Software, dilapidación y ¿transgresión?

Dentro de la historia de la computación, la década de los ochenta es conocida por la lucha incesante entre tres gigantes tecnológicos: IBM, Apple y Microsoft. La primera batalla se libró entre los dos Steve (Wozniak y Jobs) e IBM: la Apple II y el concepto de *home computer* y la IBM PC y la idea de la *personal computer* (Dormehl, 2017). Durante esta confrontación Microsoft decidió concentrarse en lo que hacía mejor —el desarrollo de *software*— mediante una maniobra muy sencilla: conservar los derechos de patentes y de *copyright* de su código.

Hasta cierto punto Apple sintió simpatía con Microsoft no solo porque Bill Gates decidió no meterse en la batalla por la dominación del *hardware*, sino también por la ayuda que prestó para el desarrollo del sistema operativo de la Apple II. Al final, los Steve veían a su compañía como una dedicada a los dispositivos físicos lo cual los hizo un rival directo de una empresa que durante décadas se había estado dedicando al procesamiento de datos: IBM (Dormehl, 2017).

Sin embargo, en poco tiempo Apple se dio cuenta que la estrategia de Bill Gates no era ser un aliado en su lucha contra IBM. A la par que Microsoft ayudaba a Wozniak con el *software* para la Apple II, también se estaba en proceso de crear un nuevo sistema operativo para la IBM PC. Mientras que la IBM PC y

su sucesora —la IBM PC DOS— ganaron terreno, Steve Jobs empezó a sospechar de las intenciones de Gates pero ¿qué podía hacer? Al final Microsoft no se dedicaba al *hardware* y su nuevo sistema operativo para IBM era un producto distinto al que tenían sus Apple II.

Para continuar la lucha, Apple decidió formalizar sus relaciones con Microsoft para su primera computadora con interfaz gráfica. En sus instalaciones los Steve le enseñaron a Gates lo que sería la Apple Lisa. Apple deseaba dotarla de una serie de aplicaciones para ofrecer un mejor producto que IBM. Ambos Steve pensaron que Microsoft podía ayudar. Bill Gates aceptó y así fue como Word y Excel llegaron a Mac (Dernbach, 2012).

IBM y Apple veían en el *hardware* el elemento primordial para la hegemonía tecnológica, por lo que no les pesaba revelar elementos de su *software* a sus colaboradores. Con la valiosa apertura dada a Microsoft, Gates empezó a idear un nuevo sistema operativo con interfaz gráfica de ventanas cuya estrategia sería que su propiedad intelectual dejaría ser un respaldo jurídico para convertirse en la punta de lanza de su negocio.

En 1985 se desató la segunda batalla con la llegada de Windows 1.0. Esta confrontación se dio entre Apple y Microsoft y en el terreno del *software*. Según Jobs, Microsoft había plagiado la interfaz gráfica de Apple Lisa. Gates de manera astuta le hizo saber a Apple que, de proceder la demanda, Microsoft dejaría de licenciar Microsoft Office para Mac (Dernbach, 2012). Así empezaría una rivalidad que aún hoy en día continúa...

La guerra por las computadoras personales pasó del *hardware* al *software*. No solo definió el *who's who* dentro de la industria tecnológica de los ochenta, también marcó la pauta de lo que se conocería como la cultura de Silicon Valley: la propiedad intelectual como fundamento para el desarrollo tecnológico.

Durante los setenta y con la llegada de las primeras computadoras a las universidades, las patentes y el *copyright* se consideraban una cuestión secundaria al desarrollo de *software*. Generaciones de programadores fueron educados bajo la idea de que el código era un bien común; la propiedad intelectual se restringía a los componentes de *hardware* de las computadoras que estaban

en los laboratorios de las universidades.

Pero la lucha entre IBM, Apple y Microsoft cambió el panorama universitario. Bill Gates demostró el éxito económico de la gestión de la propiedad intelectual sobre el *software* pese a que ya durante algunos años tanto IBM como Apple habían empezado a proteger el código de sus máquinas. Con este gran sobresalto los programadores tuvieron que elegir: continuar compartiendo su código a otros programadores —un fundamento de lo que después se conocería como «ética *hacker*»— o ser fichados por alguna empresa y hacer que su código fuese «protegido» por derechos de propiedad intelectual (Stallman, 2016).

En el desarrollo de *software* la «protección» significa la transformación del código en mercancía en cuya venta rara vez se otorga el código fuente. Ante esta falta de acceso, a los programadores solo les es posible estudiar o modificar un programa mediante un tortuoso proceso de ingeniería inversa, el cual corre el peligro de ser truncado por las *violaciones* a la propiedad intelectual que esto conlleva.

Ante estas dificultades, varios programadores empezaron a caer en cuenta que la libertad de compartir el código no era un carácter dado dentro de su campo, sino un supuesto ético dentro de su misma práctica. A la par de la lucha tecnológica que produciría lo que algunos han llamado «capitalismo informacional» (Lévy, 2016), en 1983 surge un pequeño movimiento de programadores con el objetivo de mantener la «libertad» dentro del desarrollo de *software*.

