Software Libre / Código Abierto

Clase 23 ABR 2018

Hugo J. Curti

(Transcripción/Edición: Elías Todorovich)

Es importante entender el espíritu de una licencia. Una licencia de Software Libre o de código abierto básicamente dice, usalo como quieras, podés modificarlo, podés compartir las modificaciones. Lo único que te pedimos es que mantengas el crédito, las autorías. Algunas licencias te piden que si hacés derivados, que por lo menos lo pongas con una licencia que te dé las mismas atribuciones. Es lo que se llama el copyleft. La cláusula de copyleft protegía al Software Libre, sobre todo al principio, para evitar que alguien lo modificara y lo cerrara.

Otro peligro que enfrenta el Software Libre son las patentes. No es lo mismo propiedad intelectual que patente. La patente es algo mucho más fuerte y viene de otro principio legal distinto que el de la propiedad intelectual. Puedo ir a un registro de propiedad intelectual y registrar algo que hice. Internacionalmente no es necesario, la ley internacional no exige eso, está aceptado que con poner la línea de copyright, con año y nombre, eso es una declaración de propiedad intelectual. El tema es si hay un conflicto: ahí se necesitan herramientas para probar quién lo hizo antes.

Es muy difícil compatibilizar el espíritu de una compañía grande que busca lucro por sobre todas las cosas, con el espíritu de un sistema que busca garantizar la libertad de los usuarios. La filosofía del Software Libre es compatible con el mundo de las empresas pero no escala a niveles tan altos ¿Qué pasa cuando una cosa se hace muy grande? Tarde o temprano se convierte en un monstruo y con las empresas pasa lo mismo. Una empresa grande tiene una especie de conciencia propia y a veces no hay nadie que pueda controlarla y eso es lo que se vuelve incompatible con el Software Libre. La voracidad de la empresa se vuelve incompatible. Mi hija instaló Spotify en su teléfono. Spotify es un software privativo que utiliza DRM para mantener control sobre la música. El modelo de negocios de ellos es así y si no te gusta, no lo usas ¿Por qué pretenden armar un modelo de negocios basado en vender copias? Las discográficas dicen que hay que pagarle a los músicos de alguna manera y es el argumento que usan para vender las copias, sostener las leyes de propiedad intelectual y todos estos modelos de negocio cuando los mismos músicos a veces dicen que eso no les interesa, porque de cada mil pesos que se recaudan, al músico le va uno, el resto va a parar a una industria que se autojustifica todo el tiempo. Es un problema en el plano social y en el plano económico, no técnico. Desde el Software Libre se sostiene que es mucho más sano desde el punto de vista social porque fomenta la colaboración en vez de la competencia, y desde el punto de vista económico porque distribuye. El Software Libre permite emprendimientos de pequeña envergadura que funcionan muy bien y que se basan en servicios. No hay forma más genuina de ganar el dinero que trabajando. El Software Libre propone que uno preste un servicio

usando herramientas libres, y cobre por ello. Cualquiera que se haya tomado el trabajo de conocer la herramienta y estudiarla, la puede usar. Eso genera movimiento en la economía, se fomenta que todos tengamos circulante. Tanto desde lo social, como desde lo económico, el Software Libre es una alternativa superior, pero tiene que enfrentarse al status quo y en el mundo como estaba planteado en aquella época cuando surgió, era peor que ahora. Actualmente se considera que está cumplida la misión del proyecto GNU.

DRM

DRM (Digital Rights Managment)[*1] es una tecnología apodada "Digital Restriction Management" por gente detractora de esta idea. Consiste en la aplicación de técnicas criptográficas para controlar quién puede y quién no puede hacer determinada cosa con un recurso. Es una generalización de lo que hizo TiVo[*2]. TiVo tenía en el intérprete algo tal que si el hash no coincidía... Primero, el software era abierto pero el intérprete no. El intérprete o el hardware mismo no es abierto. Sigue siendo según la opinión de muchos que estamos del lado del Software Libre, un intento por mantener un viejo paradigma, tratando de hacer que cosas que naturalmente se difunden, que se pueden copiar sin error, lo cual es un beneficio muy grande que la tecnología nos da, lo convierten, la retroceden artificialmente en algo que es escaso, que tiene un costo de copia. Por ejemplo si grabo analógicamente un tema musical digital en origen, y me lo quedo en otro teléfono, pierdo calidad. Qué necesidad hay de hacer eso? Uno puede no usar Spotify, Netflix, en Netflix pasa algo parecido. Aunque Netflix te da series, y a las series normalmente las miras una sola vez. Pero si te gusta coleccionar películas, Netflix no te sirve. Con la música es más común coleccionar la que te gusta, la guardas y la escucha muchas veces. Los modelos como Spotify no te lo permiten porque utilizan técnicas de DRM. No se los puede criticar por eso porque es su modelo de negocio. Ellos no han encontrado otra manera de subsistir que no sea a través de eso. Lo interesante sería encontrar una forma alternativa, que no viole las leyes, ni infrinja el copyright, porque no hace falta infringir los copyright. Es verdad que en una película de bajo presupuesto es más parecido a como se trabaja en Software Libre. Uno construye algo y no gasta mucho dinero en la construcción porque usas cosas que ya están hechas, combinas y haces un aporte tuyo. Entonces se necesita poca gente, poca inversión, relativamente poco esfuerzo para obtener un producto probablemente bueno. No requiere que tenga que recuperar una inversión y entonces no tengo un problema en dejarlo libre. No puedo pedirle a Netflix que haga eso, cuando gastó millones de dólares para producir películas, para comprar derechos de otros, y armar todo ese monstruo. Ahora no es lógico pedirles que regalen las películas. No es lógico pedirles que no usen DRM. Lo que sí puedo, es cuestionar todo el modelo de negocio. Es decir, hasta dónde eso es bueno para mí. Yo miro las series de Netflix una sola vez y lo uso legalmente aunque no me gusta que usen DRM. Spotify no, por ejemplo. Me cansé de escuchar renegar a mi hija "Vamos a escuchar tal música y cuando la va a escuchar no bajó, se perdió, etc.". Con el software pasa algo parecido a la música. El software no es algo que usas una sola vez.

Pregunta: Desde el lado del Software Libre se plantea que todo el software sea libre o se ven como dos modelos conviviendo?

