SOFTWARE LIBRE Y EL TRABAJO INFORMÁTICO EN EL CAPITALISMO DEL CONOCIMIENTO

Pablo Miguez
Universidad de Buenos Aires, Argentina

Resumen.- En este trabajo analizo las consecuencias del trabajo inmaterial en la producción de software en el marco de la nueva ciudadanía digital. Estudio las relaciones entre los trabajadores informaticos en la producción de software y servicios informáticos a partir de la comparación entre dos modelos de desarrollo de software, el inspirado en el software libre y el que deriva del software «propietario». El trabajo de los trabajadores informáticos fue analizado desde la teorización previa sobre el movimiento del sofware libre. Algunos de sus principales exponentes hacen un uso singular de las categorías de la antropología económica, cuya pertinencia es peusta en duda a la luz del análsis de la organización del trabajo informático: El trabajo informático consiste en el desarrollo de software, esto es, la construcción de programas como producto final, donde conviven equipos, proyectos y redes. El proceso de trabajo reconoce importantes transformaciones con las tecnologías de la información que dan lugar a un nuevo tipo de ciudadanía digital, pero la explotación, lejos de desaparecer, se ha acomodado a las nuevas circunstancias.

En este trabajo me ocupo de las relaciones entre los trabajadores informaticos en la producción de software y servicios informáticos a partir de la comparación entre dos modelos de desarrollo de software, el inspirado en el software libre y el que deriva del software « propietario ».

Hasta los años ochenta la revolución informática era fundamentalmente una realidad visible en Estados Unidos, en el ámbito de las universidades y laboratorios. Los años noventa fueron los años de la revolución informática en el mundo, acelerada por la consolidación de Internet. Pusieron de una modalidad de trabajo que era propia de los ámbitos académicos e institucionales que le dieron origen. Yochai Benkler asegura que Internet es un espacio construido bajo tecnologías que permiten construir espacios compartidos, en donde la inteligencia es aportada por los usuarios finales más que por núcleos centrales inteligentes (Benkler, 2003).

Mientras el modelo Microsoft de software propietario se basa en los “nuevos cercamientos” el conocimiento el modelo del software libre es un modelo tecnoinstitucional abierto y cooperativo, propio de los nuevos bienes públicos informacionales (Jollivet, 2007). Analizaremos ambos modelos en este trabajo. En el caso del software libre para mostrar que algunas posiciones dentro del

campo echan mano de las categorías de la antropología económica para legitimarse. En el caso del software propietario, para mostrar las inconsistencias de las teorizaciones habituales que tienden a idealizar los nuevos tipos de trabajo y para sostener la pertinencia del enfoque etnográfico para analizar las prácticas y los sentidos que tiene para los trabajadores informáticos.

El movimiento del software libre

Los trabajos sobre el software libre se producen asiduamente desde los años ochenta, sobre todo por parte de Richard Stallman, fundador de la Free Software Foundation. Otro referente importante para este movimiento es Eric Raymond, fundador de Open Source. Sus textos canónicos *Cultivando la noosfera* y *La Catedral y el bazar* son fundamentales para entender la lógica que inspira a los trabajadores informáticos, al menos en lo que atañe al desarrolo de soluciones informáticas. Sobre su trabajo nos ocuparemos en las siguientes páginas.

Estos textos fueron escritos a partir de la experiencia de primera mano del autor como programador de software. Sin mostrar interés antropológio ni prentender realizar una etnografía, el autor describe las relaciones entre quienes se ocupan del desarrollo del software libre, al que caracteriza como una *economìa del don.*

Richard Stallman**,** fundador en 1985 de la Free Software Foundation creó una comunidad de producción e intercambio de software basada en la cooperación y con el objetivo expreso de eliminar las restricciones a la copia o modificación de los programas de computación que en el contexto patentista norteamericano pretendían transformar todo el código “abierto” en código “propietario”. Desarrolló un sistema operativo diferente a Unix (aunque basado en él, GNU significa “GNU no es Unix.”), que permite un uso libre del software, esto es, estudiar el programa, copiarlo y mejorarlo con la condición de que quede abierto para el uso del público. La licencia GPL surgió como un “hack” al sistema anglosajón de copyright y se denominó *copyleft*. El movimiento del software libre nace para oponerse a la comercialización del software en general. El desarrolo del movimiento reconoce un salto notable cuando Linus Torvalds, estudiante finlandés, escribe un kernel o núcleo del sistema operativo compatible con los kernels Unix del software propietario. Torvalds llenó un vacío en el proyecto GNU, que se empezo a llamar GNU/Linux y dejó de estar confinado a los expertos, lo que atrajo la atención de las grandes empresas de informática como IBM, HP, Sun Microsistems o Apple.

En el campo del desarrollo del software podemos encontrar dos corrientes relativamente enfrentadas, la corriente del « software libre » y por el otro, la iniciativa Open Source, encabezada por Eric Raymond, más interesada en los beneficios tecnológicos y económicos que se derivan del código abierto.

