento de los hackers, he examinado las verdaderas costumbres que regulan la propiedad y el control del software de código abierto. Aquí muestro que implican una teoría de derechos de propiedad homólogos a la teoría de Locke sobre posesión de tierras. Luego la relaciono con un análisis de la cultura hacker como una "cultura del don" en la que los participantes compiten por prestigio, cediendo tiempo, energía, y creatividad. Finalmente, examino las implicaciones de éste análisis para resolución de conflictos en la cultura, y desarrollo algunas implicaciones preescriptivas.

1 - UNA INTRODUCCIÓN CONTRADICTORIA

Cualquiera que ve el ocupado, tremendamente productivo mundo del software de código abierto en Internet por un tiempo, es propenso a notar una interesante contradicción entre lo que los hackers de código abierto dicen que creen y la forma en que realmente se comportan (entre la ideología oficial de la cultura de código abierto y su verdadera práctica).

Las culturas son máquinas adaptativas. La cultura de código abierto es una respuesta a un conjunto identificable de caminos y presiones. Como es costumbre, la adaptación de la cultura a las circunstancias se manifiesta como conciencia ideológica y como implícita, inconsciente o semi-consciente sabiduría. Y, como no es poco común, las adaptaciones inconscientes son en parte raras con la conciencia ideológica.

En éste ensayo, me adentraré en las raíces de esa contradicción, y la usaré para encontrar esos caminos y presiones. Deduciremos algunas cosas interesantes sobre la cultura hacker y sus costumbres. Concluiremos sugiriendo caminos en la cual la sabiduría implícita de la cultura hacker puede ser mejorada.

2 - LAS VARIACIONES DE LA IDEOLOGÍA HACKER

La ideología de la cultura de código abierto (lo que los hackers dicen que creen) es claramente un tópico complejo por sí sólo. Todos sus miembros coinciden que el código abierto (esto es, software que es redistribuido gratuitamente y que puede ser fácilmente evolucionado y modificado para adaptarse a cambios necesarios) es bueno y de valioso significado y esfuerzo colectivo. Este acuerdo define efectivamente la membresía en la cultura. Sin embargo, las razones individuales y las que dan varias subculturas por ésta creencia varían considerablemente.

Un grado de variación es el fervor que aplican las distintas personas; si el desarrollo de código abierto es apreciado meramente como un medio necesario para un fin (buenas herramientas y juguetes divertidos y un juego interesante que jugar) o como un fin en sí mismo.

Una persona de gran fervor dirá: "El software libre es mi vida! Existo para crear programas útiles y recursos informativos, para luego cederlos". Una persona de fervor moderado dirá: "El código abierto es bueno, en el que estoy dispuesto a dedicar un tiempo significante". Una persona de poco fervor dirá: "Sí, el código abierto está bien a veces. Yo juego con él y respeto a la gente que dedica tiempo en él".

Otro grado de variación es la hostilidad hacia el software comercial y/o las compañías que dominan el mercado de software comercial.

Una persona muy anti-comercial dirá: "El software comercial es un robo. Yo escribo software libre para acabar con éste mal". Una persona moderadamente anti-comercial dirá: "El software comercial en general está bien porque los programadores merecen ser pagados, pero las compañías son malignas". Una persona que no es anti-comercial dirá: "El software comercial está bien, yo sólo uso y/o escribo software de código abierto porque me gusta más". (Actualmente, dado el crecimiento del software de código abierto desde la primera versión pública de éste ensayo, uno también puede escuchar "El software comercial está bien, siempre y cuando consiga el código o haga lo que quiero que haga.")

Todas éstas actitudes implicadas en las anteriores categorías son representadas en la cultura de código abierto. La razón por la que vale la pena destacar las distinciones es porque implican diferentes agendas, y diferentes comportamientos.

Históricamente, la parte más visible y mejor organizada de la cultura hacker ha sido muy fervorosa y muy anti-comercial. La Free Software Foundation (FSF) fundada por Richard M. Stallman (RMS) soportó una gran parte del desarrollo de código abierto desde principios de los 80's, incluyendo herramientas como Emacs y GCC las cuales todavía son las bases del mundo de código abierto, y parece que lo serán en el futuro visible.

Por varios años la FSF fue el foco más importante de la comunidad de código abierto, produciendo una gran cantidad de herramientas todavía críticas para la comunidad. La FSF también fue el único sponsor del código abierto con una identidad institucional visible para observadores exteriores de la cultura hacker. Ellos efectivamente definieron el término "software libre", deliberadamente dándole un peso confrontacional (que la nueva etiqueta "código abierto" deliberadamente elude).

De éste modo, las percepciones de la cultura hacker desde dentro y desde fuera tendió a identificar a la cultura con el fervor y la actitud anti-comercial de la FSF. El vigoroso camino de la FSF para frenar el software comercial se convirtió en lo más cercano a una ideología hacker, y RMS lo más cercano a un líder de la cultura hacker.

Los términos de la licencia de la FSF, la GPL (General Public License), expresa sus actitudes. Es muy ampliamente usada en el mundo del código abierto. El Metalab de North Carolina (formalmente conocido como Sunsite) es el más grande y popular repositorio de software en el mundo Linux. En julio de 1997 la mitad de los paquetes de software usaban la licencia GPL.

Pero la FSF nunca fue el único juego en la ciudad. Siempre hubo una tensión silenciosa, menos confrontacional y más amigable con el mercado en la cultura hacker. Los pragmáticos no eran tan leales a una ideología como a un grupo de tradiciones de ingenieros fundadas en tempranos esfuerzos del código abierto que predecieron la FSF. Estas tradiciones incluyeron, principalmente, las culturas técnicas de Unix y la Internet pre-comercial.

La actitud típica de pragmatismo es sólo moderadamente anti-comercial, y su mayor crítica contra el mundo

corporativo no es sólo el atesoramiento. Más bien es la negativa del mundo para adoptar sus superiores métodos incorporando Unix y estándares abiertos y software de código abierto. Si el pragmático odia algo, es menos probable que acapare en general como los reyes del software; antiguamente IBM, ahora Microsoft.

Para los pragmáticos, la GPL es más importante como una herramienta que un fin en sí mismo. Su valor principal no es como un arma contra el atesoramiento de las empresas, si no como una herramienta para animar el compartimiento de software y el crecimiento del desarrollo del modo bazar en la comunidad. El pragmático valora tener buenas herramientas y juguetes más que su antipatía por el comercialismo, y puede usar software comercial de alta calidad sin incomodidad ideológica. Al mismo tiempo, su experiencia con el código abierto le enseñó standards de calidad técnica que muy poco software de código cerrado tiene.

Por muchos años, el modo de vista del pragmático se expresó dentro de la cultura hacker de un modo negativo a adentrarse de lleno en el GPL en particular y a la agenda de la FSF en general. Durante los 80' y principios de los 90', ésta tendencia tendió a ser asociada con los fans del Unix de Berkeley, usuarios de la licencia BSD, y los primeros esfuerzos para crear Unixes de código abierto bajo las bases de la licencia BSD. Estos esfuerzos fallaron en crear comunidades de bazar de tamaño significante, y se volvieron seriamente fragmentados e inefectivos.

No fue hasta la explosión de Linux a principios de los años 93-94 que el pragmatismo halló su verdadera base de poder. Aunque Linus Torvalds nunca se opuso a RMS, él puso un ejemplo mirando benignamente al crecimiento comercial de Linux, apoyando públicamente el uso de software comercial de alta calidad para tareas específicas, y burlándose gentilmente de los más puros y fanáticos elementos de la cultura de código abierto.

Un efecto colateral del rápido crecimiento de Linux fue la introducción de un gran número de nuevos hackers para los cuales Linux era su lealtad principal y la FSF era primariamente de interés histórico. El pensamiento de la nueva ola de hackers puede describir el sistema como "la elección de una generación GNU", más tendenciosa a emular a Torvalds que a Stallman.

Incrementalmente fueron los puristas anti-comerciales los que se encontraron en una minoría. Cuanto cambiaron las cosas no se hicieron aparentes hasta el anuncio de Netscape en febrero de 1998 en el que decía que distribuiría el código fuente del Navigator 5.0. Esto incitó más interés en el software libre dentro del mundo corporativo. La llamada subsiguiente a la cultura hacker para explotar ésta oportunidad sin precedente y renombrar el "software libre" como "código abierto" se encontró con una aprobación instantánea que sorprendió a todos los involucrados.

La parte pragmática de la cultura de código abierto se estaba convirtiendo en policéntrica hacia mediados de los 90'. Otras comunidades semi-independientes empezaron a brotar de las raíces de Unix/Internet. De éstas, la más importante después de Linux era la cultura de Perl bajo las órdenes de Larry Wall. Más pequeñas, pero significantes, eran las culturas de los lenguajes Tcl de John Osterhout, y Python de Guido van Rossum. Estas comunidades expresaron sus ideologías independientemente, inventando sus propios proyectos, no basados en la licencia GPL.

3 - TEORÍA PROMÍSCUA, PRÁCTICA PURITANA

A través de todos éstos cambios, sin embargo, prevaleció una amplia teoría de consenso de lo que era "software libre" o "código abierto". La expresión más clara de ésta teoría en común puede ser encontrada en varias licencias de código abierto, todas las cuales tienen cruciales elementos en común.

En 1997 éstos elementos en común fueron destilados en la Debian Free Software Guidelines, las cuales se convirtieron en la Open Source Definition (OSD). Bajo las guías definidas por la OSD, una licencia de código abierto debe proteger el derecho incondicional de cualquier parte a modificar (y distribuir versiones modificadas de) software de código abierto.

