ArtefaCToS
Vol. 5, n.º 1, 83-122
Diciembre 2012
eISSN: 1989-3612

El conflicto en el Software: una aproximación a la incidencia del *fork* en los proyectos de software libre*

Conflicts in Software: An Approach to the Incidence of Forking in Free Software Projects

Javier Taravilla Herrera
Universidad de Salamanca/Universidad Autónoma de Madrid. Alcobendas
javiertaravilla@gmail.com*

Fecha de aceptación definitiva: 10-marzo-2014

«Code is law». Lawrence Lessig
«Los datos son objetos políticos». @acorsin

* La mayor parte de la bibliografía de este trabajo está en inglés, por lo que, salvo la anterior etimología en lengua original, el resto de las traducciones incluidas son mías.

Resumen

La presente investigación pretende ampliar el marco de trabajo de los estudios actuales en filosofía o sociología de la tecnología, realizando un acercamiento a una práctica surgida en las comunidades y movimiento de Software Libre de los últimos años conocida como *fork*. Una práctica que consiste inicialmente en hacer una copia del código fuente de un programa de software para desarrollarlo en una dirección distinta de la original, pero que analizada en detalle sugiere un amplio abanico de cuestiones relativas al comportamiento específico de estas comunidades.

Palabras clave: fork, bifurcación, software libre, axiología, filosofía de la tecnología.

Abstract

This research aims to extend the framework of the current studies in philosophy or sociology of technology, making an approach to a practice emerged in the communities and free software movement of last years, known as fork. A practice described initially as making a copy of the source code of software to be developed in other direction than the original code, but analyzed in detail suggests a wide range of issues concerning to the specific behavior of these communities.

Keywords: *fork, free software, axiology, philosophy of technology.*

1. Introducción

Para facilitar el acceso a las cuestiones tratadas o lectura del trabajo, facilito en primer lugar información sobre la estructura del trabajo.

En las secciones 1 y 2 analizo las definiciones del término *fork* y las comparo con otros ejercicios de copia o replicación de software, seguido de un recorrido por los trabajos publicados y las distintas percepciones sobre ello, de entre las que he seleccionado algunas preguntas que pueden ser respondidas empíricamente en el marco de esta investigación. A continuación, en las secciones 4 y 5 describo el trabajo de campo realizado y analizo el comportamiento de las comunidades que sostienen una serie de proyectos originales y sus correspondientes *forks*, a través de la observación de los resultados de continuidad de sus proyectos y la medición de las líneas de código de sus versiones de código fuente. Un análisis que tiene un alto contenido experimental, pues a pesar de seguir la línea de otros trabajos o

utilizar herramientas conocidas, pretende desplegar su actividad sobre una cantidad de proyectos no antes realizada.

Por último, en las secciones 6 y 7 manifiesto el interés en las investigaciones que desde el ámbito de la filosofía de la ciencia y la tecnología, nos permitan conocer el comportamiento y valores que guían la actividad creadora y productiva de los colectivos de software libre[1], debido a su notable relevancia para las concepciones políticas de la tecnología moderna, así como en la configuración del intercambio y creación de conocimiento en nuestro mundo actual.

2. Origen e historia del término

Para poder hablar de *forks* o *forking*, lo primero que se precisa es una aclaración del origen y definición del término. Un concepto que surge en la lengua inglesa de principios del siglo XIV como «to divide in branches, go separate ways»[2], y es incorporado a la informática del siglo XX a través del grupo de estándares de operaciones de portabilidad de los sistemas UNIX (conocidos como POSIX), como la orden o script [fork()] que permite que un sistema operativo realice una réplica de sí mismo y existan dos copias del mismo funcionando independientemente y/o realizando tareas distintas.

Es por ello que en analogía con esta operación, en informática se conoce como «Fork» (*software fork*) a la creación de un proyecto de software que, partiendo de un código ya existente, se desarrolle en una dirección distinta a la original. Estaremos por tanto ante un *fork* cuando topemos con una aplicación o programa desplegado paralelamente por dos o más grupos o

1. A pesar de ser usados de manera indistinta o sea más popular el término «open source», en el presente trabajo utilizaré el término «software libre» para referirme a este tipo de código. Ello se debe a que a pesar de que «software libre» comparta los aspectos técnicos del «open source» o «software de fuente abierta» (en castellano), refleja mejor los aspectos filosóficos o morales de este movimiento. Para ver más sobre esta distinción se pueden consultar los artículos de Richard Stallman: «¿Qué es software libre?», disponible en http://www.gnu.org/philosophy/free-sw.html, «Por qué el código abierto pierde el punto de vista del software libre», localizable en http://www.gnu.org/philosophy/open-source-misses-the-point.html o el artículo de Miquel Vidal: «Por qué evitar la expresión "Software de fuentes abiertas"», en http://gsyc.es/~mvidal/docs/no-sfa.pdf.

2. «Fork», Online Etymology Dictionary, disponible en: http://www.etymonline.com/index.php?term=fork.

comunidades de desarrollo, basando su trabajo en un mismo código fuente. Un fenómeno que aunque pueda encontrarse en proyectos de software propietario –probablemente de modo ocasional– está especialmente circunscrito al ámbito del software libre al estar legalmente permitido por sus licencias[3].

2.1. *Definición*

Son variados los intentos de definir el ejercicio del *fork*, *forking* o *code forking* con una mayor precisión. Uno de los primeros intentos lo encontramos en Bar y Fogel (2003, 213) en donde «Si la gente tiene desacuerdos importantes con las decisiones de los mantenedores [de un proyecto], pueden hacer una copia separada del código y empezar a distribuir su propia versión del programa, en donde sus decisiones sí serán implementadas. Esto es conocido como fork del código (a veces "ramificación" [*branching*] y no debe ser confundido con una rama literal en el CVS»)[4]. Una descripción repetida por Fogel (2005, Capítulo 4, apartado «Forkability»)[5] cuando dice que «la capacidad de cualquiera para hacer una copia del código fuente y usarlo para comenzar un proyecto competidor, es conocida como fork». Algo parecido a lo que sugiere el Diccionario Hacker de Eric Raymond conocido como *Jargon File*, en donde «un fork es lo que ocurre cuando dos o más versiones de un paquete de código fuente se desarrollan en paralelo con una base común de código, teniendo entre ellas diferencias irreconciliables»[6].