La postura de Raymond, si bien cuestiona el software propietario, lo hace por razones diferentes a las de Stallman. Raymond no se opone, en principio, a las corporaciones de la industria informática. Para él, el software propietario no

facilita la innovación tecnológica porque impide el desarrollo de software de mejor calidad. Stallman, en cambio, busca defender ante todo la libertad de los usuarios de comprender el funcionamiento de los programas, aunque no fueran más eficientes.

En función de ello, Raymond hace una descripción de las tareas de los desarrolladores de software y propone una teorización que justifica el desarrollo creativo del software, con independencia de que sea libre o propietario. Para ello utiliza las categorías de la antropología económica. Por fuera del ámbito académico hace un uso no especialista de las mismas que sirven para perfomar las prácticas en el seno del movimiento.

Los usos de las categorías de la antropología económica en el estudio del movimiento del software libre

Construir una comunidad

Para Raymond iniciar un proyecto de software libre es construir una comunidad, para lo cual recomienda seguir lo que el denomina “El modelo del bazar”. Haciendo una referencia implícita al trabajo de Cleeford Geertz en Marruecos (que describe una “economía del bazar” donde la negociación en un mercado tumultuoso de objetos y prácticas sustituye la imposición por la fuerza) Raymond propone que ese esquema es el que ha inspirado e inspira actualmente la producción de software.

Raymond se propone hacer explícita la lógica de funcionamiento aparentemente anárquica de la producción de software pero no sólo a los efectos de descubrir una dinámica inconciente sino con la pretensión instrumental de mejorar dicha producción. “…mientras que la creación de programas sigue siendo esencialmente una actividad solitaria, los desarrollos realmente grandes surgen de la atención y la capacidad de pensamiento de comunidades enteras. El desarrollador que usa solamente su cerebro sobre un proyecto cerrado está quedando detrás del que sabe como crear en un contexto abierto y evolutivo en el que la búsqueda de errores y las mejoras son realizadas por cientos de personas.” (Raymond, 1997: 25)

Para algunos programadores, la producción de un software era equivalente a la construcción de un edificio. Se necesitaba construir como si fuera una catedral, planificando cuidadosamente cada paso para que un proyecto que involucra gran número de variables no se desmorone. Sin embargo, a partir de su propia experiencia personal Raymond notó que la mayor parte de los proyectos exitosos no siguieron este camino sino uno muy diferente, cuyas reglas no estaban claras pero que sin duda remiten a un espíritu de colaboración y a un tipo de trabajo que no se corresponde con la planificación anticipada de todos los elementos. La colaboración en la red se convirtió en la regla general de todos los proyectos, originando de manera más o menos espontánea una comunidad de desarrolladores.

Raymond propone explícitamente que la creación *ex profeso* de dicha comunidad es ahora una condición necesaria para el éxito del “modelo del

bazar": "Esta claro que uno no puede partir de cero en el estilo bazar. Con él, uno puede probar, buscar errores, poner a punto y mejorar algo, pero sería muy difícil *originar* un proyecto en un modo semejante al bazar. Linus no lo intentó de esta manera. Yo tampoco lo hice así. Nuestra naciente comunidad de desarrolladores necesita algo que ya corra para jugar. Cuando uno comienza la construcción del edificio comunal, lo que debe ser capaz de hacer es presentar una *promesa plausible*. El programa no necesita ser particularmente bueno. Puede ser burdo, tener muchos errores, estar incompleto y pobremente documentado. Pero en lo que no se puede fallar es en convencer a los co-desarrolladores potenciales de que el programa puede evolucionar hacia algo elegante en el futuro." (Raymond, 1997: 21)

Las características del líder del proyecto, sus calificaciones y el estado del código deben poder ser evaluadas por los potenciales miembros de la comunidad antes de decidir formar parte del proyecto. Ahora bien, la membresía no es algo que se adquiera de manera directa o automática. La comunidad hacker, dice Raymond, requiere para admitir a sus miembros que entiendan "el juego de la reputación". Como señala Raymond: "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)." (Raymond, 1999: 24). La calidad de miembro es dada solamente por los que ya están dentro de la cultura.

Cultura hacker como cultura del don

Veamos cuales son las razones por las cuales Raymond sostiene que nos encontramos ante una cultura del don. El autor sostiene explícitamente que "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*." (Raymond, 1999: 9)

Para Raymond, la cultura del don constituye un modelo aparentemente superador del modelo del mando jerárquico y del modelo del intercambio. En los primeros el lugar de los bienes escasos es decidido por una autoridad central, respaldada por la fuerza. Nuestra sociedad responde frecuentemente al segundo tipo, es básicamente una cultura de intercambio, por lo cual "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." (Raymond, 1999: 9)

Estas son las características que Raymond otorga a las culturas 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.” (Raymond, 1999: 9)Para Raymond, a diferencia de las sociedades primitivas donde se analiza el don, en la cultura hacker se compite por el prestigio en un contexto de abundancia, no de escasez: “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.” (Raymond, 1999: 10)