De éste modo, la teoría implícita de la OSD es que cualquiera puede hackear cualquier cosa. Nada previene a media docena de personas a tomar cualquier producto de código abierto, duplicar el código fuente y evolucionar con él en diferentes direcciones.

En la práctica, esas divisiones casi nunca ocurren. Las divisiones en los proyectos principales son muy raras, y siempre acompañadas de la justificación pública. Esta claro que, en casos como la división GNU Emacs/XEmacs, o gcc/egcs, o las varias fisuras de los grupos BSD, que los divisores sintieron que iban contra una norma claramente fuerte de la comunidad.

De hecho la cultura de código abierto tiene un elaborado pero inadmitido conjunto de costumbres propietarias. Estas costumbres regulan quien puede modificar el software, las circunstancias en las cuales puede ser modificado, y (especialmente) quien tiene el derecho a redistribuir versiones modificadas a la comunidad.

Los tabúes de una cultura tiran sus normas en alivios agudos. Por lo tanto, sería útil si resumimos algunos aquí.

- Hay una gran presión social contra la división de proyectos. Esto no sucede excepto una gran necesidad, con mucha justificación pública, y con otro nombre.
- Distribuir cambios en un proyecto sin la cooperación de los moderadores no se produce, excepto en casos especiales como la portación de arreglos triviales en el software.
- No se quita el nombre de una persona de los créditos del proyecto sin el consentimiento explicito de la misma.

En el resto de éste ensayo, examinaremos éstos tabúes y costumbres de propiedad en detalle. Inquiriremos no sólo en como funcionan sino que revelan sobre la dinámica social e incentivos estructurales de la comunidad de código abierto.

4 - PROPIEDAD Y CÓDIGO ABIERTO

Que significa propietario cuando la propiedad es infinitamente reduplicable, altamente maleable, y la cultura que la rodea no coercibe relaciones de poder?

Realmente, en el caso de la cultura de código abierto ésta es una pregunta fácil de responder. El propietario de un proyecto de software es aquel que tiene el derecho exclusivo, reconocido por la comunidad, a distribuir versiones modificadas.

(Al discutir sobre propiedad en ésta sección usaré el singular, ya que todos los proyectos son propiedad de alguna persona. Debe ser entendido, sin embargo, que los proyectos pueden ser propiedad de grupos. Examinaremos las dinámicas internas de esos grupos más tarde en éste ensayo.)

Coincidiendo con las licencias standard de código abierto, todas las partes son iguales en el juego evolucionario. Pero en práctica hay una muy bien reconocida distinción entre los parches oficiales, aprobados e integrados en el software por los mantenedores públicamente reconocidos, y los parches de terceras partes. Los parches que

no sean oficiales son poco comunes, y generalmente no confiables.

Esta redistribución pública es un punto en disputa fundamentalmente fácil de establecer. La costumbre anima a la gente a corregir el software para uso personal cuando es necesario. La costumbre es indiferente para la gente que redistribuye versiones modificadas dentro de un grupo de desarrollo. Es solamente cuando las modificaciones son puestas en la comunidad de código abierto, a competir con el original, que la propiedad se convierte en un punto en disputa.

Existen, en general, tres formas de adquirir la propiedad de un proyecto de código abierto. Uno, el más obvio, es fundando el proyecto. Cuando un proyecto tiene sólo un mantenedor y todavía se encuentra activo, la costumbre no permite cuestionar quien es dueño del proyecto.

El segundo camino es que la propiedad del proyecto sea pasada hacia ti por el propietario anterior. Es bien entendido en la comunidad que los propietarios de los proyectos tienen derecho a pasar el proyecto a sucesores competentes cuando no están dispuestos o no pueden invertir el tiempo necesario en el desarrollo o trabajo de mantenimiento.

Es significante que en el caso de proyectos mayores, éstas transferencias de control son generalmente anunciadas.

Para proyectos menores, es suficiente con cambiar la historia incluida en el proyecto para notar el cambio de propiedad. Si el anterior propietario no transfirió voluntariamente el control, él o ella deben reasegurarse el control con el soporte de la comunidad, objetando públicamente dentro de un período razonable de tiempo.

La tercera forma de adquirir la propiedad de un proyecto es observar que éste necesite trabajo y el propietario ha desaparecido o perdido interés. Si quieres hacer esto, es tu responsabilidad hacer un esfuerzo para encontrar al propietario. Si no lo encuentras, entonces puedes anunciar el relevamiento (como por ejemplo en un newsgroup de Usenet dedicado a la aplicación o tema correspondiente), diciendo que el proyecto parece estar huérfano y que estás considerando tomar responsabilidad de él.

La costumbre demanda que permitas pasar cierto tiempo antes de seguir con un anuncio en el que te declares el nuevo propietario. En éste intervalo, si alguien más anuncia que estaba trabajando en el proyecto, su reclamo esta sobre el tuyo. Es bien considerado dar pública noticia de tus intenciones más de una vez. Mejor aún si lo anuncias en foros relevantes (newsgroups, listas de correo); y más aún si demuestras paciencia esperando por respuestas. En general, el esfuerzo más visible de permitir al anterior propietario u otros a responder, mejora tu demanda si no se presenta ninguna respuesta.

Si has pasado por éste proceso en busca de los usuarios del proyecto, y no hay objeciones, entonces puedes proclamarte propietario del proyecto huérfano y tomar nota de ello en el archivo de la historia del proyecto. Esto, si embargo, es menos seguro que el paso de mando a través del anterior propietario, y no puedes esperar ser considerado completamente legítimo hasta que hayas hecho mejoras substanciales en busca de la comunidad de usuarios del proyecto.

He observado éstas costumbres en acción por veinte años, yendo más allá de la historia pre-FSF de software de código abierto. Hay varias características interesantes. Una de las más interesantes es que los hackers las han seguido sin estar completamente conscientes de ellas. Verdaderamente, lo anterior puede ser el primer resumen consciente y razonable que haya sido escribido.

Otra es que, por costumbres inconscientes, las han seguido con notable consistencia. He observado literalmente la evolución de cientos de proyectos de código abierto, y todavía puedo contar el número de violaciones significantes que he observado o escuchado con mis dedos.

Una tercera característica es que mientras éstas costumbres han evolucionado a través del tiempo, lo han hecho en una dirección consistente. Esa dirección ha llevado a animar a más responsables, a ser más notorio públicamente, y más cuidado en preservar los créditos y cambios en la historia de los proyectos en formas que establezcan la legitimidad de los actuales propietarios.

Estas características sugieren que las costumbres no son accidentales, sino que son producto de cierto tipo de agenda implícita o modelo generativo en la cultura de código abierto que es completamente fundamental en la forma en la que opera.

Una anterior respuesta apuntó a que la contrastante cultura hacker de Internet con la cultura cracker/pirata ilumina los modelos generativos de los dos bastante bien. Retornaremos a los cracker para contrastarlos más adelante en éste ensayo.

5 - LOCKE Y LOS TÍTULOS DE TIERRA

Para entender éste modelo generativo, ayuda el notar una analogía histórica para éstas costumbres que están lejos del dominio del interés de los hackers. Como estudiantes de la historia legal y filosofía política se puede reconocer, la teoría de propiedad que implican es virtualmente idéntica a la teoría de ley común anglo-americana de tenencia de tierras.

En ésta teoría, hay tres formas de adquirir la propiedad de una tierra.

En una frontera, donde existe tierra que nunca tuvo un dueño, uno puede adquirir su propiedad cultivando, mezclando su labor en la tierra que no es de su propiedad, cercarla, y defendiendo su título.

Los medios usuales de transferencia en áreas establecidas son mediante la *transferencia de título*. Esto es, recibiéndolo de su anterior dueño. En ésta teoría, el concepto de *encadenamiento de título* es importante. La prueba ideal de propiedad es una cadena de transferencias extendiéndose hacia donde la tierra fue originalmente cultivada.

Finalmente, la teoría de ley común reconoce que el título de la tierra puede ser perdido o abandonado (por ejemplo, si el propietario muere sin herederos, o los registros necesarios para establecer una cadena de título para una tierra vacante expiraron).

Esta teoría, como las costumbres hackers, evolucionaron orgánicamente en un contexto donde la autoridad central era débil o inexistente. Se desarrolló durante un período de mil años. Ya que fue sistematizada y racionalizada en los principios de la era moderna por el filósofo político inglés John Locke, es a veces referida como la teoría Lockeana de la propiedad.

Lógicamente teorías similares intentaron evolucionar donde la propiedad era altamente económica o el valor de supervivencia y ninguna autoridad era lo suficientemente poderosa para forzar la locación central de bienes escasos. Esto es verdad hasta en las culturas buscadoras que no tienen concepto de la propiedad. Por ejemplo, en las tradiciones del desierto de Kalahari, no hay propiedad de terreno. Pero hay propiedad de agua y

manantiales bajo una teoría reconociblemente semejante a la de Locke.

El ejemplo del desierto de Kalahari es instructivo, porque muestra que las costumbres de la propiedad de Locke surgen solamente cuando el resultado esperado del recurso excede el costo esperado de defenderlo. La búsqueda de terrenos no es propiedad porque el retorno de la búsqueda es altamente impredecible y variable, y (aunque altamente apreciados) no es una necesidad para sobrevivir día a día. El agua, por otro lado, es vital para sobrevivir y lo suficientemente escasa para defender.