En 1984 este movimiento constituiría la Free Software Foundation (FSF) cuyo programa, motivos y objetivos se concentrarían en el *El manifiesto GNU* redactado por Richard Stallman y publicado en 1985. El manifiesto no solo relata este cambio de orden dentro de los programadores, sino que, con un fuerte carácter ético —denominado «ética kantiana» o «Regla de Oro»—, indica la necesidad de cuatro libertades para los programadores (Stallman, 2016):

1. Libertad de usar el programa.
2. Libertad de estudiar el código fuente y modificar el programa.

3. Libertad de distribuir copias del programa.
4. Libertad de mejorar y hacer públicas estas mejoras del programa.

Estas cuatro libertades implican que la obtención del *software* tiene que ser tanto del código fuente como del programa ejecutable, la adquisición no tiene que darse de manera exclusiva mediante una compra, así como la propiedad intelectual no ha de funcionar como mecanismo que limite las posibilidades de del *software*. En general se estipula que el *software* no *debe de* tratarse como mercancía, sino como un bien común no solo para los programadores, sino para la sociedad entera (Stallman, 2014).

Los programadores que veían con recelo las tendencias monopólicas de IBM, Apple y Microsoft se identificaron con este movimiento. Sin embargo, con el paso de los años empezó el descontento dentro del *software* libre principalmente por el modo de trabajar de la FSF y la constante insistencia de Stallman de temas que iban más allá de la programación como las cuestiones políticas, sociales, económicas y filosóficas del *software* libre.

En los noventas empezó una fricción en la FSF cuando un grupo de programadores propuso que se dejara hablar de «libertad» y que los esfuerzos se concentraran en la «apertura» del código. Stallman y compañía les pareció una reducción peligrosa del movimiento a su ámbito pragmático ya que gran parte de lo que consideran que está en juego no tiene que ver con la programación, sino en las bases político y sociales sobre la que se da (Stallman, 2011).

Quienes apostaban por eliminar el término de «libre» en el discurso les parecía obsesivo el empeño de Stallman por hacer del desarrollo del *software* una cuestión política y social. Al final —y como la FSF aceptaría— la principal motivación del grueso de los programadores dentro del movimiento era de índole personal (Lakhani y Wolf, 2005): aprender mediante los desafíos que implica el desarrollo abierto del *software*, la obtención de un estatus más alto dentro de la comunidad de programadores y la posibilidad de obtener mayores ganancias por su trabajo —el modelo de desarrollo y de licenciamiento de la FSF no es llamativo para grandes corporaciones ya que obliga la liberación de todo

el código realizado, eliminándose la posibilidad de ganancia mediante la gestión de derechos de propiedad intelectual.

Sin poder llegar a un acuerdo y alimentado por el éxito del desarrollo de Linux, en 1997 Eric S. Raymond publicó *La catedral y el bazar*, un tendido ensayo con el que se explicitaría un distanciamiento con el modelo de «catedral» del *software* desarrollado por las empresas pero también una ruptura con el movimiento del *software* libre. En este ensayo se resalta el modelo «bazar» llevado a cabo por Linus Torvalds, el cual rompe con lo que se suponía que era la metodología necesaria para el desarrollo de *software* de alta calidad y que es el núcleo del modelo de «catedral»: un grupo pequeño de programadores que diseñan toda la arquitectura del *software* para luego empezar la tarea de programación, cuya publicación de versiones solo sería una vez que se tuviera un producto «sin errores» (Raymond, 2016).

Según Raymnond, el modelo catedrático no solo es llevado a cabo por las empresas de *software*, sino también por la FSF. Por los desacuerdos tanto en método como en intereses, *La catedral y el bazar* es el texto fundante para el surgimiento del movimiento del código abierto en 1998: la ruptura con el *software* libre y el inicio de otro movimiento cuyo principal interés es la disponibilidad del código. El programador ya no tiene que verse inmiscuido en temáticas políticas, sociales, económicas o filosóficas, sino solo centrarse a lo que mejor sabe hacer y ver en ello la posibilidad de mejorar su profesión y sus ingresos mediante una apertura con grandes corporaciones a través de licencias de *software* que tanto permiten el uso del código como mantener una ventaja competitiva mediante su comercialización.

Finales de los noventa, las conflagraciones dentro de la «revolución» digital no solo se daban entre compañías. La guerra también se desataba entre modelos distintos de desarrollo de *software*: el privativo, el libre y el abierto.

Las guerras dentro del ámbito del *software* implica la necesidad de una postura ante la gestión de la propiedad intelectual, ¿todos los derechos reservados y su comercialización, o solo algunos derechos reservados y su distribución? ¿Colaboración delimitada

por un contrato con alguna empresa, de manera «libre» pero con la restricción de que todo lo elaborado sea a su vez un producto «libre» o de modo «abierto» en donde bien se puede dar acceso al código pero también tener un modelo de comercialización «sustentable»?