E idéntico tratamiento encontramos en las aproximaciones empíricas más recientes al fenómeno. Así, en el artículo comparativo de Ernst *et al.* (2010, 1), que pasa por ser el primer trabajo métrico en el campo, leemos

3. Sobre licencias recomendamos ver: http://www.gnu.org/licenses/license-list.es.html#DocumentationLicenses o el artículo de Miquel Vidal: «Informe de licencias libres», disponible en http://gsyc.es/~mvidal/docs/ikusnet.pdf.

4. Los CVS (*Control Version System*) son sistemas de control de versiones y consisten en ser aplicaciones en red que permiten alojar, mantener y registrar todos los cambios realizados en un proyecto, sirviendo de repositorio central y backup completo del código. Los más conocidos en el software libre son Subversion entre los centralizados, o GIT o Bazaar entre los descentralizados.

5. Cito de este modo por tratarse de una obra consultada en su versión digital en formato HTML.

6. Igual que la nota anterior, esta obra consiste en un diccionario presentado en formato HTML. En este caso, la cita pertenece a la entrada «Fork».

que «hacer fork de un proyecto es copiar la base de código y desarrollarlo en una dirección diferente a la del proyecto antiguo». Por su parte Nyman (2011, 3) sostiene esta misma concepción cuando afirma que «la forma más obvia del forking tiene lugar cuando un programa se divide en dos versiones, movido por el desacuerdo entre desarrolladores y con un código original, sirviendo como base para la nueva versión del programa». Una definición que repite en un texto posterior (Nyman y Mikkonen, 2011, 1) en donde «un fork se lleva a cabo cuando los desarrolladores hacen copia de un paquete de software y lo utilizan para el desarrollo independiente y separado del software original». Del mismo modo Kuusirati y Seppänen (2012, 2) establecen que «Hacer fork en proyectos de software libre significa copiar una base de código con o sin aviso de los desarrolladores originales, al no necesitar permiso de estos, y llevarlo hacia un nuevo grupo de objetivos», y Fung *et al.* (2012, 2) a pesar de reconocer diferencias de énfasis entre las definiciones existentes, lo sintetizan diciendo que «el forking fomenta mover la base de código de un proyecto de software libre existente en una dirección distinta a la que lo hacía su anterior líder».

Pero aquella extrapolación original desde POSIX, o la lista de definiciones anteriores, no ofrecen un criterio de demarcación suficiente para el fenómeno. Son varias las declaraciones que encontramos en este sentido en algunos de los autores anteriormente citados, especialmente en Nyman (2011) y Viseur (2012). Esto se debe a que su ejercicio tiene un evidente parecido de familia y/o similitud con otros procesos de replicación del código como son el *code reuse*, *code fragmentation*, *cloning*, *branching*, *distribution*, *modding* o incluso *hijacking*, de entre los que el *code forking* precisa ser diferenciado. Es por ello que antes de dar paso a esta clasificación, me parece importante señalar que sí que existe –a mi entender– una caracterización del *fork* suficientemente aclaratoria. Es la versión ofrecida por Robles y Barahona (2012, 3) y que, siguiendo un criterio señalado también tímidamente por Nyman, no adolece de los problemas de demarcación de las anteriores. Esta definición exige una serie de evidencias y consecuencias organizativas para diferenciar un *fork* de otras prácticas de replicación, al no quedarse en exclusiva en la rutina técnica y evolución alternativa de un código. Explora por tanto la *dimensión social del forking* al indicar la necesidad de que exista una ruptura manifiesta en el equipo, comunidad, plataforma, formaciones o colectivos de desarrollo del proyecto. Estos investigadores definen *fork* como:

> [lo] que ocurre cuando una parte de una comunidad de desarrollo (o tercera parte no relacionada con el proyecto) comienza una línea completamente independiente de desarrollo, basándose en la base de código del proyecto. Para que sea considerado un *fork*, el proyecto *debería tener*[7].
>
> 1. Un nuevo nombre de proyecto.
> 2. Una rama del software original.
> 3. Una infraestructura paralela (sitio web, CVS, listas de correo, etc.).
> 4. Una nueva comunidad de desarrollo (disjunta de la original).

Son estos cuatro elementos los que bajo mi punto de vista, permiten distinguir la práctica del fork de otros movimientos de copia de software. La exclusiva utilización de un código para incorporar funcionalidades nuevas o desarrollarlo en dirección distinta al plan original, no es suficiente para catalogar el movimiento de un código como *fork*. Se precisa de una serie de consecuencias y compromisos públicos que lo identifiquen como tal. Así, haciendo un paralelismo con otros campos del conocimiento que hacen uso de esta distinción, en el ámbito de las concepciones del *forking,* podríamos diferenciar entre una versión *fuerte* y otra *débil*. En el lado *fuerte* estarían los trabajos que exigen esta dimensión social y solo catalogan como fork aquellos proyectos que hayan surgido del enfrentamiento o falta de entendimiento manifiesta en un equipo de desarrollo, mientras que en el lado *débil* encontraríamos las versiones técnicas que identifican como fork cualquier desarrollo alternativo que tenga lugar dentro o no de las comunidades[8]. Una versión fuerte que, aunque no fuera desarrollada en detalle, se apoya en las opiniones del ingeniero David Wheeler (2007, A.6. «Forking»)[9] cuando afirma que:

> La creación o liberación simple de una nueva variante de código no crea normalmente un fork a menos que haya intención de crear un proyecto competidor. De hecho, las variantes de liberación experimental se consideran normales en el desarrollo típico del software libre […] Lo que diferencia un fork es *su intención*[10]. En un fork, la persona creadora del mismo tiene la intención de reemplazar o competir con el proyecto original del cual es fork.