La noosfera del título de éste ensayo es el territorio de ideas, el espacio de todos los posibles pensamientos. Lo que vemos implicado en la costumbre de propiedad de los hackers es la teoría Lockeana de los derechos de la propiedad en un subconjunto de la noosfera, el espacio de todos los programas. Por lo tanto "Cultivando la noosfera", que es lo que cada fundador de un nuevo proyecto de código abierto hace.

Faré Rideau (fare@tunes.org) correctamente señala que los hackers no operan exactamente en el territorio de las ideas puras. Él afirma que lo que los hackers poseen son los proyectos de programación; focos intencionales de labor material (desarrollo, servicios, etc.), a los que son asociadas cosas como la reputación, confianza, etc. Por lo tanto afirma que el espacio sobre el que se extienden los proyectos hacker, no es el de la noosfera sino algún tipo doble de ella, el espacio de los proyectos de programa. (sería etimológicamente correcto llamar a éste doble espacio la *ergosfera* o *esfera de trabajo*.)

En la práctica, la distinción entre noosfera y ergosfera no es importante para el propósito de éste ensayo. Es dudoso si la noosfera en el puro sentido en el que insiste Faré pueda decirse que exista en cualquier sentido significativo; uno tendría que ser un filósofo de Platón para creer en ello. Y la distinción entre noosfera y ergosfera es sólo de práctica importancia si uno desea afirmar que esas ideas (los elementos de la noosfera) no pueden ser poseídas, pero los proyectos sí. Esta pregunta nos conduce a puntos en cuestión en la teoría de propiedad intelectual la cual está más allá del campo de éste ensayo.

Para evitar la confusión, sin embargo, es importante notar que ni la noosfera ni la ergosfera son lo mismo que la totalidad de locaciones virtuales en el medio electrónico llamado a veces (para disgusto de muchos hackers) ciberespacio. La propiedad en el ciberespacio es regulada completamente por reglas diferentes. Esencialmente, quien es propietario de los medios y las máquinas en el cual parte del ciberespacio es hospedado, es propietario de esa pieza del ciberespacio como resultado.

La costumbre de la lógica lockeana sugiere fuertemente que los hackers de código abierto observen las costumbres que tienen para defender algún tipo de retorno por su esfuerzo. El retorno debe ser más significante que el esfuerzo de cultivar proyectos, el costo de mantener el documento de la historia del proyecto, y el costo de hacer notificaciones públicas y el período de espera antes de tomar posesión de un proyecto huérfano.

Además, la producción de código abierto debe ser algo más que simplemente el uso del software, algo que pueda ser comprometido, o diluido por las divisiones. Si el uso fuera el único problema, no habría ningún tabú contra las divisiones, y la propiedad de código abierto no se parecería a la tenencia de la propiedad. De hecho, éste mundo alternativo (donde el uso es el único producto, y la división no es problemática) es el implicado por las licencias de código abierto existentes.

Podemos eliminar ciertos tipos de producción. Ya que no se puede forzar efectivamente sobre una conexión de red, la búsqueda de poder está fuera de alcance. Igualmente, la cultura de código abierto no tiene algo parecido al dinero o una escasa economía interna, por eso los hackers no pueden estar persiguiendo algo cercanamente

análogo a la riqueza material.

Existe una manera que la actividad de código abierto ayude a la gente a ganar dinero, sin embargo. Ocasionalmente, la reputación que uno gana en la cultura hacker puede derramarse sobre el mundo real en caminos económicamente significantes. Puedes recibir una mejor oferta de trabajo, o un trabajo de consultor, o un negocio para un libro.

Este tipo de efecto colateral es raro y marginal para la mayoría de los hackers; demasiado para hacerlo convincente bajo una única explicación, aún si ignoramos las repetidas protestas de los hackers de que hacen lo que hacen no por dinero, sino como un idealismo o por amor.

Sin embargo, la forma en que esos efectos colaterales económicos son mediados merece una examinación. Abajo veremos que un entendimiento de la dinámica de la reputación dentro de la cultura de código abierto tiene suficiente poder explicativo.

6 - LA MILICIA HACKER COMO UNA CULTURA DEL DON

Para entender el rol de la reputación en la cultura de código abierto, es útil moverse más en la historia dentro de la antropología y la economía, y examinar la diferencia entre las *culturas de intercambio* y las *culturas del don*.

Los seres humanos tienen un interés innato por competir por el status social; está conectado con nuestra historia evolutiva. Para el 90% de esa historia que corre anteriormente a la invención de la agricultura, nuestros ancestros vivieron en pequeñas bandadas nómades que cazaban y recolectaban. Los individuos de mayor status (aquellos más efectivos en formar coaliciones y persuadir a los demás a cooperar con ellos) tenían las parejas más saludables y acceso a la mejor comida. Esta búsqueda de status se expresa a sí misma en diferentes formas, dependiendo largamente en el grado de escasez de bienes para la supervivencia.

La mayoría de las formas que los humanos tienen de organizarse son adaptándose a la escasez y a lo que quieren. Cada forma tiene diferentes formas de ganar status social.

La forma más simple es el *mando jerárquico* En los mandos jerárquicos, la locación de bienes escasos es hecho por una autoridad central y respaldada por la fuerza. Los mandos jerárquicos se balancean muy pobremente; se convierten en incrementalmente brutales e ineficientes a medida que aumenta su tamaño. Por ésta razón, los mandos jerárquicos por encima del tamaño de una familia extendida son casi siempre parásitos en una economía de diferente tipo. En los mandos jerárquicos, el status social es primariamente determinado por el acceso al poder coercitivo.

Nuestra sociedad es predominantemente una *economía de intercambio*. Esta es una adaptación sofisticada a la escasez que, a diferencia del modelo de mando, se balancea bastante bien. La locación de bienes escasos se realiza en forma descentralizada a través del canje y la cooperación voluntaria (y de hecho, el efecto dominante del deseo competitivo es producir un comportamiento cooperativo). En una economía de intercambio, el status social es primariamente determinado por el control sobre las cosas (no necesariamente materiales) para usar o canjear.

La mayoría de la gente tiene modelos implícitos mentales para ambos, y como interactúan entre ellos. El gobierno, la milicia, y el crimen organizado (por ejemplo) son mandos jerárquicos en la gran economía de intercambio que llamamos *mercado libre*. Hay un tercer modelo, sin embargo, que es radicalmente diferente de

ambos y generalmente no reconocido excepto por los antropólogos; la cultura del don.

Las culturas del don son adaptaciones no de la escasez sino de la abundancia. Surgen en poblaciones que no tienen problemas significantes de escasez de bienes para sobrevivir. Podemos observar a las culturas del don en acción entre culturas aborígenes viviendo en zonas ecológicas con clima templado y comida abundante. También podemos observarlas en ciertos estratos de nuestra sociedad, especialmente en el show businness y entre los muy ricos..

La abundancia dificulta las relaciones de mando para sustentar e intercambiar relaciones. En las culturas del don, el status social es determinado no por lo que controlas sino por lo que entregas.

Así los multi-millonarios elaboran y publican actos de filantropía. Y así las largas horas de esfuerzo de los hackers producen código abierto de alta calidad.

Examinado de ésta forma, queda muy claro que la sociedad hacker de código abierto es de hecho una cultura del don. Dentro de ella, no hay seria escasez de necesidades de sobrevivencia. El software es compartido gratuitamente. Esta abundancia crea una situación en la que la única medida de éxito es la reputación entre sus pares.

Esta observación no es en sí misma enteramente suficiente para explicar las características observadas de la cultura hacker, sin embargo. Los crackers y warez d00dz tienen una cultura del don que prospera en los mismos medios (electrónicos) como el de los hackers, pero su comportamiento es muy diferente. La mentalidad del grupo en su cultura es mucho más fuerte y más exclusivo que entre los hackers. Ellos acaparan secretos antes que compartirlos; uno tiene más probabilidades de encontrar grupos crackers distribuyendo archivos ejecutables que crackean software antes que pistas que te digan como lo hicieron.

Lo que esto muestra, en caso de que no haya sido obvio, es que hay más de una forma de llevar a cabo una cultura del don. La historia y los valores importan. He resumido la historia de la cultura hacker en *A Brief History of Hackerdom*; las formas en que se formó su comportamiento actual no son misteriosas. Los hackers definieron su cultura por un conjunto de elecciones sobre la forma en que la competición se llevaría. Es ésta forma la que examinaremos en el resto de éste ensayo.

7 – LA ALEGRÍA DE HACKEAR

Haciendo éste análisis de la reputación, no quiero devaluar o ignorar la pura satisfacción artística de diseñar buen software y hacerlo funcionar. Todos experimentamos este tipo de satisfacción y la prosperidad que tiene. La gente para quien no es una significativa motivación nunca se convierte en hacker en primer lugar, como la gente que no le gusta la música nunca se convierte en compositora.

Entonces quizás debemos considerar otro modelo de comportamiento hacker en el cual la pura alegría de la mano de obra es la motivación primaria. Este modelo de *mano de obra* tendría que explicar las costumbres hackers como una forma de maximizar las oportunidades de la mano de obra y la cualidad de sus resultados. Entra esto en conflicto o sugiere diferentes resultados que el modelo de reputación?

En realidad no. Examinando el modelo de *mano de obra*, volvemos a los mismos problemas que llevaron a la cultura hacker a operar como una cultura del don. Como uno puede maximizar la calidad si no hay como medirla? Si la economía de escasez no opera, que medidas hay disponibles además de la evaluación de pares?

Parece que cualquier cultura de mano de obra se debe estructurar a sí misma a través de la reputación. Y, de hecho, podemos observar esta dinámica en muchas culturas de mano de obra históricas, desde los gremios medievales en adelante.