Lo que esto implica es la relación del programador con su trabajo y lo que esta produce en general en un panorama económico. El quehacer del programador no es un hecho aislado de la «economía de la información», sino uno de los trabajos más valuados en este tipo de economía. Gracias al programador se hace posible que el flujo de «ideas» encuentren una «expresión concreta» en el código fuente que sirve de base para una serie de procesos informáticos que el usuario requiere.

La importancia del código es de tal envergadura que en él se manifiesta con claridad la definición estándar de la propiedad intelectual: «la expresión concreta de una idea». Y no solo eso, ya que si dentro del contexto de la «economía de la información», el «capitalismo informacional» o de la «revolución digital» el *software* y el *hardware* es el fundamento para la acumulación de capital, el código se convierte en ley (Lessig, 2009). Ley mercantil que normaliza la gestión de la propiedad intelectual como un intercambio entre el código en forma de programa o de servicio, y el dinero o las acciones.

El programador es consciente de su importancia dentro de este contexto económico hasta el punto de concebir su trabajo como un quehacer de élite y a la vanguardia con las exigencias de la época. No cualquiera tiene la habilidad necesaria para ser programador y, entre ellos, pocos son «buenos» programadores —personas con un dominio sistemático, profundo y creativo de su técnica. Además, entre los que están en las tecnologías de la información de la comunicación, los programadores sobresalen como los gurús de la era digital ya que su campo de trabajo les permite detectar —o crear— necesidades del usuario promedio que sus soluciones de *software* pretenden satisfacer. Por este motivo los fundadores de las empresas líderes en el campo tecnológico tienden a ser programadores de profesión.

Si se hace a un lado la parafernalia en torno a los casos de

«éxito» de los programadores que emprendieron la fundación de compañías, si se abandona la fascinación que ejercen estas empresas y las tecnologías que desarrollan, es posible distinguir cómo el ritmo de la industria tecnológica va a la par de una economía basada en la escasez. Por un lado la *necesidad* que tiene el usuario para que el *software* por lo menos le facilite hacer su trabajo. El programa informático se percibe como la panacea a la improductividad dentro de la oficina: adiós máquina de escribir, bienvenida computadora y *software* de ofimática (Santa Ana Anguiano, 2016). Por otro lado, la *carencia virtual* que representa la gestión de la propiedad intelectual del *software*: la solución ya está, solo hay que pagar por ella, tal como Adobe lo lleva a cabo dentro del campo del diseño gráfico y la industria editorial (Santa Ana Anguiano, 2017).

El crecimiento de la industria del *software* percibe como lógico y *natural* el pago por el uso de los programas. En este engrosamiento de la «economía de la información», el trabajo del programador se especializa y el número de programadores se incrementa a la par del crecimiento tecnológico.

Pero el crecimiento no está ausente de puntos de fuga. En varias ocasiones el usuario de *software* adquiere un programa por medio de un tercero —un familiar o un amigo que le presta su copia del programa, o en un tianguis o mediante un torrent— que no retribuye de manera directa al desarrollador. Aunque esto sea la justificación para la lucha contra la piratería, el daño por la ineficiencia en el proceso de acumulación de capital no implica una interrupción en el crecimiento de la industria tecnológica. El usuario retribuye de manera indirecta al prolongar en la práctica la relación que se ha creado entre un programador como empresario y un usuario como cliente. El usuario bien podrá no pagar por el uso de un programa o el desarrollador bien puede ofrecer su producto como un servicio gratuito, en cualquier caso el crecimiento está sustentado por la reproducción de un modo de usar el *software* y el amparo jurídico que concede la propiedad intelectual en caso de una actividad punitiva.

Bajo este contexto, es irracional e ilógico que el programador ofrezca gratuitamente su trabajo. Esta persona aniquila sus pro-

pias posibilidades de sustento, o al menos ese es el argumento de los adversarios del *software* libre o del código abierto (Stallman, 2016). ¿De qué va a vivir el programador? ¿Cómo va a reproducirse y crecer la industria tecnológica si se le impide los mismos frutos que hacen posible su gestación? El ala del *software* privativo ve en la donación del trabajo un gasto improductivo y *ajeno* a su medio. El ala del *software* libre ve en las restricciones de la propiedad intelectual el inicio de una distopía de la cual su quehacer no solo es independiente, sino que es una confrontación directa ante la inherente hecatombe social que acarrea el cierre del código (Stallman, 1999). Mientras tanto, el movimiento del código abierto sostiene una exageración entre ambas posturas ya que la acumulación de capital es posible bajo un ecosistema «sustentable» siempre y cuando se abandonen las catedrales y los programadores creen bazares (Raymond, 2016).

Pero ¿qué tan certera es la incompatibilidad entre la restricción o apertura del código en pos del crecimiento de la industria informática y, en general, a favor del engrosamiento de un sistema económico basado en la acumulación de capital? Las personas involucradas en el desarrollo del *software* privativo o libre coinciden en que sus actividades son antagónicas. El programador podrá colaborar en cualquier clase de proyecto, pero estos no pueden ser a la vez libres y cerrados. El código abierto permite una mayor flexibilidad que evita llegar a alguno de los extremos —aunque esto a veces implica caer en la ambigüedad (Stallman, 2007)—, pero su ejecución exige un cambio en la socialización entre programadores de un mismo proyecto. Por un lado tenemos la lucha ética y política de la gestión de la propiedad intelectual; por el otro, una disputa sobre la manera más eficiente y eficaz de organizar a los programadores. Al parecer estos conflictos generan distintos modos de crecimiento y distintas economías que no son compatibles al unísono.