En las posiciones más radicales dicen que Netflix no debería existir. Una posición menos radical dice que puede existir y vos tenés el derecho de no usarlo porque no se puede prohibir que exista. Sí podés quejarte del DRM. Podía usar Netflix en el TV pero no en la computadora con Debian. Ahora sí porque hay una pelea que se perdió, que era no incorporar DRM a los estándares. La Fundación de Software Libre no quería que se lo incorpore a los estándares. Pero hoy hay dentro de los estándares de la Web posibilidad de incluir contenido DRM. El efecto colateral de esto es que tarde o temprano los browsers van a traer manejo de DRM y se va a poder ver Netflix en la aplicación. Yo no tengo una postura radical. No digo "Windows no tiene que existir". Lo que observo es los peligros que tiene usarlo. Si para vos el beneficio es más que el costo, el riesgo que asumís es una decisión tuya.

Pregunta: En un entorno de Software Libre se podría llegar a una mega producción como la de una película de Hollywood? Por ejemplo los juegos grandes en estudios gigantes.

Habría que ser muy ingenioso. No conozco casos aún. Creo que se debe poder, hay que estudiar la forma porque el Software Libre no escala a niveles tan altos. Esas mega producciones a veces son una cosa maravillosa y hay otras que son claramente recaudatorias y comerciales. Vos podés cuestionar el principio de eso. Si no se hacen con la intención de producir algo valioso, la ideología del Software Libre no es compatible. Con la idea de armar algo grande y lindo probablemente sí. Hay que buscar la forma. Se hicieron cosas muy grandes. En juegos la cantidad de gente que trabaja para hacerlos es impresionante. El Software Libre no escala a esos niveles. Es un gran desafío para el futuro hacerlo compatible. Yo lo que creo personalmente, más allá de que no tengo una forma de decir cómo hacerlo dentro del paradigma de Software Libre, es que podría llegar a haber una forma siempre que el objetivo no sea estrictamente recaudatorio. La idea tiene que ser hacer el juego para hacer un aporte. Eso no significa que no se vaya a ganar mucho dinero igualmente. Se han llegado a hacer cosas interesantes. Si bien el juego no es Software Libre, las herramientas que se usaron para construirlo si lo son. Hay muchas cosas que dominan el mundo y están por todos lados que son Software Libre. El Kernel de Linux hoy está en Android, los servidores Apache están por todas partes, Navegadores Mozilla, son todas cosas que han llegado muy lejos, se sostienen, funcionan y llevan la delantera en muchos aspectos. Hay otros casos más difíciles pero es importante distinguir el paradigma. Es decir no es que para hacer negocio voy a hacer un juego. Es al revés: voy a hacer un juego y luego voy a hacer dinero. En la forma que se pongan esas dos prioridades se determina la compatibilidad con el Software Libre. Las peleas terminan cayendo en ese terreno.

Pregunta: Debería ser sin fines de lucro?

No, nadie te pide que no ganes dinero. A nadie se le pide que trabaje gratis. Pasa a veces que el objetivo es otro. Por ejemplo si escribo software en mi tesis para demostrar una hipótesis

puedo ponerlo disponible. No es que sea bueno, simplemente no era mi objetivo cobrar por eso.

El Software Libre lo que plantea es que no hagas grandes inversiones. Por eso no escala a niveles tan altos. O no espontáneamente, habría que pensar bien cómo escalar a un nivel tan alto. El Manifiesto lo que dice es que el objetivo no sea el software en sí. Que el objetivo sea otro y el software sea el medio para ese fin. Se provee un servicio con el software y cobras por ese servicio todas las veces que lo brindes. Pero el software es gratis, está disponible. Vale el servicio, no el software en sí. La diferencia con el software que viene del modelo privativo es que buscan ser un producto terminado que compras y te adaptas a él. Eso viene del modelo material. Compro un auto y me tengo que adaptar al auto. Me gusta, me lo compro; no me gusta, no me lo compro. Si me gustó y me lo compré, ahora me tengo que adaptar a él. El auto no se va a convertir en lo que yo quiera. El software sí tiene esa potencialidad. El software tendría que ser libre, el servicio no. El software no es un producto, es un medio para brindar un servicio. A veces el servicio es programar un software específico para un uso específico. No un software que se pretende que es un producto terminado que le va a servir a cualquiera. Puedo programar una planilla de cálculo y en vez de pretender vender mil copias de mi planilla de cálculo, digo que es una planilla de cálculo genérica. Mi modelo de negocio no es vender una planilla de cálculo que puede hacer cualquiera. Mi negocio es adaptar esa planilla de cálculo a tu necesidad y yo cobro por eso. Trabajo por el servicio de adaptarla. Qué tiene de bueno eso? Mi planilla de cálculo puede ser aprovechada por otros para hacer lo mismo. Uno puede decir "eso es competencia". No, eso es expansión. Si un japonés presta ese servicio, no compite conmigo que no voy a ir a Japón a adaptar la planilla de cálculo. Cuando me quiero acordar, mi planilla de cálculo está en todo el mundo. Además, en ese repositorio hay mucho software más que puedo usar aunque no lo escribí yo.

Pregunta: Si vos hacés un juego hermoso, terminás cobrando por el mantenimiento?

Habría que ver que servicios brindás. Preferentemente servicios que no restrinjan el acceso al software.

Pregunta: Qué pasa si hacés el juego como Software Libre y viene alguien, agrega una modificación, lo cierra y empieza a cobrarlo?

Eso está prohibido por la licencia, ahí se puede hacer un juicio. La Fundación de Software Libre te protege de eso justamente. Por otro lado, si quisiera hacer algo así, buscaría compatibilidad con el otro mundo, buscaría sponsors, gente con posibilidad de financiarte por la vía de la Fundación, no por la vía de la ganancia. Hay muchas empresas que ofrecen dinero para fundaciones. Probablemente alguna empresa podría financiarte tu juego, la Fundación le paga a los programadores y sostiene el juego, a cambio de eso la empresa obtiene dos cosas. Primero, exención impositiva. Lo que donan se descuenta de sus ganancias. Segundo es todo lo que tiene que ver con publicidad. Vas a poner "este juego está financiado por fulano de tal". Lo que sostiene el Software Libre es que tenemos que tratar de evitar, dentro de lo posible,

restringir el acceso al código o usar DRM o hacer ese tipo de cosas para sostener nuestro modelo de negocio.