Con importante respeto, el modelo de *mano de obra* es más débil que el modelo de *cultura del don*; no ayuda explicar la contradicción que empezamos en éste ensayo.

Finalmente, la motivación de la mano de obra en sí misma puede no ser psicológicamente removida del juego de la reputación como querríamos asumir. Imagina tu hermoso programa guardado en un cajón y no usado nunca más. Ahora imagínalo siendo usado efectivamente y con placer por mucha gente. Que te da más satisfacción?

Sin embargo, mantendremos un ojo en el modelo de mano de obra. Es de intuitivo interés para muchos hackers, y explica ciertos aspectos de comportamiento individual suficientemente bien.

Después que publiqué la primera versión de éste ensayo en Internet, una persona comentó: "Puede que no trabajes por reputación, pero la reputación es un pago real, con consecuencias si haces el trabajo bien." Este es un sutil e importante punto. La reputación incentiva a continuar operando aunque una persona no este conciente de ello; así, aunque un hacker comprenda su propio comportamiento como parte del juego de la reputación, su comportamiento se adaptará al juego.

Otras personas relacionaron la recompensa de los pares y la alegría de hackear como los niveles por encima de la necesidad de subsistencia en el bien conocido modelo de *valores jerárquicos* de Abraham Maslow como una motivación humana. Desde este punto de vista, la alegría de hackear es una trascendencia necesaria que no será consistentemente expresa hasta que los niveles inferiores necesiten (incluyendo aquellos de seguridad física y pertenencia o estima de los pares) ser mínimamente satisfacidos. Así, el juego de la reputación puede ser crítico en proveer un contexto social dentro del cual la alegría de hackear pueda convertirse en el motivo primario del individuo.

8 – LAS VARIAS CARAS DE LA REPUTACIÓN

Hay razones generales para toda cultura del don de por qué tiene valor jugar por el prestigio:

Primero, la reputación entre los pares es una recompensa primaria. Estamos predispuestos a experimentarlo de esa forma por razones evolucionarias tocadas anteriormente. (Mucha gente aprende a redirigir su camino por prestigio en varias sublimaciones que no tienen conexión obvia para cierto grupo, como el honor, integridad ética, piedad, etc.; esto no cambia el mecanismo subrayado.)

Segundo, el prestigio es un buen camino (y en una pura economía del don, el único) para atraer la atención y cooperación de otros. Si uno es bien conocido por generosidad, inteligencia, de buen trato, habilidad de líder, u otras buenas cualidades, se vuelve mucho más fácil persuadir a otra gente en que se beneficiarán en asociarse a ti.

Tercero, si tu economía del don está en contacto con una economía de intercambio o con mandos jerárquicos, tu reputación puede volcarse en ellas y ganar mayor status también en ellas.

Más allá de éstas razones generales, las condiciones peculiares de la cultura hacker hacen al prestigio aún más precioso que una cultura del don del mundo real.

La principal *condición peculiar* es que los artefactos que uno entrega son muy complejos. Es mucho más difícil distinguir objetivamente un buen regalo de uno malo. El éxito de lo que uno entrega por status es dependiente del juicio crítico de los pares.

Otra peculiaridad es la relativa pureza de la cultura de código abierto. La mayoría de las culturas del don son comprometidas. Ya sea por relaciones con economías de intercambio como el comercio de bienes de lujo, o por relaciones con economías de mando como la familia o agrupaciones de clanes. No existen analogías significativas de ellas en la cultura de código abierto; así, formas de ganar status de otra forma que no sean por reputación de pares son virtualmente ausentes.

9 – DERECHOS DE PROPIEDAD E INCENTIVOS DE REPUTACIÓN

Estamos ahora en una posición para poner juntos los análisis previos en una cuenta coherente de costumbres hacker de propiedad. Entendemos ahora el producto de cultivar la noosfera; sirve como reputación entre pares en la cultura deil don de los hackers, con todas las ganancias secundarias y efectos colaterales que ello implica.

Desde éste entendimiento, podemos analizar las propiedades lockeanas de las costumbres de los hackers como un medio de maximizar los incentivos de reputación; de asegurar que los creditos de nuestros pares vayan donde es debido y no vayan donde no es debido.

Los tres tabúes que observamos anteriormente tienen perfecto sentido bajo éste análisis. La reputación de uno puede sufrir injustamente si alguien se apropia de su trabajo; estos tabúes (y costumbres relacionadas) intentan prevenir que esto pase. (O, poniéndolo más pragmáticamente, los hackers generalmente se abstienen de dividir los proyectos de otros para poder negar la legitimidad de otros haciendo lo mismo contra ellos.)

- Dividir proyectos es malo porque expone a los contribuyentes a un riesgo de reputación que sólo pueden controlar estando activamente en los dos proyectos simultáneamente después de la división. (Esto generalmente sería muy confuso o difícil para ser práctico.)

- Distribuir parches no oficiales (o, mucho peor, binarios no oficiales) expone a los propietarios a un injusto riesgo de reputación.

- Sacar el nombre de alguien de un proyecto es, en un contexto cultural, uno de los peores crímenes. Roba la aportación de la victima al presentarla como propia del ladrón.

Por supuesto, dividir un proyecto o distribuir parches no oficiales también ataca directamente la reputación del grupo de desarrollo original. Si divido un proyecto o distribuyo parches no oficiales de un proyecto, estoy diciendo: “hiciste una mala decisión (fallando al llevar al proyecto donde lo estoy llevando)”; y cualquiera que use mi variación del programa está apoyando este desafío. Pero esto en sí mismo sería un desafío justo. No es suficiente por lo tanto para explicar los tabúes, aunque es indudable que contribuye a ello.

Todos éstos comportamientos infligen daño en la comunidad de código abierto como en la victima. Implícitamente dañan a la comunidad entera decrecentando la posibilidad de que cada contribuidor potencial perciba la probabilidad de que su aportación a la comunidad sea recompensada.

Es importante notar que hay explicaciones alternativas para dos de éstos tres tabúes.

Primero, los hackers frecuentemente explican su antipatía en dividir proyectos. Ellos pueden observar que la división tiende a romper en dos la comunidad de codesarrolladores, dejando a los dos proyectos hijos con menos cerebros para trabajar.

Una persona me señaló que es inusual para más de un hijo de una división a sobrevivir con suficiente mercado en el largo plazo. Esto fortifica los incentivos de todas las partes a cooperar y evitar la división, porque es difícil saber en adelanto quien estará en el lado perdedor y ver un montón de su trabajo desaparecer enteramente o languidecer en la oscuridad.

La antipatía hacia los parches no oficiales se explica observando que pueden complicar el seguimiento de bugs enormemente, e infligir trabajo en los mantenedores, que ya tienen suficiente en corregir sus propios errores.

Hay una verdad considerable en éstas explicaciones, y ciertamente hacen su parte para reforzar la teoría lockeana de la propiedad. Pero mientras que es intelectualmente atractiva, falla en explicar por qué tanta emoción o por qué se vuelve tan territorial en las infrecuentes ocasiones en la que los tabúes se rompen. No sólo por las partes involucradas, sino también por los espectadores y observadores que frecuentemente reaccionan ásperamente.

También está el tercer tabú. Es difícil ver como cualquier cosa salvo el análisis del juego de la reputación pueden explicar esto. El hecho de que éste tabú sea rara vez analizado mucho más profundamente que “No sería justo” es revelador en su propia forma, como veremos en la próxima sección.

10 – EL PROBLEMA DEL EGO

Al principio de éste ensayo mencioné que el conocimiento adaptativo inconsciente de una cultura está frecuentemente separada de su conciencia ideológica. Hemos visto un ejemplo de esto en las costumbres de propiedad lockeanas, las cuales han sido ampliamente seguidas a pesar del hecho de que violan las licencias estándar.

He observado otro ejemplo interesante de éste fenómeno discutiendo sobre el juego de la reputación con hackers. Muchos de ellos resistieron el análisis y se mostraron renuentes a admitir que su comportamiento era motivado por un deseo de reputación entre pares o *satisfacción del ego*.

Esto ilustra un punto interesante acerca de la cultura hacker. Conscientemente desconfía y desprecia el egotismo y motivaciones basadas en el ego; la auto-promoción tiende a ser cruelmente criticada, aun cuando la comunidad pueda parecer que tenga algo para ganar de ello. Como encaja ésta actitud con una estructura de incentivos que aparentemente funciona enteramente sobre el ego no tiene explicación.

Una larga parte de ella, sin duda, emana de la generalmente negativa actitud euro-americana hacia el ego. La matriz cultural de la mayoría de los hackers les enseña que desear la satisfacción del ego es una mala (o al menos inmadura) motivación. Solamente subliminadas y disfrazadas formas de *reputación de pares*, *auto-estima*, *profesionalismo* o *cumplido del orgullo* son generalmente aceptables.

Podría escribir otro ensayo entero sobre las raíces insanas de esta parte de nuestra herencia cultural, y la asombrosa cantidad de daño que hacemos creyendo que alguna vez hemos tenido motivos que no sean otro que nuestro ego.

Pero no estoy haciendo filosofía moral o sicología aquí, así que simplemente observaré un tipo menor de daño hecho por la creencia que el ego es maligno, el cual es éste: se les ha hecho difícil emocionalmente a muchos hackers entender conscientemente la dinámica social de su propia cultura.

Pero no hemos terminado con ésta línea de investigación. El tabú de la cultura contra el comportamiento del ego es mucho más intensificado en la sub-cultura hacker de lo que uno puede sospechar. Ciertamente el tabú es débil (o inexistente) entre otras culturas del don como la cultura de pares de la cultura del teatro o entre los muy ricos.