Si la industria del *software* se trata como un «organismo vivo» (Bataille, 1987), este solo puede optar por un esquema de crecimiento o trasladar su desarrollo de manera paulatina a cada modelo de producción para así prolongar su crecimiento. Si cada postura ante el desarrollo de *software* es un organismo distinto,

el límite «inmediato» se dará entre cada uno de estos (Bataille, 1987).

En la práctica, los programadores se ven inmersos en proyectos de todas estas índoles, lo que da pie a interpretar que en la llegada al límite de cada uno de estos organismos acontece un mecanismo de adaptación por el cual la industria del *software* se «abre» gracias al *software* libre o de código abierto, así como el movimiento primordialmente social o comunitario del *software* libre o del código abierto se beneficia de su apertura con su rival privativo. Una pauta relevante para esta interpretación es la creación de la Fundación Linux, cuya comunidad está compuesta desde programadores entusiastas del *software* libre o del código abierto, hasta grandes corporaciones como IBM, Microsoft, Facebook, Twitter o Google —Apple y la FSF son los grandes ausentes.

Sin embargo, lo que se percibe como una evolución de la industria tecnológica y de la cultura de Silicon Valley, también permite una lectura distinta: el aumento de la presión dentro del ecosistema informático que desemboca en la extensión y la dilapidación.

El organismo vivo no sería la industria del *software* o el imaginario de la comunidad de programadores, sino el conjunto de estos sin coherencia sobre sus intereses económicos o posturas políticas. La posibilidad de hablar de «conjunto» está regido bajo una actividad en común: el trabajo especializado del programador cuyo producto siempre es el código.

La industria del *software* ha permitido el crecimiento de este grupo al expandir su dominio gracias al trabajo realizado por los programadores. Aquí yace una doble dependencia que atañe de manera directa al ritmo de crecimiento: la industria requiere de programadores así como estos necesitan de una industria para satisfacer sus necesidades económicas. Por otro lado, el crecimiento de esta profesión también ha dado pie a la gestación de diversas comunidades, varias de ellas a favor de la apertura del código. Allí se da paso a un doble empuje: la exigencia de cada vez mayor de que la industria socialice sus conocimientos mediante la apertura de su código a las comunidades y la deuda

inconsciente que tiene cada programador de dar acceso a los frutos de su trabajo.

La presión en este grupo se da en al menos cuatro frentes:

1. Presión de la industria y las comunidades de generar más puestos de trabajo.
2. Presión de cada programador de encontrar un trabajo que al menos subsane los enormes recursos invertidos para su formación profesional.
3. Presión de la industria y las comunidades de abrir su código para todo el ecosistema de desarrollo de *software*.
4. Presión de cada programador de liberar su código como pago de una deuda.

El entrecruzamiento de estas exigencias ha provocado un crecimiento acelerado del organismo hambriento de código. Uno de sus primeros efectos ha sido el aumento en su extensión (Bataille, 1987). Las empresas, instituciones, organizaciones o colectivos orientados al desarrollo de *software* han proliferado al mismo ritmo en que la antropofagia cancela el crecimiento de algunas comunidades en pos del crecimiento de otras, cuyo resultado final es el crecimiento general del grupo de los programadores. Aunque de manera individual un programador se muestre insatisfecho o en desacuerdo ante la vorágine de su profesión, difícilmente negará que esta lucha tiene la consecuencia de mejorar sus habilidades técnicas y sociales que lo llevan una y otra vez a combatir en la arena.

No obstante, el espacio para el crecimiento no es ilimitado. No se trata aquí de un espacio físico ni tampoco de esa entidad abstracta conocida como «ciberespacio». El lugar se da en un terreno económico y político: los mismos límites del sistema económico y de las políticas gubernamentales que sirven de base para el sustento diario de los programadores. El primer gran efecto de expansión se dio en los países desarrollados y en geografías muy bien delimitadas como lo es la bahía de San Francisco. Sin embargo, la expansión se ha dado ya en todo el globo.

Por un lado, las políticas estatales han favorecido el crecimiento —e imitación— del modelo de la «economía de la información»,

como la apuesta del gobierno de Jalisco de hacer de la Zona Metropolitana de Guadalajara el Silicon Valley mexicano (Popescu, 2017), o el aumento de la industria electrónica en China, India o Brasil. Por otro lado, la disparidad de salarios entre naciones han permitido una dispersión del capital afuera de los países desarrollados no por un compromiso social, sino por la ventaja que representa la contratación de programadores en países en vías de desarrollo ya que sus salarios son mucho más bajos que sus pares de primer mundo. Empresas estadounidenses o europeas continúan teniendo el control sobre la propiedad intelectual y los procesos de diseño y producción, mientras que empresas localizadas en América Latina, China o India ofrecen la mano de obra para el cumplimiento de los proyectos de desarrollo de *software*.