7. La cursiva es mía.
8. Curiosamente se ha llegado al punto en el que en el conocido sistema de control de versiones GitHub, se refieren al ejercicio del *branch* como fork, algo de lo que hablaremos más adelante.
9. Texto solo localizable en digital y en versión HTML.
10. La cursiva es mía.

Por tanto, insisto en que el *forking* en el código libre no se explica suficientemente si atendemos en exclusiva al ejercicio de copia, replicación o escisión de un código fuente, y es necesario observar su dimensión táctica. Dicho de otro modo: atender al conflicto. Con esta definición me distancio también de los «forks autoproclamados» examinados por Nyman en sus trabajos[11], y que son considerados «fork» bajo exclusivo criterio de sus creadores, sin la presencia necesaria de las cuatro condiciones anteriores. Son iniciativas alojadas en un repositorio o *forjas de código* –plataformas existentes en la red para el desarrollo colaborativo– que recogen en su descripción «ser fork del proyecto X». Algo que a mi entender caería en la «versión débil» mencionada, pudiendo confundir con otros tipos de copia de software.

De este modo, una vez aclarada la definición de fork con la que trabajo, procedo a describir esos otros tipos de replicación, a fin de favorecer su ordenación y colaborar con los intentos de demarcación de algunos de los autores citados.

2.2. *La diversidad de la replicación*

Desde que a principios de los años 80 Richard Stallman definiera las cuatro libertades del software, estas pueden –y deben– ser entendidas como principios normativos e inspiradores de las actividades del movimiento del software libre, así como garantes de este fenómeno de *copia o replicación* del código. Unas libertades que además funcionan como criterio selectivo para determinar si una pieza, programa o producto de software es o no «código libre». Estas cuatro libertades son:

1. La libertad para usar o ejecutar el programa con cualquier propósito.
2. La libertad de estudiar cómo funciona el programa y modificarlo a voluntad, para lo cual es imprescindible el acceso al código fuente.

11. Me refiero a los textos firmados en solitario por Nyman o cofirmados con otros investigadores, incluidos en la bibliografía y que son resultado de sus trabajos de investigación de tesis doctoral.

3. La libertad para redistribuir copias del programa, ya sea de forma gratuita o no, ayudando con ello a otras personas a que lo puedan necesitar.
4. La libertad de distribuir versiones derivadas o mejoradas a terceros, permitiendo que otras personas o comunidades puedan beneficiarse de dichas mejoras[12].

Un conjunto de permisos que ofrecen una de las primeras tesis de este trabajo: el *forking* debe ser visto como algo inherente a las cuatro libertades del software. Es decir, en el software libre el *forking* debe ser visto como algo *natural* al ser una práctica perfectamente acogida y justificada en sus principios fundadores. Estas libertades operan como una axiología de la programación o *axiología hacker* que puede ser interpretada como «creadoras de modos de vida» (Winner, 1987) para el caso concreto del desarrollo de código y aplicaciones informáticas. Así, en la vertiente ingenieril han aparecido tecnologías y herramientas específicas centradas en el desarrollo de estos principios colaborativos o intercambio de código y conocimiento, como son los sistemas de control de versiones (CVS) o los *bugtrackers*[13], mientras que en el ámbito legal una miríada de licencias cubren las cuestiones jurídicas. Por otro lado, desde la perspectiva axiológica, los hackers/programadores han podido dotar de un sentido social, cultural y político a su labor. Y es que si como dice Stallman «el software libre respeta tu comunidad», los principios que lo definen proporcionan a cualquier persona o comunidad la posibilidad de crear una vía de desarrollo de código paralela e independiente a otras existentes, o que los programas puedan sean replicados en función de los intereses de un equipo o conjunto de personas. De esta manera, los fenómenos de replicación que podemos localizar en la creación de software libre son los siguientes:

12. Las cuatro libertades del software son ampliamente conocidas. Para consultarlas propongo ir a la fuente principal, que no es otra que la página del proyecto GNU, germen del movimiento del software libre y en el que desde su fundación colabora Richard Stallman: http://www.gnu.org/philosophy/free-sw.es.html. Igualmente, si queremos saber más sobre el contexto y momento en el que fueron definidas, recomiendo la lectura de «El proyecto GNU» en STALLMAN: *Software libre para una sociedad libre*, localizable en http://www.gnu.org/philosophy/fsfs/free_software.es.pdf o en edición papel.

13. Un *bugtracker* es un sistema en red de seguimiento de errores, fallos o mejoras del código, utilizado en muchos casos como principal vía para incorporar parches o mejoras en el código de un proyecto. Ambas herramientas son esenciales para el crecimiento y metodología de desarrollo del software libre.

a) *Code Reuse*

Una práctica habitual y –como acabamos de sugerir– *naturalizada* en los ambientes informáticos y en especial de software libre, que consiste en la utilización de un código para la construcción de un nuevo proyecto mediante la reutilización de sus componentes. Algo común en los albores de la informática moderna como solución a las necesidades de creación de grandes sistemas de software de manera controlada, fiable y rentable. Un comportamiento que ha sido ampliamente estudiado en la creación de software de los 90 y que no se adscribe solo al desarrollo específico de código, sino que es reconocido como estrategia esencial de evolución e innovación en toda la industria. Así lo señala Barnes (1991, 1) para quien «La característica definitoria de la buena reutilización no es la reutilización del software *per se*, sino la reutilización de la capacidad humana de resolución de problemas».

Son variados los aspectos del software que pueden ser usados de nuevo, tales como la descripción de problemas, las propuestas y análisis de viabilidad de proyectos, los modelos de negocio, las tablas de decisión, los prototipos, las bases de datos y redes de desarrolladores, los diccionarios, etc. (Isoda, 1995). Una práctica que ha sido analizada en casos de software libre por Haefliger *et al.* (2008, 1) donde lo definen como «software diseñado para ser reutilizado y proporcionar funcionalidades a otros proyectos de software». Hay otros trabajos como el de Capiluppi *et al.* (2011) que estudian la inclusión de la librería libavcodec en los componentes del software multimedia FFmpeg, presente en la actualidad en más de 140 proyectos de software libre.