La “competencia por el prestigio”

Las causas principales para la búsqueda de prestigio remiten a ciertas características generales como a cierta particularidades de la cultura hacker, que hacen que esta sea la única manera de obtener “status”. En primer lugar porque para los individuos se trata de una “recompensa primaria”. En segundo lugar, porque sirve “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.” (Raymond, 1999: 11)

Más allá de estas 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.” (Raymond, 1999: 11-12)

La propiedad del proyecto de software libre

Aquel que es reconocido por la comunidad con derecho a distribuir el software y sus versiones modificadas es el propietario del desarrollo. Es el propietario de un tipo particular de bienes, que no son mas que ideas, códigos algoritmos,

etc… que no encontramos físicamente en ningún lugar sino que habitan en la noosfera.

Raymond señala que “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.” Aunque la propiedad del proyecto puede modificarse, los mecanismos para hacerlo están lejos de ser naturales. Si el fundador del proyecto pierde interés o no realiza el soporte técnico necesario y otros desarrolladores quieren hacerlo la sucesión de la propiedad es un punto a discutir. Si se desarrollaran modificaciones al proceso original y fueran puestas en la comunidad se suscitarían disputas sobre la propiedad. Pero esto es menos frecuente de lo que sugiere la idea de colaboración en la red: “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.” Además de ello “no puedes esperar ser considerado completamente legítimo hasta que hayas hecho mejoras substanciales en busca de la comunidad de usuarios del proyecto.” (Raymond, 1999: 5)

Para Raymond no debemos confundir la noosfera con el Ciberespacio. La noosfera es una especie de reservorio de ideas mientras que el ciberespacio es el espacio virtual donde estas ideas se localizan. El desarrollo del software libre implica “cultivar la noosfera”. La discusión sobre la posibilidad del patentamiento del software, de asignar derechos de propiedad intelectual sobre los programas informáticos es un tema que atraviesa estas actividades. Raymond es un defensor de la Open Source Iniciative, que busca liberar los contenidos del ciberespacio de los cercamientos que le imponen los derechos de propiedad, sin embargo también defiende una idea de propiedad para el campo de la noosfera. Raymond defiende la noción de propiedad en la noosfera y la vincula con la etología. Afirma 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." (Raymond, 1999: 19)

Las prácticas y los tabúes

Programar, "picar código" según los programadores, constituye la principal actividad de los programadores. La posibilidad de modificar el código y distribuir los programas constituye la razón de ser de los defensores del software abierto. Sin embargo, Raymond distingue diferentes tendencias al interior de la cultura hacker "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." (Raymond, 1999: 1).

En sus orígenes "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." (Raymond, 1999: 3). Pero aunque en 1997 la mitad de los paquetes de software usaban la licencia GPL de la FSF, su licencia se desarrollaba en medio de "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." (Raymond, 1999: 3).

De esta manera, Raymond establece la distinción entre "idealistas" y "pragmáticos" dentro de la comunidad La primera representa a los defensores a una postura anti-comercial y anticorporativa. La segunda representa la postura menos radical: "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." (Raymond, 1999: 3).A partir del desarrollo del sistema Linux, la posición de los idealistas se vio, según Raymond, crecientemente desfavorecida: "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." (Raymond, 1999: 3).Así es como otras comunidades empezaron a brotar de las raíces de Unix/Internet inventando sus propios proyectos, no basados en la licencia GPL. Vercelli sintetiza agudamente los términos de la disputa entre ambas posturas: "Una licencia esta infectada, según Raymond (2003) cuando requiere que cada trabajo derivado del software licenciado también sea licenciado bajo los mismos términos. La GPL es la licencia de estándares abiertos que más infectada está. Stallman (2002) ha contestado estas diferencias por dentro de la comunidad explicando que '*Free Software'* y '*Open Source*' describen, más o menos, la misma categoría de software. Sin embargo, dicen cosas diferentes sobre el software propiamente dicho y sobre los valores que su producción envuelve. Para Stallman y el proyecto GNU, el concepto *FREE SOFTWARE* sigue expresando la idea de que la libertad -y no solo la tecnología- es importante para nuestras sociedades." (Vercelli, 2004: 133)

En función de lo señalado, al analizar las prácticas concretas de los informáticos Raymond señala que éstos responden a una "teoría promiscua" del movimiento del software libre que se contrapone con una "practica puritana". Prueba de ello es la existencia de determinados tabúes que confirman indirectamente la operatividad de la teoría lockeana de la propiedad, no sólo en el plano material sino también en el campo virtual. Para Raymond, las prácticas de los trabajadores informáticos en la creación de software remiten a la teoría de la propiedad de Locke y las costumbres propietarias son "un medio de maximizar los incentivos de reputación; de asegurar que los créditos de nuestros pares vayan donde es debido y no vayan donde no es debido" (Raymond 1999:12).

En teoría, cualquier persona puede hackear u operar sobre cualquier producto de código abierto. Sin embargo, señala Raymond: "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." (Raymond 1999:3).