Si bien el salario de un programador en alguno de estos países en vías de desarrollo es mayor a la media nacional, solo representa una fracción de lo que costaría un programador en países del primer mundo, produciéndose así una dispersión y aumento de las plazas laborales. Sin embargo, por las facilidades ofrecidas por el aparato estatal para la gestación de este flujo de capital, la limitación de las empresas extranjeras a principalmente dar acceso en lo que concierne a la producción de código —quedando en general el personal excluido en su planificación o su apropiación— y el carácter desigual de los salarios y posibilidades de crecimiento entre programadores de primer y de tercer mundo, esta segunda «ola» de expansión puede categorizarse como la gestación de la «maquila digital». El programador en México, China, India o Brasil recibe un salario «justo» para reproducir y aumentar su calidad de vida, pero sin la posibilidad de poder ir más allá de la producción incesante de código —muchas veces con jornadas laborales de más de diez horas y en constante competitividad— o de poder trascender su propia realidad social —el salario no es suficiente para mudarse a Estados Unidos o a la Unión Europea, quedando a la intemperie de un contrato o de la fortuna de ser llamado a dar sus servicios a uno de los grandes centros tecnológicos.

Estas expansiones del organismo informacional provoca una

devaluación de su alimento: lo que antes costaba miles de dólares desarrollar ahora solo es necesario unos cientos de dólares. En los años noventa tener una página *web* era símbolo de estatus, hoy en día es tan vano que con un pequeño presupuesto es posible crearlas o, mejor aún, ¿para qué si de manera gratuita puede darse de alta una cuenta en una red social? Esto produce un hambre de innovaciones que se satisface a través de la creación de nuevas tecnologías, muchas de ellas que quizá serán olvidadas una vez caducada su novedad —guiño a Palm OS, BlackBerry, Myspace o MSN Messenger. En la actualidad, el alimento por excelencia son las aplicaciones móviles, las cuales dan pie a un nuevo campo de crecimiento para los programadores.

La innovación en el *software* produce «formas de vida cada vez más onerosas» (Bataille, 1987). La inversión en nuevas tecnologías llega a límites considerados absurdos por economías tradicionales que se basan en la producción de productos físicos. El despilfarro se da a todos los niveles: desde las empresas *top* que apuestan por la inteligencia artificial, la criptografía y la minería de datos, hasta pequeñas instituciones que utilizan sus recursos para la producción de aplicaciones poco funcionales y que son más una muestra de opulencia que un producto con un retorno de la inversión bien planificado.

Si se delimita cada una de estas organizaciones a su espacio geopolítico todas muestran la característica de ser de las industrias o comunidades más onerosas de su horizonte económico o, bien, las que no temen sacrificar su vida. En Silicon Valley las *startups* solo pueden llegar a su límite de crecimiento si apuestan por la posibilidad de una completa dilapidación de sus capacidades técnicas: no es posible ser una *startup* que hace lo mismo que un pez gordo de la bahía de San Francisco o que pretenda evitar la refriega, al menos que la muerte sea la máxima expresión del lujo (Bataille, 1987), como la multimillonaria compra de WhatsApp por parte de Facebook, o de YouTube por Google. En México, las instituciones con la capacidad de dilapidación de recursos tecnológicos no se presenta en organizaciones exiguas sino en instituciones gubernamentales —el SAT p.ej.—, universitarias —como la UNAM— o privadas —instituciones bancarias,

equipos de fútbol, etcétera.

La supuesta evolución de la industria de la información se presenta aquí como una llegada a su límite que, una vez terminada la expansión de su crecimiento, empieza su hecatombe al no tener otro medio de disipar su presión (Bataille, 1987). Y aunque la dilapidación onerosa se presente de manera evidente como la inversión absurda de nuevas tecnologías, su mayor gasto improductivo se da en el mismo nexo que une a los programadores: la donación de su trabajo.

Ante el viejo argumento en contra del *software* libre y del código abierto sobre la irracionalidad de regalar el código, se ha de contemplar lo siguiente: pocos son los programadores involucrados en esta clase de proyectos que efectivamente pueden vivir de ello. La mayoría de ellos *donan* su tiempo libre disponible; es decir, a la par del trabajo que trae el pan a la mesa, deciden darle continuidad a su quehacer a través de proyectos de *software* libre o de código abierto. Entre los que sí les es posible vivir bajo esquemas libres o abiertos, su sustento económico depende de las *donaciones* realizadas a sus proyectos. En ambos casos el *don* es el alimento que permite el crecimiento de proyectos de *software* libre o de código abierto.

La donación de los frutos del trabajo no tiene un carácter ornamental, sino que, por su carácter ritual y sus expectativas sociales implicadas, es una forma de institución dentro del grupo de los programadores. Este fenómeno podría denominarse como «prestaciones totales de tipo agonístico» (Mauss, 2009).