11 – EL VALOR DE LA HUMILDAD

Habiendo establecido que el prestigio es central para los mecanismos de recompensa de la cultura hacker, ahora necesitamos entender porque parece tan importante que este hecho permanezca semi-cubierto y largamente inadmitido.

El contraste con la cultura cracker es instructivo. En esa cultura, el comportamiento para la búsqueda de status es abierto y hasta bramante. Estos crackers buscan ser aclamados por realizar *zero-day warez* (software crackeado redistribuido en el mismo día de la realización de la versión original) pero mantienen la boca cerrada sobre como lo hacen. A estos magos no les gusta mostrar sus trucos. Y, como resultado, la base de conocimiento de la cultura cracker aumenta despacio.

En la comunidad hacker, por contraste, el trabajo de uno es su declaración. Hay una estricta meritocracia (el que hace más méritos gana). La mejor jactancia es decir que funciona, y que cada programador competente puede ver que es buen material. De éste modo, la base de conocimiento de la cultura hacker aumenta rápido.

El tabú contra el ego por lo tanto aumenta la productividad. Pero hay un segundo efecto; lo que es directamente protegido aquí es la calidad de la información en el sistema de evaluación de pares de la comunidad. Esto es, la jactancia es suprimida porque se comporta como una interferencia tendiente a corruptir los signos vitales de experimentos en creatividad y comportamiento cooperativo.

Por razones muy similares, no se ataca al autor sino al código. Hay una interesante sutilidad que refuerza el punto; los hackers se sienten muy libres para insultarse entre ellos sobre diferencias ideológicas y personales, pero ningún hacker ataca públicamente cuan competente es otra persona. La crítica se dirige siempre hacia los proyectos, y no hacia las personas.

Además, no se le hecha la culpa a un desarrollador por los bugs anteriores; el hecho de que un bug halla sido corregido es generalmente considerado más importante que el hecho de que halla existido. Como una persona observó, uno puede ganar status arreglando bugs de *Emacs*, pero no arreglando bugs de *Richard Stallman*.

Esto crea un interesante contraste con muchas partes de las academias, en el que corregir el trabajo defectuoso de otros es un modo importante de ganar reputación. En la cultura hacker, ese comportamiento es más bien censurado.

El medio de la cultura hacker del don es intangible, sus canales de comunicación son pobres en expresar un matiz emocional y el contacto cara a cara entre sus miembros es la excepción más que la regla. Esto da una menor tolerancia a la interferencia que otras culturas, y traza un largo camino para explicar el tabú contra los ataques a la competencia de las personas. Cualquier incidente significante de insultos entre hackers sobre la

competencia de los demás rompería intolerablemente la reputación de la cultura.

Hablar suavemente es funcional si uno aspira a ser un mantenedor de proyecto exitoso; uno debe convencer a la comunidad de que tiene un buen juicio, porque la mayor parte del trabajo de mantenedor va a ser juzgando el código de otra gente. Quién se verá inclinado a contribuir trabajo a alguien que claramente no puede juzgar la calidad de su propio código? Los contribuyentes potenciales quieren líderes de proyectos con suficiente humildad y clase para poder decir, cuando sea objetivamente apropiado, "Sí, eso anda mejor que mi versión, lo usaré". Y capaz de dar crédito cuando sea debido.

Otra razón para el comportamiento humilde es que en el mundo del código abierto, rara vez tratas de dar la impresión de que un proyecto está terminado. Esto podría llevar a un contribuyente potencial a sentir que no es necesitado. Si uno se jacta a través del código, y luego dice, "Bueno, no hace *x*, *y*, y *z*, así que no puede ser tan bueno", los parches para *x*, *y*, y *z* aparecerán rápidamente.

Finalmente, he observado personalmente que el comportamiento auto-deprecativo de algunos líderes hackers reflejan un real (y no injustificado) miedo de convertirse en objeto de culto. Algunos ejemplos son Linus Torvalds y Larry Wall. Una vez, en una cena con Larry Wall, bromeé, "Tú eres el hacker alpha aquí, así que elige el restaurant". El se acobardó. Fallar en distinguir sus valores al compartir sobre las personalidades de sus líderes ha arruinado una buena cantidad de comunidades voluntarias, un modelo del cual Larry y Linus no pueden fallar en estar completamente conscientes. En la otra mano, la mayoría de los hackers desearía tener los problemas de Larry, si sólo lo pudieran admitir.

12 – IMPLICACIONES GLOBALES DEL MODELO DEL JUEGO DE LA REPUTACIÓN

El análisis del juego de la reputación tiene más implicaciones que no pueden ser obvias inmediatamente. Muchas de ellas derivan del hecho de que uno gana más prestigio fundando un proyecto exitoso que cooperando en uno existente. Uno también gana de proyectos que son innovativos. En la otra mano, es más fácil hacerse notar contribuyendo a un proyecto existente que hacer notar a la gente un nuevo proyecto. Finalmente, es mucho más difícil competir con un proyecto exitoso que llenar un nicho vacío.

Así, hay una distancia óptima de los vecinos de uno (los proyectos que compiten con el tuyo). Estando muy cerca, el producto de uno sería un pobre regalo. Estando muy lejos, nadie sería capaz de usarlo, comprenderlo o percibir la relevancia del esfuerzo de uno (otra vez, un pobre regalo). Esto crea un modelo de cultivar en la noosfera que más bien se asemeja a los pobladores propagándose en una frontera física. Los proyectos tienden a ser empezados para llenar vacíos funcionales cerca de la frontera.

Algunos proyectos muy exitosos se convierten en *asesinos de categoría*; nadie quiere cultivar cerca de ellos porque competir contra la base establecida para obtener la atención de los hackers sería muy duro. La gente que puede encontrar sus esfuerzos distintivos termina añadiendo extensiones para esos grandes y exitosos proyectos. El clásico ejemplo de *asesino de categoría* sería *GNU Emacs*; sus variantes llenan el nicho ecológico para un editor tan completo que ningún competidor llegó más allá de la primera etapa desde los principios de los 80'. En lugar de eso, la gente programa módulos para *Emacs*.

Globalmente, éstas dos tendencias (el llenado de vacíos y los asesinos de categoría) han llevado a una tendencia predecible en los proyectos a través del tiempo. En la década del 70 la mayoría de código abierto que existía eran juegos y demos. En la década del 80 el empuje estaba en el desarrollo y herramientas de Internet. En los

90’ la acción cambió hacia los sistemas operativos. En cada caso, un nuevo y más dificultoso nivel de problemas fue desarrollado cuando las posibilidades de los anteriores fueron desarrolladas al máximo.

Esta tendencia tiene interesantes implicaciones en el futuro cercano. A comienzos de 1998, Linux se veía mucho más como un *asesino de categoría* para el nicho de los sistemas operativos de código abierto. Gente que de otra manera se dedicaría a programar sistemas operativos competidores, está ahora programando drivers y extensiones para Linux. Y la mayoría de las herramientas de bajo nivel que la cultura nunca imaginó teniendo en código abierto ya existen. Que queda?

Aplicaciones. Mientras entramos al siglo 21, parece seguro predecir que el desarrollo de código abierto se incrementara hacia el último territorio virgen, las aplicaciones para el hogar. Un claro indicador es el desarrollo de *Gimp*, el programa tipo *Photoshop* que es de código abierto y que posee una interfaz de fácil uso. Otro es el desarrollo de entornos de escritorio como *KDE* o *Gnome*.

Una persona ha destacado que la analogía del desarrollo de proyectos también explica por qué los hackers reaccionan con tanto odio hacia la política de Microsoft de acomplejar y cerrar las fuentes de los protocolos de Internet. La cultura hacker puede coexistir con la mayoría del software cerrado; la existencia de Adobe Photoshop, por ejemplo, no hace que el territorio de *Gimp* (su equivalente de código abierto) sea menos atractivo. Pero cuando Microsoft tiene éxito en acomplejar un protocolo para que sólo los propios programadores de Microsoft puedan programar software para él, no solamente perjudican a los consumidores al extender su monopolio. También reduce la cantidad y calidad de noosfera disponible para que los hackers cultiven. Entonces no asombra que los hackers se refieran a Microsoft como *contaminante de protocolos*; están reaccionando exáctamente como agricultores mirando a alguien envenenar el río.

Finalmente, el análisis del juego de la reputación explica el dicho de que no te conviertes en hacker llamándote a ti mismo hacker, sino que te conviertes en hacker cuando otros hackers te llaman hacker. Un hacker es alguien que ha mostrado que él o ella tiene la habilidad técnica y entiende como trabaja el juego de la reputación. Este juicio solamente puede ser dado por los que ya están dentro de la cultura.

13 –¿CÓMO VALORAR UN REGALO?

Hay modelos consistentes en la forma en que la cultura hacker valora las contribuciones y retornos de sus pares. No es difícil observar las siguientes reglas:

1. Si no funciona tan bien como esperaba que funcione, no es bueno. No importa que tan inteligente y original es la idea.

Esta regla no es una demanda de perfección; al software beta y experimental se le permite que tenga bugs. Es una demanda que el usuario sea capaz de estimar los riesgos.

Esta regla subraya el hecho de que el software de código abierto tiende a permanecer en estado beta durante largo tiempo, y de que no se realizan versiones 1.0 hasta que los desarrolladores estén muy seguros de que no tendrán una sorpresa desagradable. En el mundo del código cerrado, la versión 1.0 significa: “No toques esto si eres prudente.”; en el mundo del código abierto se lee como: “Los desarrolladores están apostando sus reputaciones en esto.”