Prueba de ello son los tres tabúes que señala Raymond como propios de la cultura de código abierto, a saber:

1) "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." El riesgo de perder reputación "sólo lo pueden controlar estando activamente en los dos proyectos simultáneamente después de la división. (Esto generalmente sería muy confuso o difícil para ser práctico.)"

2) "Distribuir cambios en un proyecto sin la cooperación de los moderadores no se produce, excepto en casos especiales como la aportación de arreglos triviales en el software."

3) “No se quita el nombre de una persona de los créditos del proyecto sin el consentimiento explícito de la misma.” Ello equivaldría a robar el aporte de esa persona al proyecto.

Pero además de la pérdida de prestigio del grupo de desarrollo original, destaca Raymond, existen otras razones “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.”

En el segundo caso, los parches no oficiales “pueden complicar el seguimiento de bugs enormemente, e infligir trabajo en los mantenedores, que ya tienen suficiente en corregir sus propios errores.” (Raymond 1999:12).

Según Raymond, a pesar de que las manifestaciones públicas de los hackers cuestionan el egoísmo y las motivaciones basadas en el ego y hacen un culto de la humildad de manera manifiesta, ocultando sus verdaderas inclinaciones hacia el honor o prestigio individual. Reconocer esto corrompería las bases del espíritu creativo y cooperativo. Por lo tanto la humildad redunda en una mayor productividad.

Como se valora un don o regalo

Raymond señala que existen reglas para la valoración de las soluciones informáticas dentro de la comunidad. La primera es que el programa funcione tan bien como se esperaba, aunque puede tener errores o bugs. Segundo, un trabajo original, que “extiende la noosfera” es mejor que duplicar una pieza existente de territorio, salvo que se trate de un software “cerrado”.Tercero, el trabajo que está en una distribución, que circula, es mejor que el que no lo hace “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)” En cuarto lugar, el trabajo usado por muchos es mejor porque significa que las soluciones alternativas no funcionan igual de bien. Quinto, “la continua devoción al trabajo difícil y aburrido (como el debugging, o escribir documentación) es más prestigioso que hacer las cosas divertidas y fáciles”. Y finalmente “Las extensiones de funciones no triviales son más prestigiosas que los parches de bajo nivel y el debugging.” (Raymond 1999:18).

Propiedad, estructura del proyecto y resolución de conflictos

Para evitar estos conflictos Raymond propone una estructura de proyectos basada en el modelo del “dictador benevolente” que, como veremos luego, sería el modelo predominante también en la producción de software propietario: “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." (Raymond 1999: 21)

Se espera que el dictador consulte sus decisiones con los co-desarrolladorres: "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 estas formas u otras, los co-desarrolladores son los contribuyentes que hacen una continua y substancial inversión de tiempo en el proyecto (Raymond 1999: 22).

Productividad en el proyecto de SL

Después de toda la argumentación aparece el verdadero fundamento de las tesis de Raymond: "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." (Raymond 1999: 25) Las razones para apoyar el software no propietarios están ligadas a la productividad, no a la impugnación de un sistema que se sustenta en la privatización de la colaboración común "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." (Raymond 1999: 25)

Y, según Raymond, esto resulta de los propios intereses de los programadores "La "función utilidad" que los hackers de Linux están maximizando no es económica en el sentido clásico, sino algo intangible como la satisfacción de su ego y su reputación entre otros hackers. (Uno podría hablar de su "motivación altruista", pero ignoraríamos el hecho de que el altruismo en sí mismo es una forma de satisfacción del ego para el altruista). " (Raymond, 1997: 26). De alguna manera, el individualismo sigue presente y es potenciado por la comunidad: "Pienso que el futuro del software libre será cada vez más de la gente que sabe como jugar el juego de Linus, la gente que deja atrás la catedral y abraza el bazar. Esto no quiere decir que la visión y la brillantez individuales ya no importen; al contrario, creo que en la vanguardia del software libre estarán quienes comiencen con visión y brillantez individual, y luego las enriquezcan construyendo positivamente comunidades voluntarias de interés." (Raymond, 1997: 26)

Algunas consideraciones sobre el trabajo de Raymond

De este recorrido por el trabajo de Eric Raymond podemos advertir el uso de las categorías de la antropología económica no sólo para intentar dar cuenta sino también para legitimar las prácticas observables en la producción de software. Así lo percibe uno de los miembros mas prominentes de la comunidad del software libre. Sin embargo, la proximidad del análisis de Raymond con las teorías liberales que defienden la competencia (Adam Smith) y la propiedad (Jhon Locke) ameritan prestar cuidadosa atención de los abusos a los que se presta la recuperación de la idea del don. La antropología económica se constituyo justamente por oposición a estas miradas reduccionistas de la complejidad social. Raymond se apropia de las categorías utilizándolas de una manera descontextualizada, haciendo un uso que no interroga el punto de vista de los sujetos involucrados

Llama la atención que el autor remita a la tradición económica neoclásica para referirse a la conducta de los desarrolladores. La Escuela neoclásica defiende la existencia de agentes racionales y atomísticos que maximizan funciones objetivos “la utilidad” a partir de las dotaciones presupuestarias que poseen, indendientemente de su contexto social o cultura, lo que no se condice con las consideraciones de un individuo cuya practicas están inmersas en un mar de condicionamientos y significaciones sociales. Si Polanyi, en referencia a la economía clásica de Smith y Ricardo, cuestionaba “el surgimiento del credo liberal” y la noción de “mercado autorregulado” actualmente no podemos más que impugnar la referencia a los economistas neoclásicos para caracterizar la interacción de los individuos fuera del campo económico, e incluso dentro de él.