Pregunta: Hay diferencias en este movimiento entre occidente y oriente?

Es más fuerte en occidente. En oriente no tengo conocimiento de cuáles son los movimientos que hay pero han sido distintos porque ellos tienen otra lógica. Países como India y Japón ya se han occidentalizado mucho con lo cual vas a encontrar algo muy parecido a lo que hay acá. Pero Rusia, por ejemplo, tiene otra filosofía muy distinta y no tengo mucho conocimiento de cómo se maneja allá. El Software Libre nació como una oposición al modelo que propuso convertir al software en producto en occidente. En occidente existió el fenómeno de la economía del consumismo, en oriente no, con lo cual si hay alguna movida va a ser diferente de la que hay acá porque ya la cultura es distinta. El Software Libre se presentó acá como un movimiento que dice que la idea de convertir el software en producto es algo malo, contranatural y hay que buscar otras maneras de hacer los negocios por todos los efectos negativos que aquello tiene. Te obliga a adaptarte al software por ejemplo. Si compras una planilla de cálculo y si no te gusta como está armado el menú te la tenés que aguantar y si no hace lo que querés también. Si tenés en conocimiento, el software al hacer algo inmaterial, tendrías que poderlo adaptar. El software te ofrece esa posibilidad y el modelo no se lo permite. Esto lo tiene que pensar cada uno. Hasta qué punto vas a renunciar a esta libertad por usar una herramienta privativa con un supuesto beneficio? Lo mismo, cuánto te condiciona en la esfera social y en la esfera económica esa herramient?. Es pensar más allá del beneficio a corto plazo. Más allá de encontrar una librería privativa que resuelve el problema esta tarde y si tengo que ir por mi camino voy a perder una semana o un mes. Mañana ya voy a estar preguntándome si es buena idea quedarme con esto o encarar mi propia solución. No me gusta quedar atrapado en ese tipo de cosas. Estás dejando cosas que son muy importantes para vos en manos de otros. No desconfío porque sean "otros", si puedo desconfiar porque no se cuál es el objetivo del otro o si se que sus valores lo llevan en primer lugar a ganar dinero. Haría tratos con esa persona? No me convence. Si lo necesito urgente por ahí hago un trato y le compro pero no pienso quedarme con eso para siempre porque no es inteligente. Un día puede decirme, "Te gustó? antes valía 1000, ahora vale 10 mil" porque sabe que no tengo otra salida. Te tiene dominado.

Voy a contar una experiencia. Estaba ayudando a la gente de sistemas en el puerto Quequén. En el 2010 o 2011 me vino a ver el gerente de mantenimiento con un problema en el sistema que estaban utilizando para la carga de los datos del cereal que se lleva ahí de los camiones a los barcos. Funcionaba lento y ya habían tenido que contratar a tres personas para cargar datos. Era un buen sistema pero escrito en Clipper en los años 90' que corría en DOS y tenía problemas. Era un sistema distribuído con máquinas por todos lados y empezó a tener problemas de red, corrupción de BD y problemas que tenían que ver con una escala que no resistía. Llegó el momento en que el cuello de botella de todo el proceso de descarga de cereales era este sistema. Si ustedes lo piensan, dónde puede estar el cuello de botella en un proceso como éste? En las balanzas, por ejemplo? No, era en la carga de datos y faltaban tres

meses para la cosecha de soja, con lo cual se iba a llenar el elevador de camiones, debían procesar mil camiones por día mientras que ahora procesaban 200 al límite del sistema. Era una situación de urgencia. En ese momento me pidieron que escriba un sistema nuevo, un sistema que luego tomó años en escribirse. Había mucha experiencia para aprovechar del sistema anterior. La captura de requerimientos estaba bien pero la tecnología era para tirarla porque no aprovechaba los procesadores ni nada. Entonces qué hacemos? Ellos querían comprar un sistema "enlatado" que se adapta bastante bien a las terminales portuarias a una empresa de Córdoba. Pero el sistema de ellos estaba acomodado al funcionamiento de ellos y al otro sistema había que adaptarse. Mi propuesta fue que lo iba a ayudar a resolver el problema en tres meses pero no vamos a escribir un sistema nuevo en tres meses porque eso no lo puede hacer nadie. En esa solución iban a poder procesar en tres meses la cosecha de soja pero después de eso les pedí el compromiso de armar el sistema nuevo. La segunda parte de la propuesta que les hice fue que el sistema lo armaran los mismos chicos que trabajaban, aunque no fueran profesionales. No mandarlo a hacer. Algunos ni programar sabían, pero tenían el potencial. Lo normal era que me mandara al diablo, pero aceptó. Se ve que por un lado estaba muy urgido y por otro lado era una propuesta buena aunque no es lo que primero haría cualquiera. Entonces agarrás un sistema que está por colapsar, y buscas el punto crítico. El punto crítico de ese sistema era la red porque una BD de 16 bits en DBase era muy vieja aunque funcionaba en máquinas muy nuevas en modo compatibilidad, usando la placa de red en modo compatibilidad. Todo lento, más fallas de concurrencia que había por todas partes en la programación. Pero el punto crítico era la red, si eliminamos la red resolvemos el problema. Le pedí al que había hecho el sistema en Clipper, unas 40 rutinas, que cambia las pantallas de terminal por una línea de comandos. Todo lo que es entrada va como argumento y todo lo que es salida, lo retornamos por pantalla con un JSON o algo parecido. Reescribir las 40 rutinas era un trabajo de más o menos una semana, porque no había que reescribir la lógica, sino la parte de la pantalla nada más. En un servidor metimos un DOSEMU[*3], que es un emulador de DOS, y las 40 rutinas para que corran ahí y armamos una interfaz Web. En ese momento nadie sabía programar. Agarramos un Drupal[*4] y le dije a los chicos -armen las interfaces con Drupal. Usando formularios de Drupal armaron todas las pantallas. En el servidor pusimos simplemente una rutina genérica que yo escribí y tomaba los datos de las pantallas e invocaba las rutinas de Clipper, y tomaba las respuestas. Eso lo logramos hacer en un par de meses y en el último mes nos dedicamos a entrenar a la gente a usar interfaz Web porque no sabían. En la terminal había cosas que eran distintas. La cuestión es que se pasó la cosecha de soja. En el sistema funcionando centralizado, los problemas de concurrencia no estaban. El sistema era el mismo pero llevamos a todo lo que corría en muchas máquinas a un solo servidor y resolvimos la urgencia. Ahora que ya no estaba más el agua al cuello, venía lo importante: había que escribir la BD, había que plantear la arquitectura, etc. y tomó tres años escribir el sistema nuevo.