Pero volviendo al objeto de esta sección, el *code reuse* se distingue del *forking* en ser una conducta que no pretende *en principio* (recordemos *la intención* de la que hablara Wheeler) la ruptura de ninguna comunidad. Es practicado por equipos que quieren integrar nuevas funcionalidades con rapidez, prefieren priorizar sus esfuerzos en otras partes del código, trabajan con recursos limitados, o quieren mitigar los costes del desarrollo. Un concepto al que podría subsumir la descripción de Robles y Barahona (2012) de *Derivation/distribution*, pues la definen como «aquellos programas o sistemas de software que surgen desde la base de un proyecto existente, con la intención de que sean compatibles con el proyecto original». Unos autores que ponen como ejemplo a los cientos de distribuciones basadas en el

sistema operativo GNU/Linux existentes en la actualidad[14]. Una práctica que se diferencia nuevamente del fork en no buscar ruptura o separación de un equipo de desarrollo, sino solamente el inicio o adaptación de un proyecto, basándose en la disponibilidad del código y/o el interés de un nuevo grupo de desarrollo y/o usuarios. Del mismo modo, el *port* o *porting* también puede incluirse como *code reuse*, pues consiste en adaptar un código para que se ejecute en un entorno distinto de aquel para el que fue diseñado. Un caso habitual de porting es la adaptación de un mismo programa para distintos sistemas operativos.

b) Code Fragmentation

Este ejercicio consiste en dividir una misma base o porción de código en distintas versiones o distribuciones. Una práctica identificable con el *branch* o *branching* (en español «ramas», «hacer ramas» o «ramificación») de la tipología de Robles y Barahona –y reconocible por cualquier desarrollador– como proceso de duplicación de una parte o totalidad de un código desde dentro de un sistema de control de versiones, con el objetivo de realizar y probar distintos cambios en él. Es decir, abrir o iniciar una rama o «branching» es algo que ocurre diariamente en la experimentación de mejoras e incorporación de nuevas características, y se diferencia del *forking* en que tampoco persigue fragmentar –al menos inicialmente– un proyecto o comunidad. Algo que ya era recogido por Fogel (2005) solo que en un sentido distinto al propuesto aquí. Recordemos que para este autor, hacer fork podía también ser conocido como ramificación (*branching*) no debiendo ser confundido con una rama literal en el CVS o sistema de control de versiones. Sin embargo, esta especificación difícilmente puede ser sostenida en un mundo en el que «hacer ramas» ha proliferado como ejercicio habitual y diario para transformar código *en una misma dirección*. Es por ello que Kuusirati y Seppänen (2012, 4) insisten en diferenciar claramente entre *branching* y *forking*, sosteniendo que:

14. Se puede saber más sobre las derivaciones o distribuciones del árbol de GNU/Linux en: http://es.wikipedia.org/wiki/Distribución_Linux.

> Hacer una rama en un proyecto es distinto del forking. Las ramas son realizadas dentro de los proyectos para permitir desarrollos paralelos. En lugar de que todos los creadores trabajen en el mismo código, se puede crear una nueva rama para un cierto desarrollo, tal como incorporar características especiales o arreglo de fallos. Así, una vez que estas características están completamente desarrolladas en la rama, pueden ser fusionadas (*merged*) en el proyecto principal. En el caso del fork, la fusión pocas veces ocurre.

Pero no se escapa que la distinción entre *code reuse* y *code fragmentation* puede presentar dificultades. Podríamos alegar que la fragmentación de un código siempre supone algún tipo de reutilización, o que la reutilización ocurre tras la separación de una parte. Una confusión manifestada cuando Nyman (2011) presenta como modelos de *code fragmentation* las distribuciones de GNU-Linux, que recordemos, siguiendo el catálogo de Robles y Barahona, habíamos identificado con el *code reuse*. Así, aceptando que son conceptos solapables con difícil distinción en determinados momentos, propongo diferenciar entre ellos a través de la distinción *interior-exterior*. De este modo *code reuse* sería el trabajo que se realiza *desde fuera* de un proyecto, principalmente por equipos de desarrollo que quieren aprovechar un trabajo para incorporarlo a otro. Por otro lado, *code fragmentation* o *branching* tendría lugar dentro de los sistemas de control de versiones y repositorios, realizado por los programadores que pretenden desplegar el código en la dirección original. Es evidente que esta ramificación podría conducir a un fork en el futuro, pero en principio no debería considerarse como tal.

Una propuesta que también aparece en Fung *et al*., a través de su separación entre *fork endógeno* y *fork exógeno*. Su distinción surge del modo de trabajar en los sistemas de control de versiones distribuidos[15] y mediante la fijación de una relación antecesor-sucesor entre proyectos, establece que el fork endógeno es creado para verter sus desarrollos en un mismo proyecto de software original (*antecesor*). Esta tarea es desempeñada por parte de los llamados «desarrolladores sociales» (*social developers*) que trabajan en el fork sucesor. En cambio, el fork exógeno lo funda otro colectivo y establecen interacción a través de los bordes de las comunidades (*community borders*). Pero según avanzan la idea, señalan que «un fork es a un árbol de forks, lo

15. Los sistemas de control de versiones distribuidos son aquellos en los que cada usuario tiene una versión del código, utilizando un repositorio central para sincronizar los intercambios. Entre los más conocidos están GIT o Mercurial.

que una rama es a un árbol de un sistema de control de versiones en donde las ramas son creadas para llevar a cabo tareas auxiliares» (Fung *et al.*, 2012, 4), para más adelante confirmar que «un fork puede usarse para iniciar una rama de desarrollo independiente en un nuevo proyecto de software libre». Pues bien, es en este momento de *independencia* en el que entiendo que sí estaríamos ante un fork, de modo que propongo considerar el fork endógeno exclusivamente como *branching* –o incluso *friendly* o *experimental fork*[16]–, al realizarse con la intención de colaborar con la serie original de desarrollo y no deberse a ningún desencuentro o conflicto en la comunidad. Para finalizar, afirmo que aunque no pueda establecerse una diferencia estricta entre *code reuse* y *code fragmentation*, sí podemos ofrecer una distinción de ambas en comparación con el *code forking*.