La cultura en las comunidades de *software* libre o de código abierto es una del «reciclaje» —la asociación no es mía sino de una compañera de trabajo—: un constante recibir, transformar y dar (Mauss, 2009). El programador recibe con un supuesto carácter incondicional el código de otro programador, el cual utilizará e incluso mejorará para la consecución de su trabajo. A continuación, el programador que se identifica con este modelo de desarrollo dará a cambio el código fruto de su trabajo u otra forma de retribución no económica, como es la labor de enseñanza o de difusión de otros proyectos de código abierto. O bien, en la medida de lo posible buscará llevar a cabo donaciones

económicas, aunque esta forma de agradecimiento es la menos común e incluso despreciable dentro de la misma comunidad —más bien es un puente exógeno para que el no-programador apoye el trabajo del programador.

La incondicionalidad es supuesta ya que si bien el trueque no le interesa tener como intermediario a la moneda, sí se da a modo de crédito (Mauss, 2009). El programador que usa el código de otro contrae una deuda, en forma de compromiso social, que solo se satisface mediante una retribución igual o mayor: donación de su trabajo ya sea como código, enseñanza o difusión. En estas transacciones el honor (Mauss, 2009) en la *egoboo* (Raymond, 2016) tiene una función más relevante que el valor monetario del trabajo. El programador del *software* libre o de código abierto no hace una evaluación cuantitativa de su trabajo donado en relación con otros, sino más bien establece una relación cualitativa cuyo objetivo es la obtención de un mayor estatus dentro de la comunidad.

El rango dentro de estas comunidades no se rige por la cantidad de dinero que cada programador pueda dar al conjunto sino en el grado de compromiso y de horas dedicadas al proyecto. Si se toma en cuenta que la mayoría de los programadores colaboran en estas comunidades durante su tiempo libre, su aumento de rango dentro de la tribu depende de manera directa en la disposición de una dilapidación de ese tiempo siempre abierto a su uso en otras actividades improductivas. Por otro lado, el valor del trabajo de los programadores que viven de esta clase de proyectos queda ligado a cuántas horas está dispuesto a dar para la reproducción y crecimiento de la comunidad.

En ambos casos el fenómeno de la donación del trabajo no queda constreñido a una realidad económica o jurídica: nadie está obligado a dar dinero ni hay ley que indique cómo se ha de regular este intercambio de trabajo. En su lugar, se da un fenómeno normativo y moral —detectado desde hace tiempo por el movimiento del *software* libre (Stallman, 2001)— donde la dilapidación «total» es proporcional al mantenimiento o aumento del rango dentro de la tribu (Mauss, 2009).

La obligación de dar, de devolver y de recibir —en estas

comunidades se desaconseja reinventar la rueda, por lo de manera constante se exhorta a adoptar y quizá modificar el código dado por otros— es tan internalizada que no se concibe como un deber. Tal vez este es uno de los factores que causaron molestia cuando el movimiento de *software* libre explicitó el «deber kantiano» implícito en estas prácticas de donación. El deber no se indica mediante el lenguaje, sino a través de los distintos momentos en los que se da la donación, por los cuales el programador incluso con vigor está dispuesto a que su vida sea una completa dedicación a la producción y mantenimiento de código.

La dilapidación llega a tal grado que algunos elementos del organismo informacional prefieren gastar sus recursos económicos en pos del *software* libre o del código abierto, ir a prisión, renunciar a otras libertades civiles —como el libre tránsito— o incluso la inmolación. Los hermanos Dúrov son un ejemplo de despilfarro económico al apostar por el código abierto en proyectos como Telegram o al hacer una donación millonaria a Wikipedia (Pigareva, s. f.). El juicio contra los fundadores de The Pirate Bay es una muestra de una puesta al límite de la ética *hacker* libertaria en pos del acceso abierto a la información (Sullivan, 2017). El asilo político de Julian Assage, Edward Snowden, o la falta de paradero de Alexandra Elbakyan o los fundadores de Library Genesis son ejemplos de renuncia a otras libertades. Por último, el suicidio de Aaron Swartz por la acusación de terrorismo debido a la liberación de *papers* y la asociación de este hecho con su *Guerilla Open Access Manifesto* (Swartz, 2017) hacen ver la disposición de hacer de la muerte la mayor expresión del don y del gasto improductivo.

Pero el gasto oneroso no solo se da en la donación excesiva entre los miembros de una tribu, sino que su forma más espectacular acontece cuando el mismo organismo informacional empieza a despilfarrar su excedente en forma de apertura de proyectos de *software* completos. La política de Microsoft inauguró el modo de crecimiento mediante una negociación del uso de la propiedad intelectual del *software*. Sin embargo, de manera paralela las comunidades de *software* libre y de código abierto alimentaban al organismo informacional mediante la donación de su trabajo.

No se trata de una evolución en la alimentación que va de lo privado a lo público, ni de una conflagración total entre dos bandos. Sino de dos modelos interdependientes de desarrollo de *software*. Sin la presión ocasionada por el *software* privado las comunidades de *software* libre o de código abierto no se hubieran visto impulsadas a acelerar su desarrollo. Al final, la guerra exige un compromiso de destrucción mutua (Bataille, 1987) mediante la carrera armamentista sinfín (Barlow, 2016). Sin la donación aparentemente irracional de las comunidades de *software* libre o de código abierto, las empresas desarrolladoras de *software* privado no hubieran visto la necesidad del trabajo voluntario para el mantenimiento a largo plazo de su código. Es decir, en el límite del crecimiento de la «economía de la información» se pierde la productividad en la gestión restringida de la propiedad intelectual, por lo que la apertura se vuelve menester para la sobrevivencia.