¿Estás urgido? ¿Tenés una solución privativa que lo resuelve hoy? Todo bien, usala, pero no consideres que el problema está resuelto porque no está resuelto. Solamente resolviste la urgencia pero no resolviste el problema. No sabes cuál va a ser la progresión a futuro de esa librería privativa. Entonces, no te quedas tranquilo con eso. Voy a tener mi propia solución y cuando pueda suelto la librería privativa. El ideal es como una línea guía: es cierto que muchas

veces no se puede realizar porque hoy con la coyuntura que tenes no podes ir hacia donde el ideal te propone. Entonces, vas para otro lado porque no te queda otra pero no hay que olvidarse de a dónde quería ir. Si no termino caminando al revés de para donde quería. Andá para donde puedas pero no te olvides de a dónde ibas, así cuando tengas la oportunidad vas a retomar. Así, el ideal tiene sentido como guía. No es una propuesta radical. Es una propuesta de modelo para ordenar el sentido que le damos a las cosas. El Software Libre propone eso, que el software no debe tomarse como un producto y eso nació en occidente porque fue en occidente donde al software se lo tomó como producto. Ahora salieron muchísimas ideas para armar modelos de negocio buenos manteniendo ese ideal. Por lo que he leído y he visto, son buenos negocios pero no escalan a los niveles de Google o Facebook. Por otro lado me pregunto si será sano llegar hasta esos niveles. Hay lugares del mundo como EEUU donde es una vara de éxito llegar a esos niveles. Cuanto más grande es tu empresa, más exitoso sos. Pero ojo con eso. Eso no es necesariamente cierto. Es algo que está tan asumido que muchos ni se lo cuestionan pero no es necesariamente cierto. Hay otras varas para medir el éxito. Por ejemplo, que tan bueno es el servicio que prestás o qué calidad tenés de atención hacia las personas a las que les prestas el servicio. Además de prestarles el servicio, ¿los tratas bien? Se van contentos con vos o se van pensando "lástima que es bueno, si no me voy con otro". Eso también es una medida de éxito que puede valer mucho más que medir el tamaño de la organización. Entonces se puede hablar de muchas empresas de pequeña y mediana envergadura que perfectamente se acoplan al modelo y que producen muy buenos resultados y que en ningún caso implica que tengas que renunciar a ganar dinero. El punto es que si ganar dinero es lo más importante en tu vida por sobre todas las cosas, puede que choques contra el modelo de Software Libre. Ahora, si ganar dinero está entre tus objetivos pero no es el más importante, probablemente compatibilices con el modelo. Es una cuestión ideológica. Es falso que el Software Libre es gratis y uno tiene que trabajar gratis, y tiene que donar todo lo que hace. Tampoco se hace por razones altruistas. Es una forma de pensar que no implica que seamos santos ni que seamos buenos.

El Software Libre es un modelo técnicamente superior, eso prácticamente no lo discute nadie hoy, al menos la relación costo/calidad siempre es menor. A veces sale el contraejemplo de Apple, que produce un software muy bueno, bonito, integrado; pero qué costo tiene? No solamente costo para el usuario final, sino el costo que tuvo producirlo. Y cuando llegamos al terreno de la seguridad hacemos agua. En ese terreno el Software Libre siempre gana porque hay más ojos que lo ven. Hay muchas maneras de medir la seguridad. Si la medís por la cantidad de problemas que tuviste, no es una medida del todo exacta, porque no es lo mismo si a ese software lo usa mucha o poca gente. Pero el software privativo suele tener serios problemas de seguridad y por estar el código restringido, no está la potencialidad de revisarlo; y si pasa algo grave lo tapan. Con el Software Libre es al revés, ante la sola potencialidad de que haya un problema de seguridad se arma un escándalo. Una vez, un desarrollador de Debian cometió un error y en un binario de Secure Shell (ssh)[*5] mandó acotada la cantidad de bits aleatorios para generar las claves, con lo cual se generaban claves relativamente fáciles de predecir. Alguien lo vió y lo reportó. Se armó un escándalo y se corrigió en días. ¿Hubo algún ataque concreto que pudo aprovechar esa vulnerabilidad para ganar algo? Que yo sepa, no,

pero el escándalo se armó igual. Es al revés que con el software privativo que se les llega a escapar algo, es algo que ya se produjo, un ataque real con consecuencias reales. Hubo un caso muy serio, el WannaCry[*6] el año pasado, y otro caso grave fue un ataque muy feo que le hicieron a Sony, que les robaron cantidad de información.

Pregunta: Puede hacerse un WannaCry para Linux?

En Linux el impacto está mucho más controlado. Ponele que hubiera un WannaCry para Linux. Mandar un binario en Linux es mucho más difícil que en Windows porque está mucho más protegido. Qué vas a atacar si lo único que podés escribir es tu zona. Te agarraría tus archivos pero no te compromete todo el sistema, no pasa por la red a otro lado. Hasta el mismo autor del WannaCry tuvo miedo y le puso una llave de seguridad que se descubrió y por eso se detuvo. Había que registrar un dominio, si el virus lo veía, se detenía. Eso es una llave de seguridad.

El Software Libre está expuesto y eso produce una seguridad de mucho más calidad que la seguridad por oscuridad.

Los invito a reflexionar cómo se quieren posicionar personalmente. Qué les ofrece el software privativo y cómo está basado en conceptos que son antinaturales mientras que el Software Libre está basado en algo mucho más natural. Con lo cual, con un poco de ingenio y buenas ideas puedan armar proyectos de Software Libre. Se necesita ingenio para escalar a algo grande. Tiene que haber alguna manera, hay que buscarla.