Asimismo, el autor nos remite al marco teórico liberal para justificar la apropiación de los bienes de la tecnoesfera. El rescate del liberalismo clásico responde a que Locke entraba su defensa del derecho de propiedad en base al trabajo. Sin embargo, en filosofía política es conocido el hecho de que la tradición contractualista naturaliza la propiedad como un derecho individual. Esto se contrapone con la idea de que la generación de ideas, conocimientos proviene del “común”. La noción marxista de *Generall Intellect* nos parece mucho más fructífera para analizar estas nuevas dimensiones de la apropiación (Benkler, Negri). Proponemos un acercamiento al campo económico desde la crítica de la Economía Política afín a nuevas corrientes que, provenientes del marxismo, son potentes para explicar las características del trabajo en el siglo XXI. (Negri, Virno)

Aunque no se trate de un antecedente válido, el trabajo de Raymond no invalida la posibilidad de acercarse a las prácticas de los desarrolladores de sorftware libre desde una perspectiva etnográfica, que puede ser de gran utilidad para abordar este tipo de relaciones. Estas consideraciones sobre la pertinencia etnográfica pueden trasladarse de la misma manera al estudio del proceso de trabajo informático “propietario”, no libre, y queda pendiente, sin embargo, analizar si debemos echar mano también de las enseñanzas de la sociología del trabajo y de la Critica de la economía política, en un diálogo que se muestra potencialmente fructífero. En este trabajo me propongo señalar la potencialidad de tender puentes en esa dirección, lo que refuerza la percepción de Florence Weber de que la antropología económica, lejos de estar confinada a las sociedades primitivas, tradicionales o no occidentales, puede dar cuenta

de la complejidad que envuelve el trabajo informático, elemento central de las nuevas tecnologías de la última fase del capitalismo avanzado (Weber, 2009: 29-34).

Organización del trabajo informático: Equipos, proyectos y redes

El trabajo informático consiste en el desarrollo de software, esto es, la construcción de programas como producto final. La singularidad de este producto, que lo diferencia de las mercancías tradicionales, es su carácter inmaterial. Pero ello no impide que al igual que los productos tradicionales ofrecidos por la industria sean susceptibles de ser estudiados a partir de las etapas que comprenden su "ciclo de vida", expresión muy utilizada en informática. Estas etapas son: la toma de requisitos del cliente, el diseño, la arquitectura de software, el análisis funcional, las pruebas parciales y de conjunto, la aplicación y el mantenimiento (Castillo, 2009). Por otro lado, otra característica fundamental es que se suele trabajar en *equipos* de trabajo, en relación con *redes* y en función de diferentes *proyectos*, por lo cual la colaboración en red se vuelve central.

La operación de coordinar diferentes "recursos" en la búsqueda de un objetivo determinado y por un período de tiempo limitado justifica la aparición de la figura de los "líderes del proyecto".La figura del jefe de proyecto asume las características del "manager", que no da órdenes ni espera las consignas de la dirección, es el "hombre de las redes"

Sobre la organización "horizontal" del trabajo

El trabajo en la programación informática se muestra como ejemplo de una organización menos jerárquica donde circula el conocimiento. Ellos se suele adjudicar a la naturaleza del producto.

En general, escribir programas significa resolver docenas de diferentes problemas "nuevos",ordenar valores, disponerlos en estructuras eficientes de memoria, entregar interfaces de usuario claras, equilibrar la carga de un conjunto de procesadores, intercambiar datos entre ordenadores remotos, leer y escribir diferentes formatos de datos y muchas otras tareas.

Para Martín, que programa desde los quince años, programar es "picar código", algo que requiere cada vez más exigencia pero el poner en juego cada vez más habilidades no implica adquirir mayores derechos sobre el producto de su trabajo:

Martín: *La diferencia son principalmente las herramientas que uno tiene como desarrollador. Hoy en día son herramientas más ágiles, son mucho más complejas, permiten atacar… hacer programas complejos. Un programa complejo no puede tener una solución simple, o sea, con las herramientas que hoy en día tenemos disponibles. Antes había otro tipo de programación, no era programación orientada a objetos, por ejemplo…*

Pablo: *¿Hoy para programar un desarrollador tiene más exigencia?.*

Martín: *Mucha más exigencia.*

Pablo: *¿Y eso siempre se incrementa, no hay manera que no sea así?.*

Martín: *Las tecnologías van evolucionando*.