c) *Clon (software cloning)*

Un clon es un programa de software desarrollado con la intención de que sea imitación de otro. Puede conseguirse mediante ingeniería inversa, reescritura del programa entero (en otro lenguaje de programación) o a través de la imitación de su estructura, apariencia y funciones. Se diferencia del fork –y de las versiones previas de replicación– en el sentido de que no ejecuta o hace utilización fáctica de una base de código existente; de ahí que sea *software cloning* en lugar de *code cloning*. Sus ejemplos pueden ser los cientos de redes sociales, apps o juegos que existen en la actualidad y que sin reutilizar las líneas de código de otros proyectos, ofrecen el mismo servicio o imitan su aspecto y funciones.

d) Mod/Modding

El «mod» o *modding* es la estrategia por la cual se realiza una mejora, personalización o adaptación de un software existente, pero sin incorporarla a la versión central del programa (*main version*) ni suministrarse al resto de

16. Nociones también incluidas en ROBLES y BARAHONA (2012) para designar los forks que surgen con la intención de realizar mejoras en un proyecto determinado.

desarrollos, usuarios o distribuciones que se hagan de él. Es utilizada en exclusiva por sus creadores y podrían ser tipificadas como ramificación marginal (*marginal branching*). Una práctica que se encuentra muy extendida en la industria y consumo de videojuegos (Scacchi, 2010).

e) Hijacking

En último lugar, el *hijacking* o «secuestro del código» es el ejercicio mediante el cual un proveedor, mayormente comercial, intenta privatizar un código fuente. Algo descrito inicialmente por Lerner y Tirole (2002) pero que autores como Viseur (2012), al igual que Mateos y Steinmueller (2003, 12) identifican con el forking. Para estos últimos, el *hijacking* acontece cuando «las personas a favor de unas revisiones, deponen al líder del proyecto que se ha opuesto a ellas y lo dejan sin seguidores». Una interpretación que en caso de seguirse, identificaría como *hijacking* a muchos de los proyectos que más adelante veremos en este estudio, o incluso a todos aquellos forks cuyo resultado sea el abandono del proyecto original. Sin embargo, autores como Moody (2009) animan a distinguir el fork de los movimientos hostiles y comerciales de las empresas, al subrayar que compañías y comunidades son colectivos que se distinguen esencialmente en que unas pueden ser compradas y vendidas mientras que otras no. En esta coyuntura, el *fork* sería una vía para poder mantener un determinado código a salvo y no necesariamente un secuestro. Así, ante los intentos de empresas por hacerse con el control de algunos proyectos de software libre –caso de OpenOffice.org–, las comunidades tienen la posibilidad de hacer *fork* desde la última versión libre y continuar su desarrollo anterior. De este modo, identifico el *hijacking* como lo acuñaron en su origen los citados Lerner y Tirole, como movimiento manifiestamente hostil –en el sentido fuerte de secuestro– en que un código es cerrado o apropiado por una empresa o colectivo, impidiendo su utilización por parte de terceros. Un ejercicio identificable con las estrategias de privatización de un código fuente, en donde el código es hurtado a toda comunidad.

3. Estado de la cuestión: investigaciones, percepciones y preguntas seleccionadas

La naturaleza y metodología del movimiento del software libre ha despertado gran interés en disciplinas como la gestión de equipos y proyectos, economía, sociología o ingeniería. En concreto hay mucha literatura volcada en investigar el comportamiento de las propias comunidades (Mockus, 2002; Crowston, 2005; Robles y Barahona, 2009, por poner algunos ejemplos) o la evolución del software (Deshpande, 2008). Sin embargo, son muy pocos los trabajos que estudian las reacciones y relaciones de estos colectivos bajo las tensiones del *fork*. A su vez, si cerramos un poco más el objetivo, no hay ninguna investigación aún que se ocupe del papel de las empresas en las rupturas de comunidades de código libre, ni sobre la evolución de las escisiones en los colectivos de software con una gestión poco participativa.

De este modo, los artículos académicos que actualmente podemos encontrar con un trabajo de campo u observación empírica de estos hechos son menos de una docena. Está la serie de trabajos del grupo de Nyman (2011, 2012, 2013), el texto de Ernst *et al.* (2010), el análisis de Viseur (2012), el trabajo de Fung *et al.* (2012), el ejemplar de Gamalielsson (2012), Kuusirati y Seppänen (2012) y el artículo de Robles y Barahona (2012). Todos ellos han sido publicados en los últimos tres años por lo que hay que esperar un crecimiento inminente de las publicaciones sobre el fenómeno. Este trabajo pretende ocupar un espacio entre ellos y sugerir algunas propuestas de análisis.

Una ausencia de investigaciones que se debe a mi entender a dos factores principales: la presión cultural en contra y una cuestión evolutiva. Veámoslo:

3.1. *La oposición cultural*

El catálogo de valoraciones o corriente de opiniones que encontramos sobre estos movimientos es mayoritariamente negativa. Así, si visitamos de nuevo el *Jargon File* de Raymond y consultamos la entrada «Fork»[17] leeremos

17. Entrada «Fork» en el *Jargon File,* en: http://catb.org/jargon/html/F/fork.html.

que «Hacer fork es poco común. De hecho es raro que los ejemplos individuales [de forks] cobren gran importancia en la épica hacker»[18]. Y si a continuación revisamos el término «Forked» encontramos que «Forking es considerado una cosa mala –no solo porque implique mucho trabajo perdido para el futuro, sino porque suele ir acompañado de una gran cantidad de lucha y acritud entre grupos sucesores, con temas como son la legitimidad, la sucesión y diseño de la dirección. Existe una seria presión contra el forking»[19].