Unix: Arquitectura de Filtros y Tuberías

Son tres conceptos muy distintos entre sí y GNU también. Los SO Unix nacieron en los laboratorios Bell, después pasó a System V y terminó en manos de Santa Cruz Operation Inc. (en 2001 se renombró como Tarantella, Inc., que a su vez se vendió a Sun Microsystems en 2005). Hablamos de las patentes. Unix es privativo pero fue creado en los Bell Labs por los mismos autores que crearon el lenguaje C, que son Kernigan, Ritchie y Thompson. El concepto, técnicamente, es excelente y nos vamos a detener a hablar de Unix como concepto. El lenguaje C tiene una filosofía similar. Proponía incorporar conceptos de programación estructurada a, básicamente, una programación en lenguaje de ensamblado. Así se simplificaba la programación en ensamblado y se llegaba a un lenguaje fácil de portar. Se buscaba que los programas pudieran correr en distintas arquitecturas (ISA, Instruction Set Architecture) y el lenguaje de ensamblado depende fuertemente de la arquitectura. Se buscaba un lenguaje eficiente. Partieron de la idea que programaban en ensamblado y querían programar en alguna cosa que pudiera ser más portable pero manteniendo la eficiencia del ensamblado y crearon el lenguaje C. El lenguaje C es muy sencillo. Es muy rico luego por las librerías. Y ellos mismos se pusieron a pensar, ¿un SO cómo tendría que ser? Eligieron el patrón de filtros y tuberías. Es una arquitectura de software que propone componentes sencillos

y especializados que se llaman filtros y un mecanismo para interconectarlos que son las tuberías. Así, la última fase del software, la conexión, la hace el mismo operador para lograr un objetivo. Por ejemplo en Unix tenes un comando que muestra la lista de procesos, otro que muestra la lista de usuarios, un comando para imprimir, uno pra enviar por la red, entonces si tengo que imprimir la lista de usuarios, tomo el comando de mostrar los usuarios, y lo conecto con el comando de mandar a la impresora. Unix está armado de esa forma. Hay comandos para procesar textos, para mostrar en pantalla, para administrar archivos. Para cada cosa hay una especialidad. Hay un comando para copiar archivos especializado con un montón de opciones para ajustar la copia de archivos. Hay otro comando que sirve para buscar textos, otro que te permite buscar archivos, si conectas el que busca archivos con el que busca textos, podés buscar textos en un conjunto de archivos. La arquitectura de filtros y tuberías es muy buena para administración. Tiene una ventaja muy importante, es muy potente. La desventaja es que requiere formación, que ensayes, que practiques, que aprendas. Requiere un esfuerzo activo de tu parte. La interfaz principal es una línea de órdenes. La línea de órdenes funciona con tu memoria de hábito, que es una parte de nuestra mente, que puede aprender cosas repetitivas. Con la memoria de hábito aprendemos a hablar, a manejar, a andar en bicicleta. La memoria de hábito se refuerza cada vez que repetís el hábito y se va perdiendo cuando dejás de hacer el hábito, a pesar de que hay partes que no se pierden nunca. La memoria de hábito es muy fuerte pero perdés práctica. Si bien hay mil comandos, vas a aprender 10, 15, 20 comandos y con estos comandos vas a hacer todo lo que quieras y el día que tenés que usar otros vas a consultar la documentación, lo vas a recordar, y lo vas a aprender a usar ahí. Así es como se interactúa con Unix. Además, en la época que se creó Unix no había recursos para crear una interfaz gráfica; eso apareció mucho después. La interfaz gráfica, por otra parte, es una interfaz que requiere muchos más recursos pero también posee muchos más recursos sobre todo en lo que se refiere a interactividad. El nivel de interactividad de una interfaz gráfica es mucho más alta que el de una interfaz de una línea de órdenes. Hay realimentaciones visuales y sonoras que en una línea de órdenes son mucho más limitadas. Las interfaces gráficas pueden ser mucho más intuitivas en el sentido que las ves sin haber estudiado nada y más o menos la entendés. Mucho feedback visual, si paso por arriba del ícono del disquete dice "Guardar archivo". La interfaz gráfica anda muy bien para cosas que no son repetitivas o que son muy específicas. Uno de mis primeros trabajos en relación con sistemas fue en el laboratorio de la Facultad en el año 1996-97. En aquel momento había dos redes, una basada en NT y otra basada en Unix, más precisamente en un SO de Silicon Graphics. Las máquinas tenían mayormente Windows pero había algunas con Unix. Un día mi jefe me pidió que le imprimiera una lista de usuarios, más de mil, de las dos redes. En NT fui al administrador de usuarios y era una pantalla con una barra de scroll. Y ahora cómo los saco de acá? Copiar y pegar? Botón para imprimir no había, botón para exportar no había. Finalmente decidí bajarme un programa que pudiera leer la lista de usuarios que está en el sistema y genere un archivo de texto. Tuve que usar una herramienta externa para hacer eso. Después me fui a Unix y lo hice en 30 segundos filtrando la lista que sale vinculada a los passwords y conectando a la impresora. Luego me senté a reflexionar sobre eso. ¿Cómo perdí tres horas en Windows NT? Eso definitivamente es un problema. De quién es la culpa de eso? Mi jefe: a quién se le ocurre pedir una lista de usuarios por impresora? Pero él tenía un requerimiento. A mi no me importa

para qué la quiere. Si no la necesitara no me la habría pedido. No puede ser culpa de él. Será culpa mía que no entiendo nada de esto? Pero tan malo no debo haber sido, en el otro sistema tardé 30 segundos. Habrá sido culpa del que diseñó la interfaz? Cómo no se le ocurrió poner un botón para imprimir los usuarios? Después pensé, ¿cómo se le tiene que ocurrir eso? ¿Es algo muy común? No, entonces tampoco le puedo echar la culpa a él. Entonces de dónde viene la responsabilidad del problema. No es el diseño de la interfaz pero sí de la forma de la interfaz. La interfaz gráfica no es flexible, es rígida en el sentido que yo puedo hacer con la interfaz únicamente lo que el diseñador pensó que yo quería hacer, lo cual le pone una responsabilidad muy grande al diseñador de la interfaz más allá de lo humanamente aceptable porque como se le puede ocurrir todo lo que a mi potencialmente se me podría ocurrir hacer. Ahí volví a valorar el tema de los filtros y las tuberías. Al que diseñó Unix no se le ocurrió jamás que yo podría querer imprimir la lista de usuarios pero planteó que en algún momento yo iba a querer ver los usuarios y que en algún momento yo iba a querer imprimir algo y me dejó que lo conecte yo. Flexibilidad. Una característica muy importante de las interfaces de administración. Me permite dar a mi dar el último paso, que soy en definitiva, el que necesita el servicio.