Pablo: ¿Y el tiempo de formación? ¿Es más fácil aprender?.

Martín: *No, no, para nada.*

Pablo: *¿Cómo se compatibiliza… estudian más ahora que antes?.*

Martín: *Lo que pasa es que ahora hay mas información, hay mas medios o canales para que cada uno que quiera ser desarrollador lo haga por sus propios medios, incluso las empresas para reducir la curva de aprendizaje ponen personas disponibles para la capacitación a nivel básico, de sus metodologías de trabajo, de las herramientas que ya tiene preparadas para re-utilizar. Es como tener piezas o cajas negras o sistemas...hay que utilizar esas cosas que ya tenemos armadas, que están garantizadas, que tiene calidad, que sabemos que funcionan.*

Como en cualquier otro trabajo, Martín pone a disposición del empleador su "capacidad lógica" y no le parece que el producto de su trabajo le deba pertenecer:

Martín: *Estás trabajando para tu empleador…todas las líneas de código que estas escribiendo no te pertenecen…vos nada más estas vendiendo tus conocimientos y lo plasmas en forma de software… tu capacidad lógica… para generar líneas de código … si no lo haces vos lo hace otro.*

Pablo: *¿Por qué? Digo… si cada uno es distinto.*

Martín: *Cada uno lo puede hacer de distintas maneras, de ahí que la solución de uno sea brillante y la solución de otros sea menos brillante.*

Pablo: *¿Y nunca te afecta saber que vos generaste esa resolución del problema y no es tuya?.*

Martín: *No, porque justamente un desarrollador se dedica a responder por la demanda de su empleador.*

Desafíos, rotación y liderazgo: las nuevas formas de explotación (y las viejas)

En el sector es habitual la rotación en los niveles inferiores (junior) de desarrollo, lo que se contrapone con la supuesta "implicación" promovida desde los jefes de proyecto. Para Daniel, jefe de una pequeña empresa de desarrollo de menos de 10 empleados, la división del trabajo supone contratar trabajadores formados y con capacidades que trasciendan el aspecto meramente técnico:

Daniel: *Primero está el análisis, relevamiento, encuestas, herramientas para hacer un buen análisis, problemática asociadas. Luego esta el diseño y esa*

parte es más costosa, mucho mas costosa, requiere más recursos, mejores recursos.

Pablo: *¿Qué son los recursos?.*

Daniel: *Necesito más programadores, buenos analistas que tengan buen feeling con el cliente, trato con la gente, porque ellos son los que están con la persona que tiene el problema.*

Sin embargo, considera que la rotación es inevitable y que el liderazgo define el éxito del proyecto:

Daniel: Tiene mucho que ver los equipos de trabajo, los líderes.

Pablo: ¿Los líderes de proyecto?.

Daniel: Si. Los líderes de proyecto... Todos lo miran a él como un líder entonces hay que trabajar en eso para mantener motivada a los equipos de trabajo, para llevar adelante los equipos de trabajo.

Pablo: ¿Cómo trabaja alguien que es un joven, que es investido como un líder y esta con alguien más grande, que está hace mas tiempo….

Daniel: El liderazgo se genera generalmente en este rubro por mayor conocimiento.

Alguien que se destaca como líder tiene conocimiento y autoridad porque tiene más conocimiento que otro.

Para Daniel, para evitar la rotación la tarea de un buen líder es plantear "desafíos" a la altura del trabajador:

Pablo: *O sea, el desarrollador es más fácilmente reemplazable que el analista.*

Daniel: *Si, totalmente. Teniendo bien organizada la parte de desarrollo en una empresa que se dedica al desarrollo de software…los desarrolladores son los que mas rotan.*

Pablo: *¿Por qué rotan? ¿Porque les ofrecen otros trabajos?.*

Daniel: *Por muchos motivos, uno es porque no tiene la motivación suficiente en cuanto al trabajo, en cuanto a la calidad del trabajo que están haciendo. Eso puede ser un motivo…Ellos quieren hacer otras cosas, usar otros lenguajes, quieren aprender otras cosas. Si no tiene desafíos interesantes, por ejemplo, sino tienen que hacer programas para bancos, empresas multinacionales, o sea, desafíos interesantes…los programadores se van…*

Según Daniel, el líder condiciona el éxito del proyecto pero éste debe seleccionar a trabajadores que acepten la tarea "desafiante" y al mismo tiempo, no se distraigan en su realización. Para Carlos, manager de una empresa de desarrollo de menos de 10 empleados, el trabajo de sus empleados esta muy bien remunerado, pero reconoce que nunca lo haría:

Carlos: *Es algo que se aprende, un pibe en tres años está programando y está ganado un salario que un médico…que hay médicos que nunca lo vana a ganar…Todos los médicos no ganan 3000 magos por mes, seguramente… algunos ganaran más pero en promedio ganan menos….*

Pablo: ¿Pero no un poco tedioso el trabajo para un joven?.