2. El trabajo que extiende la noosfera es mejor que el trabajo que duplica una pieza existente de territorio.

La forma ingenua de poner esto sería: *El trabajo original es mejor que duplicar las funciones del software existente*. Pero no es en realidad así de simple. Duplicar las funciones de software cerrado cuenta tan original si haciendo eso rompes un protocolo cerrado o formato cerrado.

Así, por ejemplo, uno de los proyectos de más alto prestigio en el presente mundo de código abierto es *Samba* (el código que permite a las máquinas Unix actuar como clientes o servidores del protocolo propietario de Microsoft SMB para compartir archivos). Hay muy poco trabajo creativo para desarrollar aquí; es mayormente un problema de obtener correctamente los detalles de ingeniería inversa. Sin embargo, los miembros del grupo *Samba* son percibidos como héroes porque neutralizan un esfuerzo de Microsoft de cerrar con llave poblaciones enteras de usuarios y acordonar una gran sección de la noosfera.

3. El trabajo que se incluye dentro de una distribución es mejor que el trabajo que no se incluye. El trabajo que está dentro de todas las distribuciones principales es más prestigioso.

Las distribuciones principales incluyen no sólo las grandes distribuciones de Linux como Red Hat, Debian, Caldera, y S.u.S.E., sino también otras distribuciones que se entiende que tienen reputación por sí mismas para mantener y certificar calidad (como las distribuciones BSD o la colección de fuentes de la Free Software Foundation).

4. La utilización es la forma más sincera de adulación.

Confiar en el juicio de otros es básico para el proceso de crítica. Es necesario porque nadie tiene tiempo para probar todas las alternativas posibles. Entonces el trabajo usado por montones de gente es considerado mejor que el trabajo usado por unos pocos.

El haber hecho trabajo tan bueno que a nadie le interesa usar las alternativas es por lo tanto ganador de un gran prestigio.

5. La continua devoción al trabajo difícil y aburrido (como el debugging, o escribir documentación) es más prestigioso que hacer las cosas divertidas y fáciles.

Esta norma es como la comunidad recompensa las tareas necesarias a las que los hackers no se inclinarían naturalmente. Es en cierto sentido contradecida por:

6. Las extensiones de funciones no triviales son más prestigiosas que los parches de bajo nivel y el debugging.

La forma en que esto parece funcionar es que agregando una característica se obtiene una mayor recompensa que arreglando un bug (a menos que el bug sea muy grande). Pero cuando estos comportamientos se extienden a través del tiempo, una persona que hace largo tiempo que presta atención y arregla hasta los bugs más ordinarios puede bien tener el mismo prestigio que alguien que ha puesto la misma cantidad de esfuerzo en agregar pequeñas características.

Una persona ha resaltado que estas reglas interactúan en formas interesantes y no necesariamente recompensan la máxima utilidad posible todo el tiempo. Pregúntale a un hacker si es más probable convertirse en una persona conocida por una herramienta nueva de su propia autoría o por extensiones a la herramienta de otra persona, y puedes estar seguro que contestaría con la primera opción. Pero pregunta sobre:

(a) una herramienta nueva que es usada unas pocas veces al día en background por el sistema operativo pero que rápidamente se convierte en un *asesino de categoría*

versus

(b) varias extensiones a una herramienta existente las cuales no son originales o asesinas de categoría, pero son usadas diariamente por un gran número de usuarios

y es muy probable obtener cierta duda antes de que el hacker elija la opción (a).

Esta persona también agregó en referencia al autor de éste ensayo: "El caso (a) es *fetchmail*; el caso (b) son tus tantas extensiones de *Emacs*, como vc.el y gud.el". Y verdaderamente tiene razón; es más probable que me etiqueten como *el autor de fecthmail* que como *el autor de extensiones de Emacs*, a pesar que la última tiene más utilidad.

Lo que puede estar pasando aquí es que el trabajo con una original *marca de identidad* se hace notar más que el trabajo agregado en una *marca* existente. El elucidar estas reglas, y lo que ellas nos dicen sobre la cultura hacker, harían un buen tópico para mayor investigación.

14 – PROPIEDAD NOOSFÉRICA Y LA ETOLOGÍA DEL TERRITORIO

Para entender las causas y las consecuencias de las costumbres de propiedad lockeana, nos ayudará el mirarlas desde otro ángulo; el de la etología animal, específicamente la etología de territorio.

La propiedad es una abstracción de la territoriedad animal, la cual evolucionó como una forma de reducir la violencia dentro de las especies. Marcando sus límites, y respetando los límites de los otros, un lobo disminuye las chances de involucrarse en una pelea que lo puede debilitar o matar y hacerlo menos próspero reproductivamente. Similarmente, la función de la propiedad en las sociedades humanas es el prevenir conflictos entre humanos imponiendo límites que separen claramente el comportamiento pacífico de la agresión.

Esta de moda en ciertos círculos el describir la propiedad humana como una convención social arbitraria, pero esto es incorrecto. Cualquiera que tuvo un perro que ladraba cuando se acercaban extraños ha experimentado la continuidad esencial entre la territoriedad animal y la propiedad humana. Nuestros domesticados primos de los lobos saben, instintivamente, que la propiedad no sólo es una convención social, sino un mecanismo críticamente importante para evitar la violencia.

Demandar la propiedad (como el marcar el territorio) es un acto representativo, una forma de declarar que límites serán defendidos. El soporte de la comunidad al reclamo de la propiedad es una forma de minimizar la fricción y maximizar el comportamiento corporativo. Estas cosas permanecen verdaderas aun cuando el *reclamo de propiedad* es mucho más abstracto que una cerca o un ladrido de un perro, aun cuando es sólo la declaración del nombre del mantenedor del proyecto en el archivo README. Es todavía una abstracción de territoriedad, y (como otras formas de propiedad) basada en instintos territoriales los cuales evolucionaron para asistir una resolución al conflicto.

Este análisis etológico puede parecer muy abstracto y difícil de relacionarse con el real comportamiento hacker. Pero tiene algunas importantes consecuencias. Una es la popularidad de los web sites, y especialmente por qué los proyectos de código abierto con web sites parecen mucho más reales y substanciales que aquellos que no

tienen.

Considerado objetivamente, parece difícil de explicar. Comparado al esfuerzo involucrado en originar y mantener un pequeño programa, una página web es fácil, así que es difícil considerar una página web como evidencia substancial o de esfuerzo inusual.

Ni las funciones características de la web en sí misma son explicación suficiente. Las funciones comunicativas de una página web pueden ser tan o mejor servidas con una combinación de un sitio FTP, una lista de correo, y un servidor de Usenet. De hecho es bastante inusual que las comunicaciones rutinarias de un proyecto sean hechas a través de la web en vez de por una lista de correo o un newsgroup. Por qué, entonces, la popularidad de los sitios web como hogar de los proyectos?

La metáfora implícita en el término *home page* provee una pista importante. El software, después de todo, no tiene locación natural y es instantáneamente duplicable. Es asimilable para nuestras nociones instintivas de *territorio* y *propiedad*, pero solamente después de algún esfuerzo.

La home page de un proyecto concretiza un cultivo abstracto en el espacio de posibles programas, al expresarse como el *hogar* en el reino de la World Wide Web. Descender de la noosfera al ciberespacio no nos lleva al mundo real de las cercas y los perros que ladran, pero engancha el reclamo de la propiedad abstracta más seguramente para nuestro instinto sobre el territorio. Y esto es por lo que los proyectos con páginas web parecen más reales.

Este punto es mucho más reforzado por los hyperlinks y la existencia de buenas herramientas de búsqueda. Un proyecto con una página web es más probable que sea notado por alguien explorando su vecindario en la noosfera. Una página web es por lo tanto una mejor propaganda, un mejor acto representativo, un mayor reclamo del territorio.

Este análisis etológico también anima a mirar de más cerca los mecanismos para manejar conflictos en la cultura de código abierto. Nos lleva a esperar que, en adición a maximizar los incentivos de reputación, las costumbres de propiedad deben también tener un rol previniendo y resolviendo conflictos.

15 – CAUSAS DE CONFLICTO

En los conflictos sobre software de código abierto podemos identificar cuatro problemas principales:

- Quién realiza las decisiones del proyecto?
- Quién obtiene crédito o culpabilidad por algo?
- Cómo reducir la duplicación del esfuerzo y prevenir versiones modificadas?
- Qué es lo correcto, técnicamente hablando?

Si le echamos una segunda mirada al último problema (“Qué es lo correcto?”), éste tiende a desaparecer. Por cada pregunta así, o hay una forma objetiva de decidir o no la hay. Si la hay, se termina el problema y todos ganan. Si no la hay, se reduce a “Quién decide?”.

En conformidad, los tres problemas que una teoría de resolución de conflictos tiene que resolver sobre un proyecto son (A) donde se para en decisiones técnicas, (B) como decidir que contribuyentes tienen crédito y cómo, y (C) cómo mantener un grupo y producto para que no se fisure.

El rol de las costumbres de propiedad en la resolución de los problemas (A) y (C) es claro. La costumbre afirma que los propietarios del proyecto toman las decisiones. Hemos observado previamente que la costumbre también esfuerza mucha presión contra la dilución de los proyectos.

Es instructivo notar que estas costumbres tienen sentido aun si uno se olvida del juego de la reputación y las examina dentro de un puro modelo de mano de obra de la cultura hacker. En esta vista estas costumbres tienen menos que ver con la disolución de los incentivos de reputación que protegiendo el derecho de un obrero a ejecutar su visión en su forma elegida.