Y desde ahí el catálogo de juicios negativos es bastante amplio. Así Neville-Neil (2011, 2) insiste en que hay que reflexionar bien antes de promover una ruptura, pues «hacer fork de un proyecto […] a menudo te muestra como un chico petulante y mimado que quiere coger sus juguetes e irse a casa», es decir, algo propio de gente inmadura y caprichosa, mientras que Bezroukov (1999) argumenta que los egos pueden conducir al fork y que ello suponga la muerte del proyecto. Por su parte, Feller *et al*. (2000) insistían en el gran tabú que existe sobre el tema –coincidiendo con la «presión en contra» de Raymond–, y DiBona, Ockman y Stone (1999) relacionaban el fork con el miedo a perder el control por parte de las compañías, reconociéndolo además como uno de los principales riesgos a valorar al embarcarse en un proyecto de software libre. A pesar de ello, estos tres últimos investigadores muestran una postura ambigua al respecto al sugerir que el fork también supone resultados exitosos, como ha ocurrido con las réplicas del sistema operativo BSD por parte de la familia de NetBSD, OpenBSD o FreeBSD. Hay otros trabajos, que más directamente lo identifican como algo constitutivo en el software libre al afirmar que «la naturaleza del código abierto conduce a la fragmentación así como a la incertidumbre»[20].

Por su parte, para Fogel (2005) el fork debe ser siempre algo a evitar a lo que recurrir solo como último recurso (*last resort*), ya que supone una quiebra en los equipos de desarrollo, ahonda en la atomización de las comunidades y disipa la unidad ante el «enemigo» del software propietario. Una concepción *fogeliana* basada principalmente en razones prácticas (una especie de *pragmatismo anti-fork*) como son la pérdida de esfuerzos

18. Una *limitada importancia* que ha ido en aumento, si atendemos a MOODY (2011) y todas las reacciones surgidas tras las estrategias de algunas empresas en los últimos años.

19. *Idem*, «Forked» en: http://catb.org/jargon/html/F/forked.html.

20. Así podemos encontrarlo en la relación de percepciones sobre este fenómeno de la Introducción del artículo de VISEUR (2012).

y energía que se supone a una comunidad grande y unida. Un autor que también introdujo el término «forkability» –algo así como la tasa de inestabilidad en las comunidades anticipando, intencionadamente o no, los estudios empíricos sobre el tema– para designar la posibilidad de aparición de un fork y ofreciendo una serie de consejos para evitar que sucedan.

Pero de entre todos estos apocalípticos, quiero rescatar de nuevo la postura del ingeniero David Wheeler (2007). Este autor considera que el fork es una deriva poco apetecible para los desarrolladores, posición para la cual ofrece una razón cultural: la principal motivación de los programadores para desarrollar software libre es el aumento de la reputación mediante la llamada «cultura del don» (*gift culture*, que también podríamos traducir como «economía del regalo»). Este principio estipula que en el movimiento hacker el prestigio es obtenido de la mezcla de ciertas capacidades técnicas, junto a la generosidad para compartir los conocimientos –y por tanto el código– con el resto de la comunidad. Siendo así, el fork interfiere y atenta contra esta cultura de modo significativo, al ofrecer una imagen egoísta y caprichosa de sus impulsores. Por último sostiene que tras una ruptura solo existen cuatro escenarios posibles, de los que encontramos ejemplos para todos ellos en la historia del software libre. Estos son:

– El abandono del fork.
– La fusión de ambos.
– El abandono del original.
– El éxito del fork.

Llama la atención que no añada las opciones de «éxito/continuidad del original» como contraparte del primer escenario (abandono del fork), ni la desaparición o continuidad paralela tanto de un proyecto original como de su competidor[21]. Concluye este catálogo afirmando que «el porcentaje de abandono de forks es notablemente mayor al de los proyectos originales» (Wheeler, 2007, A.6. «Forking»). Una tesis de porcentaje de éxitos-abandono que será confrontada en los análisis posteriores.

Sin embargo en los últimos años también han surgido discrepancias interpretativas y los puntos de vista se han diversificado. Ha surgido una corriente de autores que valora el fork desde una perspectiva posibilista y no como algo necesariamente negativo. Destaca especialmente el caso del

21. Posibilidad apuntada también por ROBLES y BARAHONA (2012).

mencionado Linus Nyman quien señala que *de hecho* el forking puede ser la muerte del proyecto por la dispersión de esfuerzos, pero *su posibilidad* permite que un proyecto no muera siempre que haya programadores interesados en él. Sostiene que «[el fork] es al mismo tiempo salvación potencial y ruina» (Nyman, 2011, 1), que «code forking […] es una parte vital del software libre y algo que asegura su supervivencia» (Nyman, 2011, 3), o que «el fork asegura la posibilidad de supervivencia de un código, que puede ser replicado por una comunidad que no esté de acuerdo con la empresa que lo desarrolla» (Nyman, Mikkonen, 2011, 2). Llega a identificar este hecho con «la mano invisible de la sostenibilidad» y lo reconoce como la principal maniobra de resistencia del software libre frente a las estrategias de obsolescencia programada aplicadas al código. Un bando en el que también encontramos al también citado Moody (2009), el texto de Bitzer *et al*. (2005) donde se concibe como una vía de innovación en los entornos competitivos del software, o De la Cueva (2013, en el apartado «Las Prácticas del Software Libre»[22]), quien sostiene que «fork no supone necesariamente una ruptura con la comunidad sino, en muchas ocasiones, un enriquecimiento puesto que permite afrontar la solución a un problema mediante la creación de varias y diferentes herramientas».

3.2. *Evolución tecnológica*

En segundo lugar la *cuestión evolutiva* es un sencillo factor de novedad tecnológico. Y es que tras 30 años de vida, el movimiento del software libre solo puede considerarse como suficientemente extendido como base de una masa crítica, a partir del inicio del nuevo siglo. Un momento que coincide con la irrupción de las «.com» y el desarrollo y madurez de las tecnologías de red, información y comunicación. Por ello, aunque en la última década del siglo XX ya empezaran a ser conocidos y respetados algunos de sus contenidos, el movimiento del software libre seguía siendo un espacio residual reservado a *geeks* –entusiastas de la tecnología y la informática– y programadores especialistas. Es con la llegada del nuevo milenio –momento en el que curiosamente aparece una distribución con el eslogan «linux for human

22. Otro texto consultado en formato HTML.

beings»– cuando surgen las condiciones necesarias para que los principios de este movimiento se extiendan y no solo se instalen en ambientes técnicos –con nuevos perfiles de titulación y programación– sino que empiecen a afectar también a otro tipo de entornos culturales. A su vez, las tecnologías de copia e intercambio proliferan y el crecimiento y evolución de las comunidades de software libre, lleva a considerar uno de sus anatemas –el *forking*– como algo razonable y alternativa viable. Todo este crecimiento y evolución de las tecnologías ha supuesto que cada vez sea más barato hacer un fork, debilitando por tanto aquel *pragmatismo fogeliano* y facilitando con todo ello su ocurrencia.