Carlos: *Yo no lo haría…ni loco, el de analista, bueno si me pagan demasiado bien capaz que en algún momento lo haría…*

Pablo: ¿Por que no lo harías?.

Carlos: *Porque estas sentado frente a una computadora escribiendo algo todo el tiempo, todo el tiempo, que se yo… está haciendo algo con una sola interacción, de echo, cuanto menos interactúe con gente durante el proceso de trabajo mejor porque así no se desconcentra. Si el tipo esta pensando en escribir una línea, no puede hablar con alguien, interactuar con alguien para hacer su trabajo, después si, en una unidad de desarrollo. "mira esto es lo que da" pero una vez que me dieron lo objetivos tengo que escribirlos y escribirlos lo más rápido posible, y bien, documentarlo…*

Para Carlos, hacer bien el trabajo -"picar código" diría Martín- requiere la menor distracción posible y escasa interacción. A pesar de tratarse de un sector muy dinámico, en esto no se distingue de otros trabajadores del trabajo industrial. La rotación no se relaciona con la falta de "desafíos" sino con el exceso de demanda de trabajadores y los relativamente altos salarios del sector. Para Daniel, sin embargo, aunque reconoce que la rotación es un problema, más importante es encontrar un rápido reemplazo para los programadores que se van. Aquí también, propone métodos de gestión no tan diferentes a los de cualquier otro sector, sea o no tecnológicamente dinámico.

Pablo: *Estamos hablando de una empresa de 20 es una empresa que es una pyme, sumamente pequeña me imagino que el espacio es un espacio chico…un oficina.*

Daniel: *Si cuando se va un recurso el impacto es mucho mayor…no se, dos personas por semestre se van…*

Pablo: *¿Cómo planificas a largo plazo trabajo de equipo cuando sabes que son equipos muy pequeños con muchísima rotación?.*

Daniel: *Y eso… cada empresa tendrá su estrategia, será parte del Plan estratégico de cada empresa. Hay que apostar mucho a las redes, a quienes uno establece como líderes de un proceso de desarrollo, una metodología que permita hacer fácil el reemplazo de los desarrolladores, tratar de mantener a los desarrolladores junior, si?. Que sean fácilmente reemplazables porque son menos indispensables. Eso como lo haces? Haciendo herramientas, utilizando métodos de desarrollo y trabajo en equipo donde le conferís pocas responsabilidades a los desarrolladores junior, son más bien trabajos automatizados lo que hacen los desarrolladores, les quitas toma de decisiones*.

Pablo: *¿Eso no puede hacer que efectivamente se aburran, que se desmotiven rápido?*

Daniel: *Pero mientras ellos sigan trabajando van a ir creciendo y van a llegar a ser líderes en algún momento…*

El proceso de trabajo reconoce importantes transformaciones con las nuevas tecnología de la información, pero la explotación, lejos de desaparecer, se ha acomodado a las nuevas circunstancias. En su trabajo sobre los trabajadores de la industria de videojuegos Dyer- Whiteford señala las características principales de los nuevos proceso de trabajo vinculados a la industria

informática: “Concebir, escribir y programar mundos virtuales requiere una síntesis de capacidades narrativas, estéticas y tecnológicas: desarrollar los conocimientos reunidos por el programador digital, el diseñador gráfico, el que testea el software, el escenógrafo, el animador, el técnico de sonido y de música”. Y agrega “En este “trabajo inmaterial”, que es cabalmente incompatible con las técnicas de gestión tayloristas/fordistas, la industria del juego es la arena central para la experimentación del trabajo en equipo, el liderazgo carismático, los empleos de tiempos ultraflexibles, las oficinas abiertas, las jerarquías suaves, las *stock options*, una gestión participativa de los recursos humanos y un ethos del “trabajo como juego”. Esto implica una dirección *soft*, cooptación *cool* y explotación mistificada, con horarios sin fin, agotamiento físico y mental e inseguridad crónica, organizada fuera de toda tradición sindical y de protección obrera estable.”(Dyer- Whiteford , 2004:53).

A pesar de que todavía la investigación no ha terminado, a partir de algunas entrevistas y observaciones en el campo pensamos que la lógica propietaria de producción de software se muestra bastante coherente con la propuesta de Raymond. La gestión de equipos de trabajo en la producción para el mercado muestra grandes similitudes con la forma en que Raymond describe a la comunidad del software libre. La lucha por el prestigio propia se transforma, más que en mejores salarios (que son relativamente altos) en autosuperación y nuevos desafíos. El éxito de los proyectos remite alas condiciones del líder y a su prestigio basado en el conocimiento. Sin embargo nos planteamos sino la propuesta de Raymond no es una manera de legitimar una postura que antepone la eficiencia en la producción a la libertad de hacer uso del conocimiento, sea o no más eficiente.