Unix es lo que Stallman conoció y es lo que trasladó al proyecto GNU. Cuando el dijo -vamos a armar un proyecto donde haya Software Libre para cualquier cosa que uno quiera hacer con una computadora. Entonces adquirió la arquitectura de Unix pero no le gustaba el hecho de inspirarse en algo que fuera privativo. GNU quiere decir GNU is not Unix. El nombre deja claro eso, además de ser un chiste y un acrónimo recursivo. Te dejo claro que esto no es Unix, pero me inspiré en Unix porque la arquitectura es excelente. De ahí viene lo que contamos después; el arrancó con el proyecto GNU y escribió un procesador de texto, un compilador, librerías, y un SO. Necesitaba un SO y ahí se mandó con una arquitectura micro-kernel que en la época era novedosa pero no había tecnología para implementarla y se le convirtió en el cuello de botella del proyecto. Todo iba bien menos el SO que iba lentísimo. Ahí fue donde se cruzó con Linus Torvalds y convenció a Torvalds, que ya tenía su Linux con una licencia académica, para que lo publique con una licencia GNU. A partir de ese momento conectaron las dos cosas y apareció el sistema GNU-Linux que combina las dos cosas.

Patentes vs. propiedad intelectual

La patente es un invento que viene del lado de EEUU[*7], y que no tiene una relación directa con la propiedad intelectual. El concepto de patente fue creado en la época de Edison. Edison fué precursor del sistema de distribución de la energía eléctrica, aunque Edison proponía la distribución usando corriente contínua y eso después fue reemplazado por la corriente alterna que proponía Tesla por una cuestión de seguridad más que nada. La CC no te perdona, te mata. Cuando te dicen -te quedás pegado, es tal cual porque vos llegás a tomar contacto con algo que tiene corriente contínua y los músculos se contraen y te quedás agarrado al cable y te morís. La CA es cambio te "patea". En la época de Edison se discutió (es la misma discusión que para el software) como hacer para que los inventores no escondan los inventos porque

tenían miedo que se los copien. Entonces dijeron -¿qué incentivo le damos a los inventores para que en vez de guardarse los inventos, los publiquen? Ahí aparece el concepto de patente. Cuando inventás algo te vas a la oficina con el modelo, lo registrás y eso te da un derecho de exclusividad comercial durante una cantidad de años, 70 (o 75) al principio. Solamente vos podías explotar eso comercialmente. Entonces se volvió lucrativo en EEUU inventar cosas. Si bien la intención fue buena en la época, se volvió en contra porque se perdió el sentido original y pasó a ser una herramienta hegemónica de mantener poder. Los herederos de los inventores todavía pretendían que les paguen regalías otros que ya habían perdido totalmente relación con la idea original. Se volvió algo tal que si “la pego” en algo me siento a tomar el sol y que me paguen. Después se pasó del invento a lo intelectual y empezaron a patentar obras intelectuales, no solamente registrar propiedad sino patentar. Patentar es algo mucho más fuerte. Quiere decir yo inventé la ventanita con el marco verde y el borde azul, la patentó en la oficina de patentes y ahora el que quiera usar esa ventanita me tiene que pagar. Una cosa es que se patente un invento y si yo quiero fabricar algo igual tenga que ponerme de acuerdo con el que lo inventó y otra cosa muy distinta es que yo esté patentando algoritmos, combinaciones de colores, técnicas de gimnasia. En EEUU se permiten las patentes de software, se pueden patentar algoritmos, programas. Entonces las empresas grandes tienen a un programador trabajando y a un abogado sentado atrás anotando lo que hace para salir corriendo a la oficina de patentes y tienen la oficina de patentes colapsada. En Europa se peleó durante muchos años para no permitir eso y se terminó prohibiendo. En Europa no se puede patentar el software. Las patentes son un enemigo terrible del Software Libre porque no solamente se patenta un programa, se patenta un algoritmo. Es decir, si yo utilizaba un algoritmo que está patentado tengo que ir a pagarlo. La patente es el derecho comercial. Propiedad intelectual es poner el copyright arriba. No tenés que registrarlo en ningún lado. Basta con que escribas copyright, el año y tu nombre y es tuyo. En caso de conflicto, si lo ponés público es relativamente fácil buscar el origen. Si viene otro y dice que lo hizo él se puede ver quién estaba antes. No podés patentar algo que tiene el copyright de otro. En argentina no está permitido patentar el software y en la mayoría de los países del mundo tampoco. Por la propiedad intelectual sí que alguien puede hacerte un reclamo. Hay patentes comerciales de productos pero no de software. El Software Libre peleó mucho contra las patentes. En Europa se ganó la batalla porque son peligrosas, se patenta cualquier cosa. Se basan en el principio, erróneo, de que algo por que vos lo ideaste es tuyo, o sos dueño de la idea. Uno no puede ser dueño de una idea. Sí podés decir - esto se me ocurrió a mí, porque es verdad, pero no podés pretender que otro por usarla te pague. Por lo menos eso es lo que se sostiene desde el Software Libre. Lo que sí permite la ley de propiedad intelectual, es que vos digas que se puede hacer. Las licencias de Software Libre dicen que yo explícitamente estoy permitiendo que se copie, que se distribuya, etc. Eso es lo que dice la licencia. Si ponés copyright solo y no agregás nada se asume un copyright cerrado en la mayoría de los países del mundo. Se asume que vos no permitís nada. Como es tuyo, si alguien lo toca es robo. Tenés que escribir algo abajo del copyright. Si no, si vos ponés algo y alguien lo baja, es robo para la ley. Los textos de las licencias se copian y se pegan. Las licencias pueden incluir cláusulas de copyleft. No solo hay licencias GNU, hay muchas licencias.