Si en la actualidad se hace evidente la transformación del excedente en dilapidación y lujo, no es tanto por una evolución consciente y un compromiso social de la «economía de la información», sino a un elemento ya presente desde un principio en la gestación del organismo que solo en la cercanía del fin de su crecimiento da un papel primordial a las formas que pretenden evitar su muerte: donación sin medida del trabajo y la constante producción de proyectos que mueren en un corto o mediano plazo. En el organismo constituido por los programadores lo esencial es el trabajo constante, sin importar en qué medida este sea traducible en un intercambio cuantitativo por medio del dinero. Por esta hambre de código producido por el trabajo, es entendible el por qué la historia de la informática yace sobre una gran cantidad de proyectos abortados o abandonados antes de su maduración: lo importante nunca ha sido la completud de los proyectos, la creación de *software* o de nuevas tecnologías, sino cómo esto sirve de base para la prolongación de una profesión, así como de la especialización y el mejoramiento de las habilidades del programador.

En la práctica ordinaria del programador el código nunca abandona su carácter «sagrado» (Foucault, 1999). No se niega el valor

del código, sino quizá la valía de los lenguajes de programación y métodos por los cuales se hace el *software*. La lucha de los distintos modelos de desarrollo no pone en juego al trabajo del programador, sino que se trata como una violencia autoinfligida en el organismo por un conflicto ético (Foucault, 1999) que pretende su reproducción y perfeccionamiento. El dedo sobre la llaga no es sobre los límites de crecimiento del organismo informacional —incluso muchos programadores ven en el horizonte una mina de posibilidades aún no explotada, en lugar de una barrera—, sino sobre los límites racionales de cada uno de los modelos que le sirven en la producción de su alimento.

¿Cómo es posible la transgresión como la afirmación del límite y la vivencia en la contradicción (Foucault, 1999), cuando el carácter negativo es solo un elemento dialéctico cuya síntesis es el pretendido crecimiento ilimitado? Quizá la transgresión empiece cuando el programador acepte el carácter contradictorio y limitado de su trabajo: un quehacer que tanto crea los monstruos engendrados por el código y la propiedad intelectual, como las mismas armas que buscan su aniquilación, la donación que frena su apetito.

En esta aceptación del límite, el programador verá que su trabajo no es la producción de una distopía al estilo Matrix, ni la creación de una utopía al modo de un crecimiento en la calidad de vida hasta que nadie se vea obligado a realizar trabajo indeseado, sino un movimiento que forma parte del cumplimiento inútil e infinito de un universo cuya energía ni se crea ni se destruye, solo se transforma a través de la dilapidación de sus excedentes (Bataille, 1987).

¿Qué queda de la labor del programador? Darlo todo por una voluntad que no obedece a un conflicto ético ni a una perfectibilidad dialéctica, sino al mero goce de crear código pese a que esté destinado al fracaso o al aborto. En fin, ¿es esto una falta de compromiso social o mera explicitación del desinterés y la desilusión por lo «libre» que ya se da entre los programadores? ¿Qué acaso el compromiso social del desarrollo de *software* libre o de código abierto no representa también una guerra constante de una actividad contradictoria y una dilapidación del tiempo

excedente sin medida? Quizá la fatiga no sería tan frecuente si el programador no se fijara en un movimiento situado en la escasez de aperturas; tal vez la angustia podría evitarse si el mundo en el que se vive no fuese operado por la razón instrumental que «dispone de los momentos futuros» (Díaz de la Serna, 1997) en el que el crecimiento del organismo siempre es posible...

Autor: Ramiro Santa Ana Anguiano.
Redactado: noviembre del 2017.
Última modificación: noviembre del 2017.

Bibliografía

Barlow, John P. (8 de mayo de 2016). "Vender vino sin botellas. Una recopilación de ensayos críticos". En: *¿Propiedad intelectual?* 1.ªed. México: Perro Triste. Págs. 33-96. ISBN: 9786079718404. URL: https://archive.org/details/PropiedadIntelectual (visitado 13-11-2017).

Bataille, Georges (1987). *La parte maldita. Precedida de «La noción del gasto»*. 2.ªed. Barcelona: Icaria. Pág. 249. ISBN: 8474261309. URL: https://github.com/NikaZhenya/maestria-asignaturas/raw/master/semestre_1/63047_t118_curso_georges-bataille-saber-y-transgresion/bibliografia/bataille_georges-la_parte_maldita.pdf (visitado 13-11-2017).

Dernbach, Christoph (2012). *Microsoft's Relationship with Apple*. URL: http://www.mac-history.net/contact-impressum (visitado 13-11-2017).