3.3. *Sostenibilidad y software*

Pero de entre los temas surgidos en el debate a raíz de las percepciones, hay uno que me llama especialmente la atención: es la pregunta sobre la sostenibilidad. Consiste en averiguar si el fork es una herramienta que promueve o disuelve la evolución y continuidad de los proyectos de software libre, y si se pueden extraer de ello conclusiones que afecten a otros campos de la producción de artefactos y tecnología. Un concepto que importado del análisis de la relación entre sistemas naturales e industriales de la ecología y ciencias ambientales, se define para el software libre como la seguridad de que un código permanezca siempre abierto y en manos de la comunidad. Concretamente, Nyman *et al.* (2012) lo introducen diciendo que «El fork es el mecanismo más poderoso para luchar contra la obsolescencia programada y preservar la vida de un software» (p. 1) y que «para que un software sea sostenible, este debe evolucionar con sus usuarios» (p. 4). Es desde esta perspectiva, que sostenibilidad y continuidad se relacionan en el software libre no tanto como los costes habidos en su producción, sino desde la óptica de su disponibilidad e interés para terceros. De este modo, intentaré ofrecer también una respuesta a este planteamiento mediante los datos del análisis presentados ulteriormente, llegando con ello a dos de las preguntas centrales del trabajo:

- ¿Es cierta la tesis de Wheeler relativa a los resultados del forking?
- ¿Qué papel juega el fork en relación a la sostenibilidad?

4. Estudio de caso

4.1. *Territorio de análisis*

El trabajo de campo parte de los datos recogidos por los mencionados Robles y Barahona para su artículo «A Comprehensive Study of Software Forks: Dates, Reasons and Outcomes» (2012). Un artículo para el que se confeccionó una memoria de 220 forks relevantes y cuyo interés estriba en ser uno de los primeros intentos de cartografiar este fenómeno. Un documento obtenido de cruzar la información del artículo «List of Software forks»[23] de la Wikipedia en inglés y los 300 primeros resultados arrojados por el algoritmo de Google con los términos *software forks*.

Un extenso documento en el que ofrecen las siguientes referencias de cada unos de los proyectos: página web del proyecto original y el fork, fecha de ocurrencia del fork (*date*), dominio o tipo de software al que pertenecen (*domain*), razones o motivaciones (*reasons*) y resultados o evolución posterior de cada uno de ellos (*outcomes*). Del mismo modo, si navegamos entre esas categorías observamos que hay seis tipos o familias en el campo de motivaciones al fork (*reasons),* que son:

- *Technical*
- *More community-driven development*
- *Discontinuation of the original project*
- *Commercial strategy*
- *Legal issues*
- *Differences among developers teams*[24]

Pues bien; mi investigación se centra en los proyectos incluidos en la categoría de *More community-driven development* en la idea de ofrecer una primera aproximación al análisis comparado de la evolución y tamaño de código fuente de cada par (o grupo) de proyectos de la categoría. Intentaré realizar un acercamiento a la actividades de estas comunidades de software

23. Se puede ver la entrada en: http://en.wikipedia.org/wiki/List_of_software_forks.

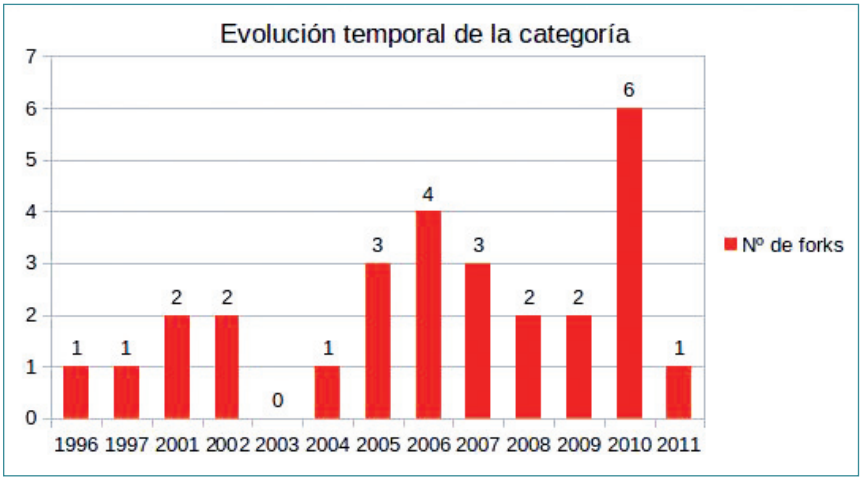
24. Estas 6 categorías son presentadas en el apartado 4.3 del artículo, si bien en la Tabla 2 de la página 10 encontramos dos más. Estas son las de *Experimental* o *friendly forks*, y que es una subcategoría de *Technical*, y Not Found o aquellos de los que no han conseguido averiguar la razón.

libre, a través de la observación del tamaño de sus bases de código fuente y sus resultados evolutivos. Una categoría que representa el 13,2% del total de proyectos identificados en dicha memoria y ocupa el tercer lugar en el orden de razones o motivaciones al forking del estudio resultante. Son 25 programas con sus correspondientes forks, pero como en algunos casos los originales reciben más de una división, estamos ante un escenario de 53 proyectos a localizar. Ordenados temporalmente, la lista de proyectos es la siguiente:

Año	Original	Fork
1996	NetHack	Slash'EM
1997	GCC	EGCS
2001	PHPNuke	PostNuke
2001	Source Forge	Savane
2002	ImageMagick	GraphicsMagick
2002	QTExtended	Opie
2004	freeglut	OpenGLUT
2005	Mambo	Joomla
2005	PHPNuke	RavenNuke
2005	SER	Kamailio
2006	Compiere	Adempiere
2006	Compiz	Beryl
2006	SQL-Ledger	LedgerSMB
2006	Hula	Bongo
2007	CodeIgniter	Kohana
2007	Asterisk	CallWeaver
2007	OpenOffice.org	go-oo.org
2008	Mambo	MiaCMS
2008	TORCS	Speed Dreams
2009	Nagios	Icinga
2009	MySQL	MariaDB
2010	OpenOffice.org	LibreOffice
2010	rdesktop	FreeRDP
2010	GNU Zebra	Quagga
2010	Project Darkstar	Redwarf
2010	Dokeos	Chamilo
2010	SysCP	Froxlor
2011	Redmine	Chiliproject

Fig. 1.