El código abierto no implica que tengas que publicarlo en un lugar determinado como Github. Puedo ponerlo con un copyright en el sitio web de la Facultad. Otro juego de licencias que está muy bueno es Creative Commons. Todas las licencias Creative Commons aparecieron mucho después (2001) y pueden usarse con software aunque fueron inspiradas en las licencias de Software Libre aunque se usan para otras cosas como libros, música, Datos Abiertos. Wikipedia por defecto publica Creative Commons. En Argentina rige Creative Commons.

La protección de la propiedad intelectual es una ley que habla de la propiedad de una creación intelectual, sea software, música, un libro, una obra de arte, una obra de teatro, etc. La de patentes es otra ley que está orientada originalmente a cosas físicas y te da derechos comerciales sobre esas cosas. Obviamente que si no tenés la propiedad intelectual de algo no lo podés patentar.

El copyright vacío dice esto es mío y no permite nada a los demás. Por eso debajo se dice que es lo que sí y que es lo que no se permite. Después está también el dominio público. Si publico algo y no pongo copyright ni nada automáticamente pasa al dominio público. Esto tampoco es lo que quiere el Software Libre porque tampoco es la idea que sea anónimo. Lo que propone es que cada uno pueda decir lo que hizo, lo que aportó, y el crédito quede, y la responsabilidad también. Propone usar las leyes de copyright para eso.

El algoritmo no tiene propiedad intelectual, tiene reconocimiento científico, que no es lo mismo. Si hago un algoritmo y lo publico y si era bueno, la historia lo va a recordar como algoritmo de Curti porque lo publiqué como Hugo Curti. La propiedad intelectual va sobre obras terminadas, concretas.

Santa Cruz Operations fue el último dueño de Unix ¿Qué estaba haciendo Santa Cruz Operations? Pretendía demandar a todo el mundo por por el nombre “stdio.h”. Microsoft fue un bebé de pecho al lado de Santa Cruz Operations en cuanto a lo agresivo contra el Software Libre. Hay empresas como Oracle que dicen ser amigas del Software Libre pero la comunidad del Software Libre le tiene pánico a Oracle. Si Oracle aparece en algún proyecto, automáticamente se hace un fork del proyecto. De todos los monstruos el que más compatibiliza con el Software Libre es Google. Android es parecido al Software Libre aunque no es libre. Es algo intermedio mucho mejor que Microsoft. Con Google hay que tener cuidado con el tema de los datos, lo mismo que con Facebook, pero eso es un tema de seguridad. Te regalan espacio, de todo, pero es una cama enorme para que te acuestes y te duermas.

Distribuciones: manejo de paquetes y modelo de maduración del software

¿Qué pasaba con el Software Libre? Estaban los fuentes disponibles y cualquiera que lo quisiera usar se tenía que bajar los fuentes, compilarlos, y tenía un sistema de Software Libre. Eso lo podía hacer un informático con una semana de tiempo disponible. Por eso aparecieron las distribuciones. La distribución es una organización que se dedica a traer los fuentes de

todos los desarrolladores de Software Libre, unir todo, compilarlo, generar un binario (hay algunas pocas distribuciones que distribuyen fuentes, por ejemplo, Gentoo). Si se distribuyen los binarios, la licencia requiere que se especifique dónde están los fuentes. Es una cuestión de comodidad. Una de las primeras distribuciones fue Slackware. Después aparecieron instaladores, manejo de dependencias, actualizaciones. Después Slackware adoptó el manejo de paquetes de Redhat. Redhat y SUSE son otras dos distribuciones importantes que nacieron con el mismo objetivo de acercar el Software Libre a los usuarios finales. Un objetivo peligroso porque en el camino se aproximan al paradigma privativo. Es peligroso porque te alienta a quitar la consciencia. SUSE es una distribución alemana y Redhat de Estados Unidos, son los que más progresaron en instaladores, manejadores de paquetes. Se inspiraron en los instaladores del software privativo de tipo siguiente-siguiente. Las dos tuvieron el mismo destino: se acercaron tanto al software privativo que en un momento dado empezaron a cobrar por las actualizaciones, y cuando se quisieron acordar fueron compradas por empresas que distribuyen software privativo. La comunidad de Software Libre se defiende de eso creando forks, abriendo los proyectos en nuevas líneas. De Redhat surgió Fedora, cuando Redhad empezó a evadir las licencias con una clara forma de pensar privativa. Y SUSE fue comprada por Novell (2003), una empresa clásicamente privativa. En ese momento apareció openSUSE.

Debian es la que más preserva las ideas originales de una distribución que fue creada especialmente para resguardar los principios del Software Libre/Código Abierto. Fue un contenedor para el proyecto GNU. Al incluir código abierto fue un poco pragmática. Si vos querés ser desarrollador de Debian tenes que pasar por un montón de filtros y de pruebas hasta que llegas a tener el suficiente crédito como para que te den espacio para que seas un desarrollador interno de Debian. Un mecanismo podría decirse iniciático. Debian importa de todos lados pero tiene que ser un desarrollador interno el que lo empaquete y lo ponga en la distribución. Debian desarrolló un mecanismo muy potente de manejo de paquetes que es el dpkg (Debian Package). Dpkg es una especie de targz pero además tiene metadatos, algo novedoso en su época. En los metadatos estaban las dependencias, conflictos, versiones. Entonces dpkg te decía que tenías que instalar antes para instalar determinado software. Inmediatamente después apareció APT (Advanced Package Tool) también de Debian, que permite armar repositorios y bajar paquetes .deb; es una herramienta acoplada a dpkg, que permite tener fuentes y bajar .deb de algún lugar (http, sink, torrent) que permite traer los paquetes e instalarlos. Dpkg decía necesitas estas 20 cosas y APT las buscaba, las bajaba y corría dpkg para instalarlas. Debian fue precursora de un sistema de instalación en línea: bajabas un bootstrap chico, booteabas y te instalabas todo de la red, sin bajar la imagen. Debian además de ser la distribución que dice -nosotros nunca vamos a ser privativos, no admiten software que no tiene licencia libre. Igualmente, lo que es Código Abierto pero no Software Libre, lo tienen separado. Una de las cosas más importantes de Debian es el modelo de maduración de software. Debian fue muy novedosa y hasta hoy tiene uno de los mejores modelos de maduración de software, es decir su política de actualización.