La pertinencia del enfoque etnográfico y el dialogo con otras disciplina

A la luz de estas observaciones creemos que la etnografía se presenta como un método adecuado para dar cuenta de las prácticas dentro del movimiento del software libre y en la producción de software propietario. La perpectiva etnográfica no se centra en la tecnología sino en los usos y la construcción de sentido alrededor de ella. (Hine : 2008). Las actividades, las relaciones y las significaciones que se forjan entre quienes participan en los procesos sociales ligados al mundo del trabajo informático, que se pueden ser tácitas o darse por supuestas, pueden hacerse explícitas. Las formas de construir sentido en el proceso de trabajo informático se articulan directamente con las prácticas no sólo técnicas asociadas a las nuevas tecnologías. (Hine : 2008). Las aseveraciones sobre estas tecnologías presentes en los medios de comunicación, revistas especializadas y publicaciones académicas no suelen penetrar en las complejas relaciones y significaciones que se entretejen en el proceso de trabajo, donde en buena medidad el trabajo trasciende el plano físico y donde el producto es inmaterial.

De esta manera, la etnografía aparece como un método particularmente útil para analizar las relaciones entre los trabajadores informáticos, tanto en la producción de software libre como en el campo del software propietario. De alguna manera, haciendo un uso no especializado de la categorías de la

antropología económica los textos de Raymond apuntan en esa dirección, para dar cuenta del trabajo informático así como para legitimar sus consideraciones al interior del movimento.

En este trabajo intentamos señalar la potencialidad de tender puentes entre la etnografía económica, la sociología del trabajo y de la Critica de la Economía Política, en un diálogo que se muestra potencialmente fructífero. Arturo Escobar se preguntaba en 1994 "¿podría ser posible dar cuenta etnográfica de la multiplicidad de prácticas asociadas con las nuevas tecnologías, en diversos contextos sociales, étnicos y geográficos? ¿De qué manera estas prácticas se relacionan a temas sociales más amplios como por ejemplo, el control de la mano de obra, la acumulación de capital, la organización de modos de vida y la globalización de la producción cultural?" (Escobar, 1994: 22) Las categorías que los trabajadores informáticos ponen en juego y movilizan en sus prácticas ponen a dialogar a la etnografía con los estudios del trabajo, y - en tanto son los trabajadores actores económicos fundamentales- con la economía (Dufy y Weber, 2009:12.)

Bibliografía

(2003) BENKLER, Yochai, "La Economía Política del procomún" en NOVATICA/UPGRADE, mayo/junio de 2003, nº 163, pp. 6-9, ATI, Madrid

(2009) CASTILLO, Juan José, "Las fábricas de software en España: organización y división del trabajo. El trabajo fluido en la sociedad de la información" en *Trabajo y Sociedad*, Nº 12, Vol XI, Otoño 2009, Santiago del Estero, Argentina

(2004) DYER- WHITEFORD, Nick, "Sobre la contestación al capitalismo cognitivo. Composición de clase de la industria de los videojuegos y de los juegos de ordenador" en Moulier Boutang, Yann, Corsanni, Antonella, y Lazzarato, Maurizio y otros (2004): *Capitalismo cognitivo, propiedad intelectual y creación colectiva*, Traficantes de sueños, Madrid.

(2009) DUFY, Caroline y WEBER, Florence, *Más allá de la Gran División. Sociología, economía y etnografía*., Ediciones Antropofagia, Buenos Aires.

(2005) [1994] ESCOBAR, Arturo, "Bienvenidos a Cyberia.Notas para una antropología de la Cibercultura.", *Revista de Estudios Sociales* Nº 22, diciembre de 2005, Universidad de Los Andes, Colombia, 15-35.

(2001) HINE, Christine, *Etnografía virtual*, Editorial UOC, Barcelona.

(2007) JOLLIVET, Pascal: "Los rendimientos crecientes de adopción creativa ¿Hacia una competencia entre dos modelos tecnoinstitucionales en el sector software?" en (2007) RIVERA RIOS, Miguel y DABAT, Alejandro (coords.): *Cambio histórico mundial, crecimiento y desarrollo.*, Universidad Nacional Autónoma de México, México DF.

(1997) RAYMOND, Eric: "La catedral y el bazar", Publicación electrónica disponible en http://biblioweb.sindominio.net/telematica/catedral.html.

(1999) RAYMOND, Eric: “Cultivando la noosfera” Publicacion electrónica disponible en http://www.geocities.com/jagem/noosfera.html

(2004) VERCELLI, Ariel: *La conquista del Ciberespacio: Creative Commons y el diseño de entornos digitales como nuevo arte regulativo en Internet.*, Publicacion electrónica disponible en http://www.arielvercelli.org/lcsdc.pdf, Buenos Aires.

(2006)YOGUEL, Gabriel, ERBES, Analía, y ROBERT, Verónica: “El sendero evolutivo potencialidades del sector de software en Argentina” en (2006) BORELLO, José, ROBERT, Verónica y YOGUEL, Gabriel (comps.): *La informática en Argentina. Desafíos a la especialización y a la competitividad*., UNGS - Ed. Prometeo, Buenos Aires.