El modelo de mano de obra no es, sin embargo, suficiente para explicar las costumbres hacker sobre el problema (B). Para analizar esto, necesitamos llevar la teoría lockeana un paso adelante y examinar los conflictos y la operación de los derechos de propiedad dentro de los proyectos así bien como entre ellos.

16 – ESTRUCTURAS DEL PROYECTO Y PROPIEDAD

El caso trivial es en el que el proyecto tiene un solo propietario / mantenedor. En ese caso no hay conflicto posible. El propietario toma todas las decisiones y recoge todo el crédito y toda culpa. Los únicos conflictos posibles son sobre problemas de sucesión (quien reemplaza al propietario si éste desaparece o pierde interés). La comunidad también tiene un interés, bajo el problema (C), en prevenir la división del proyecto. Estos intereses son expresados por una norma cultural que un propietario / mantenedor debe públicamente pasar su título a alguien si él o ella no puede seguir manteniendo el proyecto.

El caso no trivial más simple es cuando un proyecto tiene múltiples co-mantenedores trabajando bajo un único *dictador benevolente* que posee el proyecto. La costumbre favorece éste modo para proyectos de grupo; se lo ha visto trabajar en proyectos tan grandes como el kernel de Linux o Emacs, y resuelve el problema de “quien decide” en una forma que no es peor que las otras alternativas.

Típicamente, una organización con un dictador benevolente evoluciona de una organización propietario / mantenedor a medida que el fundador atrae contribuyentes. Aun si el propietario se mantiene como un dictador, introduce un nuevo nivel de posibles disputas sobre quien obtiene crédito por ciertas partes del proyecto.

En ésta situación, la costumbre coloca una obligación en el propietario / dictador en darle crédito justamente a los contribuyentes (a través de, por ejemplo, menciones en el README o en el archivo de la historia del proyecto). En términos del modelo de la propiedad lockeana, esto significa que contribuyendo a un proyecto ganas parte de la reputación que éste recibe (positiva o negativa).

Persiguiendo ésta lógica, vemos que un *dictador benevolente* no es propietario enteramente de su proyecto. Aunque tiene el derecho de tomar las decisiones, él en efecto canjea acciones de la reputación total obtenida a cambio del trabajo de otros.

Mientras el proyecto del dictador benevolente cosecha más participantes, ellos tienden a desarrollar dos hileras de contribuyentes; los contribuyentes ordinarios y los co-desarrolladores. Un camino típico para convertirse en

co-desarrollador es tomando responsabilidad de una parte importante del proyecto. Otro camino es tomar el rol de caracterizar y arreglar muchos bugs. En éstas formas u otras, los co-desarrolladores son los contribuyentes que hacen una continua y substancial inversión de tiempo en el proyecto.

El rol del que toma parte importante de un proyecto es particularmente importante para nuestro análisis y merece mayor examinación. A los hackers les gusta decir que "a la autoridad le sigue la responsabilidad". Un co-desarrollador que acepta la responsabilidad de mantener una parte importante de un proyecto generalmente obtiene el control de la implementación de esa parte del proyecto y sus interfaces con el resto del proyecto, sujeto sólo a la corrección del líder del proyecto. Observamos que ésta regla efectivamente crea propiedades adjuntas en el modelo lockeano dentro de un proyecto, y tiene exactamente el mismo rol de prevención de conflictos como otros límites de propiedades.

Por costumbre, el *dictador* es esperado que consulte con sus co-desarrolladores sobre decisiones clave. Un líder sabio, reconociendo la función de las propiedades internas de los límites del proyecto, no interferirá o revertirá decisiones hechas por los co-desarrolladores.

Algunos proyectos muy grandes descartan el modelo del *dictador benevolente* completamente. Una forma de hacer esto es poner a los co-desarrolladores en un comité en el que votan las decisiones (como *Apache*). Otra forma es rotando el dictador, cuyo control es pasado ocasionalmente de un miembro a otro dentro de un círculo de privilegiados co-desarrolladores; los desarrolladores de *Perl* se organizan de éste modo.

Tales arreglos complicados son ampliamente considerados inestables y difíciles. Esta dificultad es una función del azar de los designados por el comité, y de los comités en sí mismos; éstos son problemas que la cultura hacker entiende conscientemente. Sin embargo, pienso que un poco de la incomodidad que sienten los hackers sobre las organizaciones de comité o de rotación es porque son difíciles de encajar en el modelo lockeano que los hackers usan para razonar sobre casos simples. Es problemático, en éstas complejas organizaciones, hacer una contabilidad de la propiedad en el sentido del control o de la propiedad de la reputación. Es difícil ver donde están las fronteras internas, y de éste modo es difícil evadir el conflicto a menos que el grupo disfrute de un excepcional alto nivel de armonía y confianza.

17 – CONFLICTOS Y RESOLUCIÓN DE CONFLICTOS

Hemos visto que dentro de los proyectos, una complejidad de roles en aumento es expresada por una distribución de la autoridad designada y derechos de propiedad parciales. Mientras que esto es una forma eficiente de distribuir incentivos, también diluye la autoridad del líder del proyecto (más importante, diluye la autoridad del líder de resolver conflictos potenciales).

Mientras que los argumentos técnicos sobre el diseño pueden ser los riesgos más obvios de conflicto interno, son rara vez un caso serio de contienda. Estos son relativamente fáciles de resolver por la regla territorial de que la autoridad le sigue a la responsabilidad.

Otra forma de resolver conflictos es por la madurez. Si dos contribuyentes o grupos de contribuyentes tienen una disputa, y la disputa no puede ser resuelta objetivamente, y ninguno es dueño del territorio en disputa, el lado que ha puesto más trabajo en el proyecto gana.

Estas reglas generalmente bastan para resolver la mayoría de las disputas. Cuando no bastan, la orden del líder del proyecto usualmente basta. Las disputas que sobrepasan éstos filtros son raras.

Los conflictos no se vuelven serios a menos que éstos dos criterios (“la autoridad sigue a la responsabilidad” y “la madurez gana”) apunten a diferentes direcciones, y la autoridad del líder del proyecto sea débil o ausente. El caso más obvio en el que esto puede ocurrir es una sucesión en disputa siguiente a la desaparición del líder del proyecto. He estado en una pelea de éste tipo. Fue fea, dolorosa, larga, solamente resuelta cuando todas las partes del proyecto se volvieron lo suficientemente exhaustas para cederle el control a una persona exterior, y yo devotamente esperé no estar nunca más cerca de algo de éste tipo.

Últimamente, todos éstos mecanismos de resolución de conflictos descansan en la voluntad de ejecución de la comunidad hacker. El único mecanismo de ejecución son los insultos; condenación pública de aquellos que rompen las costumbres, y la negativa a cooperar con ellos después de que ellos lo hicieron.

18 – LOS MECANISMOS DE ACULTURACIÓN Y EL VÍNCULO CON LA ACADEMIA

Una versión anterior de éste ensayo propuso la siguiente pregunta: Cómo informa la comunidad e instruye a sus miembros las costumbres de ella? Son las costumbres evidentes por sí mismas en un nivel semi-conciente, son enseñadas por ejemplos, son enseñadas por instrucciones explícitas?

Enseñar por instrucción explícita es muy raro.

Muchas normas son enseñadas con ejemplos. Para citar un caso muy simple, hay una norma sobre que toda distribución de software debe tener un archivo README o READ.ME que contenga ciertas instrucciones. Esta convención se ha establecido desde al menos principios de los 80’s.

Por otro lado, algunas costumbres hackers son organizadas por sí mismas una vez que uno ha adquirido un básico entendimiento del juego de la reputación. La mayoría de los hackers no han tenido que aprender nunca los tres tabúes que he listado antes en éste ensayo, o clamaran al menos que son evidentes por sí mismos más que transmitidos. Este fenómeno invita a un análisis más cercano.

Muchas culturas utilizan pistas escondidas (misterios en el sentido religioso de la palabra) como un mecanismo de aculturación. Estos son secretos que no son revelados a personas ajenas a la cultura, pero son esperados a ser encontrados por los que quieren ingresar a ella. Para ser aceptado, uno debe demostrar que ha entendido el misterio y lo ha aprendido.

La cultura hacker raramente hace uso extensivo de esas pistas. Podemos ver operar este proceso en al menos tres niveles:

- Misterios sobre claves. Como ejemplo, hay un newsgroup de USENET llamado alt.sysadmin.recovery que tiene un secreto muy explícito; no puedes dejar mensajes sin saber, y saber es considerado evidencia de que puedes dejar mensajes. Los asiduos a éste newsgroup tienen un fuerte tabú en contra de revelar éste secreto.

- El requerimiento de la iniciación en ciertos misterios técnicos. Uno debe absorber una gran cantidad de conocimiento técnico antes de poder dar regalos valiosos (ej.: uno debe saber al menos uno de los principales lenguajes de programación). Este requerimiento funciona como las pistas escondidas, como un filtro de calidad.

- Misterios de contexto social. Uno se implica en la cultura participando en proyectos específicos. Cada proyecto es un contexto social de hackers en el que el que quiere ser contribuyente tiene que investigar y entender socialmente como técnicamente para que funcione (una forma común en que uno hace esto es leyendo la página web del proyecto). Es a través de éstos proyectos que los nuevos experimentan el comportamiento ejemplar de los hackers experimentados.

En el proceso de adquirir éstos misterios, el que quiere ser hacker escoge el conocimiento contextual que (luego de un tiempo) hacen que los tres tabúes y otras costumbres sean evidentes por sí mismas.