Díaz de la Serna, Ignacio (1997). *Del desorden de Dios. Ensayos sobre Georges Bataille*. 1.ªed. México: Taurus. Pág. 162. ISBN: 9681903706. URL: http://libgen.io/book/index.php?md5=7DD3EE8EDCF597F1034469F4A88186D3 (visitado 13-11-2017).

Dormehl, Luke (12 de ago. de 2017). *Today in Apple history: Apple's war with IBM commences*. URL: https://www.cultofmac.com/441919/apple-history-ibm-personal-computer/ (visitado 13-11-2017).

Foucault, Michel (1999). "Prefacio a la transgresión". En: *Obras esenciales*. Vol.1. Barcelona: Paidós. Págs. 145-160. URL: http://libgen.io/book/index.php?md5=

0CB31775184D7475AFAB44C96F7F4EC3 (visitado 13-11-2017).

Lakhani, Karim R. y Robert G. Wolf (2005). "Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects". En: *Perspectives on Free and Open Source Software*. Cambridge: MIT Press. Págs. 3-22. ISBN: 9780262062466. URL: https://mitpress.mit.edu/books/perspectives-free-and-open-source-software (visitado 13-11-2017).

Lessig, Lawrence (2009). *El código 2.0*. Madrid: Traficantes de Sueños. Págs. 31-41. ISBN: 9788496453388. URL: https://www.traficantes.net/libros/el-codigo-20 (visitado 13-11-2017).

Lévy, Pierre (8 de mayo de 2016). "El anillo de oro. Una recopilación de ensayos críticos". En: *¿Propiedad intelectual?* 1.ªed. México: Perro Triste. Págs. 168-193. ISBN: 9786079718404. URL: https://archive.org/details/PropiedadIntelectual (visitado 13-11-2017).

Mauss, Marcel (2009). *Ensayo sobre el don. Forma y función del intercambio en las sociedades arcaicas*. Buenos Aires: Katz Editores. Pág. 269. ISBN: 9789871566105. URL: http://libgen.io/book/index.php?md5=8182256190E2ED1A8578C2DF818F78B3 (visitado 13-11-2017).

Pigareva, Olga (s. f.). *Prominent Russians: Pavel Durov*. URL: http://russiapedia.rt.com/prominent-russians/science-and-technology/pavel-durov/ (visitado 13-11-2017).

Popescu, Adam (16 de nov. de 2017). *Is Mexico the next Silicon Valley? Tech boom takes root in Guadalajara*. URL: https://www.washingtonpost.com/business/is-mexico-the-next-silicon-valley-tech-boom-takes-root-in-guadalajara/2016/05/13/61249f36-072e-11e6-bdcb-0133da18418d_story.html (visitado 13-11-2017).

Raymond, Eric S. (8 de mayo de 2016). "La catedral y el bazar. Una recopilación de ensayos críticos". En: *¿Propiedad intelectual?* 1.ªed. México: Perro Triste. Págs. 97-167. ISBN: 9786079718404. URL: https://archive.org/details/PropiedadIntelectual (visitado 13-11-2017).

Santa Ana Anguiano, Ramiro (2016). *Historia de la edición digital*. Ed. por Mariana Eguaras. 1.ªed. México: Nieve de Chamoy. Págs. 5-6. URL: https://github.com/NikaZhenya/historia-de-la-edicion-digital (visitado 13-11-2017).

— (1 de ago. de 2017). *Adobe, el omnipresente*. Ed. por Mariana Eguaras. URL: http://marianaeguaras.com/adobe-el-omnipresente/ (visitado 13-11-2017).

Stallman, Richard (1999). *El derecho a leer*. URL: https://www.gnu.org/philosophy/right-to-read.html (visitado 13-11-2017).

— (2001). *¿Qué es el software libre?* URL: https://www.gnu.org/philosophy/free-sw.html (visitado 13-11-2017).

— (2007). *Por qué el «código abierto» pierde de vista lo esencial del software libre*. URL: https://www.gnu.org/philosophy/open-source-misses-the-point.html (visitado 13-11-2017).

— (2011). *Medidas que los gobiernos pueden adoptar para promover el software libre*. URL: https://www.gnu.org/philosophy/government-free-software.html (visitado 13-11-2017).

— (2014). *El software libre es ahora aún más importante*. URL: https://www.gnu.org/philosophy/free-software-even-more-important.html (visitado 13-11-2017).

— (8 de mayo de 2016). "El manifiesto de GNU. Una recopilación de ensayos críticos". En: *¿Propiedad intelectual?* 1.ªed. México: Perro Triste. Págs. 5-32. ISBN: 9786079718404. URL: https://archive.org/details/PropiedadIntelectual (visitado 13-11-2017).

Sullivan, Tom (17 de abr. de 2017). *"Pirate Bay" founders convicted by Swedish court*. URL: https://www.csmonitor.com/Technology/2009/0417/pirate-bay-founders-convicted-by-swedish-court (visitado 13-11-2017).

Swartz, Aaron (1 de jul. de 2017). *Guerilla Open Access Manifesto*. URL: https : / / archive . org / stream / GuerillaOpenAccessManifesto / Goamjuly2008 _ djvu . txt (visitado 13-11-2017).