¿Cómo madura el software? El software tiene dos fuerzas, una madurativa y otra desestabilizante. La fuerza madurativa es la que se produce con el uso, el reporte de errores y

el debug. Cada vez que arreglan un software y el software se sigue usando, el software madura. La maduración es una curva asintótica, que con el tiempo tiende a estabilizar. La cantidad de errores en el tiempo que se ven en el software es una curva asintótica a cero. Con el tiempo cambian las condiciones y el software va quedando chico, incómodo, la tecnología lo va dejando atrás. Aparecen solicitudes de características nuevas que se van agregando, pero estos nuevos fragmentos de código tienen nuevos errores y ta tasa de errores reportados vuelve a subir. Cuando uno desarrolla software tiene que publicar el software maduro. Debian tiene un mecanismo para seguir esto, lo siguen tanto por paquete como por el conjunto de una distribución. Debian publica tres versiones en un punto del tiempo: estable, unstable, y testing. Software nuevo que aparece, ni bien pasa las pruebas alfa que hace el mismo desarrollador, va a unstable. Unstable tiene lo último, poco maduro, normalmente lo usan los desarrolladores. Unstable no se recomienda para entornos de producción porque hay errores. Debian lleva una estadística de errores y cuando la tasa de errores baja a un nivel razonable pasan el paquete a testing. Con APT se puede apuntar a cada una de estas versiones. La calidad que te dan la mayoría de las distribuciones es lo que en Debian está en testing. Debian hace ese seguimiento, además de por paquete, para toda la distribución completa y hace la suma de todos los errores de la distribución ponderado por la cantidad de uso de cada paquete. Cuando toda la distribución completa bajó la tasa de errores a un nivel determinado, la congelan (freeze). Eso quiere decir que dejan de pasar paquetes de unstable a testing. Esa versión congelada debería tener una tasa de errores cada vez más baja, si no funciona así la descongelan. Si todo va bien declaran una nueva versión stable; la anterior stable pasa a ser old stable. Esto normalmente se produce cada tres años. El problema de la versión stable es que a veces es software viejo. No es que stable no tenga actualizaciones, Debian tiene un equipo de seguridad que trabaja sobre stables, testing y old stable también (long term). El equipo de seguridad va siguiendo particularmente errores de seguridad y los back-portean a las versiones stable. Por ejemplo, Mozzilla descubre un error terrible en la versión 1.5 y lo arregló en la 1.8. A veces Mozzilla back-portea el arreglo a la versión 1.5 y a veces no. Si Mozzilla no lo hace, lo hace Debian. Entonces stable tiene actualizaciones de seguridad todas las semanas. No cambian las versiones de los programas. Si el software se queda demasiado en el tiempo y no se puede tolerar, hay versiones de software nuevas compiladas con las librerías viejas para que instales ese software a costa de romper la estabilidad pero para ese software solamente. Ahora hay un repositorio de volátiles que son cosas que naturalmente cambian mucho, como las bases de datos de los antivirus. Las versiones estables de Debian tienen número y nombre. Los nombres son personajes de Toy Story. Ahora la versión stable es la 9 Stretch (17 junio 2017), la versión 10 va a ser Buster. La versión unstable se llama siempre Sid.

Ubuntu es hija de Debian pero tiene muchas diferencias. Ubuntu está pensada para usuarios finales y es mucho más proclive a aceptar software privativo con políticas de aceptación mucho más relajadas que Debian. Ubuntu viene con muchas cosas instaladas y está pensada para máquinas con muchos recursos.

Mi consejo es siempre el mismo. Académicamente, y para evolución personal siempre conviene ir por el camino más largo, bajar Debian y adaptarla.

Nota: Reorientación y Destilación

¿Qué pasa si tengo un sistema y procesador y quiero generar el sistema para otra arquitectura?

Lo que se compila en la plataforma de origen es el kernel de Linux, las librerías de C, y el compilador, reorientados para una determinada arquitectura. Se compilan también los comandos básicos para alcanzar un sistema mínimo. Se lleva ese sistema mínimo a la máquina de destino y se compila el resto del sistema en la arquitectura de destino. Lo mismo pasa cuando cambio de versión de compilador. Compilo los fuentes de la versión nueva con la versión vieja y luego compilo los fuentes de la versión nueva con el compilador nuevo. Eso se llama destilación y se puede repetir varias veces. Y el compilador versión cero? Alguien lo escribió a mano en ensamblador o código máquina.

Referencias

[*1] Wikipedia: DRM también llamado programas anticopia, es un término que se refiere a las tecnologías de control de acceso usadas por editoriales y titulares de derechos de autor para limitar el uso de medios o dispositivos digitales a personas o equipo no autorizadas.

[*2] Wikipedia: TiVo es un grabador de video digital (DVR) desarrollado y comercializado por TiVo Corporation, introducido en 1999.

[*3] Wikipedia: DOSEMU es un paquete de software de compatibilidad que permite que los sistemas operativos DOS (e.g., MS-DOS, DR-DOS, FreeDOS) y sus aplicaciones corran sobre Linux en PCs basadas en procesadores x86.

[*4] Wikipedia: Drupal es un sistema de gestión de contenidos o CMS (por sus siglas en inglés, Content Management System) libre, modular, multipropósito y muy configurable que permite publicar artículos, imágenes, archivos y que también ofrece la posibilidad de otros servicios añadidos como foros, encuestas, votaciones, blogs, administración de usuarios y permisos.

[*5] Wikipedia: Secure Shell is a cryptographic network protocol for operating network services securely over an unsecured network. The best known example application is for remote login to computer systems by users.

[*6] Wikipedia: The WannaCry ransomware attack was a May 2017 worldwide cyberattack by the WannaCry ransomware cryptoworm, which targeted computers running the Microsoft Windows

operating system by encrypting data and demanding ransom payments in the Bitcoin cryptocurrency.

[*7] Un rastreo documental sobre la historia de las patentes muestra que se formalizó por primera vez en Italia mediante el Estatuto de Venecia de 1474 por el cual las nuevas invenciones, una vez puestas en práctica, tenían que ser comunicadas a la República para obtener protección jurídica contra los potenciales infractores. Este privilegio se concedía por un periodo de 10 años. En los EUA fueron establecidas por el Congreso en 1790, bajo la autoridad del Artículo 1 Sección 8 de la Constitución.