Uno debe, incidentalmente, sostener que la estructura de la cultura del don en sí misma es su propio misterio central. Uno no es considerado acultural (nadie te llamará hacker) hasta que demuestres un nivel de entendimiento del juego de la reputación y sus costumbres, tabúes, y usos. Pero esto es trivial; todas las culturas demandan éste entendimiento de los que se quieren unir a ellas. Además la cultura hacker no revela ningún deseo en mantener una lógica interna y de mantener su folklore en secreto (o, al menos, nadie me ha dicho nada por revelarlos).

Varias personas que leyeron anteriormente éste ensayo han señalado que la propiedad de las costumbres hacker parecen estar íntimamente relacionadas con las prácticas del mundo académico, especialmente con las de la comunidad de investigación científica. Esta comunidad de investigación tiene problemas similares en minar un territorio de potenciales ideas productivas, y exhibe muy similares soluciones adaptativas a esos problemas en las formas en que usa la reputación.

Ya que muchos hackers tuvieron formación académica (es común aprender a hackear en la universidad) la extensión en la cual la academia comparte modelos adaptativos con la cultura hacker es de algo más que interés casual en entender como son aplicadas éstas costumbres.

Paralelismos obvios con la cultura del don que he caracterizado abundan en la academia. Una vez que un investigador ha logrado algo, no hay que preocuparse de los problemas de sobrevivencia. En la ausencia de éstos, el aumento de la reputación se convierte en el objetivo, que anima a compartir nuevas ideas a través de revistas y otros medios. Esto tiene sentido porque la investigación científica, como la cultura hacker, depende fuertemente en la idea de *mantenerse en los hombros de los gigantes*, y no tener que redescubrir principios básicos una y otra vez.

19 – EL REGALO DEJA FUERA DE COMPETENCIA AL INTERCAMBIO

Hay una posibilidad más interesante aquí. Sospecho que la academia y la cultura hacker comparten modelos adaptativos no sólo porque están genéticamente relacionados, sino porque los dos han evolucionado en la forma más óptima, dadas las leyes de la naturaleza y el instinto de los humanos. El veredicto de la historia parece ser que el capitalismo de libre mercado es la forma globalmente óptima de cooperar para la eficiencia económica; quizás, en una forma similar, el juego de la reputación de la cultura del don es la forma globalmente óptima de cooperar para generar (y verificar!) trabajo creativo de alta calidad.

El soporte de ésta teoría viene de un conjunto de estudios psicológicos en la interacción entre arte y recompensa. Estos estudios han recibido menos atención de la que deberían, en parte quizás porque sus popularizadores han mostrado una tendencia a sobre-interpretarlos en ataques generales contra el libre mercado y la propiedad intelectual. Sin embargo, sus resultados sugieren que algunos tipos de escasez económica en realidad disminuyen la productividad de trabajadores creativos como los programadores.

La sicóloga Theresa Amabile de la Universidad de Brandeis, resumiendo cautelosamente los resultados de un estudio de 1984 sobre motivación y recompensa, observó: "Puede ser que el trabajo por comisión, en general, sea menos creativo que el trabajo que es hecho por puro interés". Amabile observa que "Cuanto más compleja la actividad, más es dañada por la recompensa extrínseca". Interesantemente, los estudios sugieren que los salarios desinflados no desmotivan, pero si lo hacen los premios y bonos.

Así, sería económicamente inteligente entregar bonos a gente que vende hamburguesas o cava fosos, pero es más inteligente desligar el salario de la calidad en una tienda de programación y dejar elegir a la gente sus propios proyectos. Verdaderamente, éstos resultados sugieren que la única vez que es una buena idea recompensar la calidad en la programación es cuando el programador está tan motivado que el o ella hubieran trabajado sin la recompensa.

Otros investigadores apuntan a los problemas de autonomía y control de creatividad que tanto preocupan a los hackers. "Mientras el grado de experiencia de uno de ser auto-determinado sea limitada," dice Richard Ryan, profesor de sicología de la Universidad de Rochester, "la creatividad de uno será reducida también."

En general, presentar cualquier tarea como un medio más que como un fin en sí misma parece desmotivar. Aun ganando una competición con otros o ganando estima de sus pares puede ser desmotivizante en ésta forma si es experimentada en el trabajo como una recompensa (lo que puede explicar por qué los hackers son culturalmente prohibidos de buscar explícitamente o clamar esa estima).

Para complicar más el problema de la administración, controlar los dichos verbales parecen ser tan desmotivadores como el pago por el trabajo de uno. Hay una diferencia crítica entre decir, "Te estoy dando ésta recompensa porque reconozco el valor de tu trabajo" y "Estas recibiendo ésta recompensa porque te ajustas a los estándares." El primero no desmotiva; el segundo sí.

En éstas observaciones psicológicas podemos decir que un grupo de desarrollo de código abierto será substancialmente más productivo (especialmente a largo plazo, en el cual la creatividad se vuelve más crítica como un multiplicador de la productividad) que un grupo de igual tamaño y conocimiento, de programadores de código cerrado (des)motivados por recompensas.

Verdaderamente, parece que la prescripción de software de mayor productividad es casi una paradoja Zen; si quieres el producto más eficiente, tienes que dejar de tratar que los programadores produzcan. Manejar su subsistencia, darle sus cabezas y olvidarse de fechas límites. Para un director convencional esto suena locamente indulgente. Pero es exactamente el recipiente con el cual la cultura de código abierto está llevando a cabo su competencia.

20 – CONCLUSIÓN: DE LAS COSTUMBRES A LA LEY CONSUETUDINARIA

Hemos examinado las costumbres que regulan la propiedad y el control del software de código abierto. Hemos visto como implican una teoría de derechos de propiedad homólogos a la teoría lockeana de tenencia de tierras. Hemos relacionado eso con un análisis de la cultura hacker como una *cultura del don* en el que los participantes compiten por prestigio cediendo tiempo, energía y creatividad. Hemos examinado las implicaciones de éste análisis para la resolución de conflictos en la cultura.

La siguiente pregunta lógica es preguntarse: "Por qué importa esto?". Los hackers desarrollaron éstas costumbres sin análisis consciente y (hasta ahora) las han seguido sin análisis consciente. No queda

inmediatamente claro que el análisis consciente sea algo práctico. A menos que, quizás, nos podamos mover de la descripción a la prescripción y deducir formas de mejorar el funcionamiento de esas costumbres.

Podemos encontrar una analogía cercana para las costumbres hackers en la teoría de tenencia de tierras bajo la tradición de la ley común anglo-americana. Históricamente, las culturas tribales europeas que inventaron ésta tradición mejoraron sus sistemas de resolución de disputas moviéndose de un sistema de costumbres semi-conscientes a un sistema de ley consuetudinaria memorizada por los hombres sabios, y eventualmente escrita.

Quizás, a medida que nuestra población ascienda y la aculturación de nuevos miembros se vuelva más difícil, es tiempo que la cultura hacker haga algo análogo. Para desarrollar código para resolver los varios tipos de disputas que pueden surgir en conexión con los proyectos de código abierto, y una tradición arbitraria en la cual los miembros más experimentados de la comunidad puedan ser consultados para mediar disputas.

El análisis en éste ensayo siguiere como debe ser ese código, haciendo explícito eso que antes era implícito. Esos códigos no pueden ser impuestos desde arriba; tienen que ser adoptados voluntariamente por los fundadores o propietarios de los proyectos. No pueden ser completamente rígidos, ya que las presiones de la cultura tienden a cambiar con el tiempo. Finalmente, tienen que reflejar un amplio consenso de la comunidad hacker.

He empezado a trabajar en ese código, titulado tentativamente “El Protocolo Malvern”, en honor a la pequeña ciudad donde viví. Si el análisis general en éste ensayo se vuelve suficientemente aceptado, haré “El Protocolo Malvern” públicamente disponible como un código modelo para la resolución de disputas. Las partes interesadas en criticar y desarrollar éste código, o simplemente ofrecer sugerencias, están invitados a contactarme por e-mail.

21 – PREGUNTAS PARA FUTURA INVESTIGACIÓN

El entendimiento de la cultura (y el mío propio) de proyectos largos que no siguen un modelo de dictador benevolente es débil. La mayoría de esos proyectos fallan. Unos pocos se vuelven espectacularmente exitosos e importantes (Perl, Apache, KDE). Nadie entiende realmente donde radica la diferencia. Hay un vago sentido que cada proyecto es *sui generis* y se mantiene o cae en la dinámica de los miembros del grupo, pero es esto cierto o hay estrategias replicables que un grupo puede seguir?

22 – BIBLIOGRAFÍA

- *Bloodtaking and Peacemaking: Feud, Law, and Society in Saga Iceland*; Miller, William Ian
- *Principia Discordia, or How I Found Goddess and Waht I Did To Her When I Found Her*; Malaclypse the Younger
- *The adapted mind: Evolutionary psychology and the generation of culture*; J. Barkow, L. Cosmides, and J. Tooby
- *The Attention Economy and the Net*; Goldhaber, Michael K.

23 – AGRADECIMIENTOS

Robert Lanphier contribuyó a la discusión del comportamiento humilde. Eric Kidd remarcó el rol de valorar la humildad para prevenir cultos de personalidad. La sección de los efectos globales fue inspirada por comentarios de Daniel Burn. Mike Whitaker inspiró el hilo principal en la sección de la aculturación. Chris Phoenix señaló la importancia del hecho que los hackers no pueden ganar reputación haciendo que otros hackers la pierdan.

Yo soy el único responsable de lo que hay en éste ensayo, y sus errores y concepciones. Sin embargo, he acogido comentarios y sugerencias, y las he usado para mejorar éste ensayo. Un proceso que no espero que termine en algún tiempo predefinido.