Secuencia temporal



He seleccionado esta categoría por ser una de las que contiene proyectos de software libre que han experimentado separaciones, movidas por problemas de dirección en las comunidades, o por diferencias entre una parte de la comunidad y alguna de las empresas participantes. Nos encontramos por tanto con que en esta categoría hay a su vez dos tipos de rupturas:

a) Forks movidos por la reacción ante decisiones tomadas por la empresa mantenedora del proyecto. Aquí encontramos Asterisk vs Callweaver, CodeIgniter vs Kohana, Adempiere vs Compiere, Compiz vs Beryl, Dokeos vs Chamilo, Hula vs Bongo, Mambo vs Joomla-MiaCMS, MySQL vs MariaDB, Nagios vs Icinga, OpenOffice vs LibreOffice-Go.oo, ProjectDarkstar vs Redwarf, Qtopia vs Opie, Ser vs Kamailio, SourceForge vs Savane, y SQL-Ledger frente a LedgerSMB.

b) Forks creados como resultado de las divergencias en el ritmo de desarrollo impuesto por uno o varios desarrolladores. Tenemos a FreeGlut vs OpenGlut, GCC vs EGCS, GNUZebra vs Quagga, ImageMagick vs GraphicsMagick, NetHack vs Slash'EM, PHPNuke vs PostNuke- RavenNuke, rdesktop vs FreeRDP, Redmine vs Chilliproject, SysCP vs Froxlor y TORCS frente a SpeedDreams.

Digamos por tanto que estamos ante dos subcategorías que convengo llamar «versión empresa» y «versión comunidad». Y en ambos tipos encontramos rupturas impulsadas por casos de relicenciamiento (mayormente hacia licencias privativas o de *copyleft débil*; es decir licencias que prohíben la libertad de uso, copia, modificación o derivación del código, o licencias que no mantienen la viralidad de sus condiciones a las obras derivadas, respectivamente), o forks movidos por el rechazo al ritmo de crecimiento impuesto en un proyecto, traducido en la no incorporación de parches enviados o en la desatención de las mejoras solicitadas por usuarios. Para incluir los proyectos en una de estas divisiones, he recurrido a la información contenida en sus páginas web propias –cuando las hubiera–, entradas en Wikipedia y foros de los proyectos.

Dicha categoría se opone a la de *Commercial strategy*, donde se incluyen proyectos que han sufrido un fork «en dirección contraria»; es decir, escisiones realizadas por empresas que deciden iniciar una versión de desarrollo nueva con la intención de explotarla comercialmente. También encontramos en la memoria la categoría *Differences among developers teams* que, a pesar de poder resultar parecida a la seleccionada, no incluiré en el estudio.

4.2. *Objetivos*

El análisis comparativo pretende ofrecer conclusiones que puedan resultar relevantes desde el punto de vista de la sociología de la tecnología, en concreto de la sociología de las comunidades de software libre. Algo así como si quisiésemos aprender algo sobre el modo de comportamiento o gestión del conflicto de determinadas comunidades indígenas, a través de las consecuencias para su actividad más característica. No pretendo comparar funcionalidades entre proyectos (por ejemplo, cómo funciona LibreOffice frente a OpenOffice o qué características tiene cada uno), sino inferir el nivel de actividad de los equipos a raíz de un fork mediante la observación citada. Intentaré mostrar si la ruptura ha sido positiva para el código desarrollado, examinando su impacto en la continuidad de los equipos de desarrollo y el tamaño de las fuentes publicadas. Un tipo de análisis que, creo también importante señalar, difícilmente podría realizarse en casos de software propietario.

Una investigación que podría completarse con una vertiente cualitativa que siguiera los llamados «métodos mixtos» de investigación en las ciencias sociales[25]. Podría haber confeccionado una entrevista tipo para enviar a los integrantes de cada proyecto con cuestiones como: «¿Has percibido mejoras en el ritmo de desarrollo a raíz del fork?» o «¿competís con el ritmo de evolución del proyecto contrario?», pero es obvio que esta derivada excede con creces las posibilidades, tiempo y alcance de la presente investigación.

4.3. *Recolección de fuentes*

Para la realización del análisis lo primero era obtener los archivos de código fuente de cada proyecto. Para ello he rastreado las páginas web de los distintos proyectos, o navegado por las forjas de código en las que tienen alojado el código (SourceForge, FreeCode o GitHub o Gitorious principalmente). Los proyectos grandes o más antiguos suelen tener el código disponible en su página web, mientras que los más pequeños o nuevos suelen alojarlo únicamente en alguna forja de software. Así obtendremos los objetos o unidades a medir, que son las versiones publicadas o lanzamientos de un software para su uso (conocidas como *releases*).

Es aquí donde reside parte del interés de este trabajo, ya que una recopilación de fuentes semejante no se ha hecho aún en ninguna de las investigaciones del sector. No es un ejercicio automatizable y hay que subrayar su carácter etnográfico, pues aunque la publicación del código sea un imperativo en el software libre, la realidad es luego bien distinta. Los proyectos se organizan de manera autónoma y diseñan o eligen sus sitios e infraestructuras de modo diverso, por lo que no hay un único estándar de publicación o mantenimiento de las versiones del código en la red. Esto suponía que al principio no hubiera seguridad de encontrar las fuentes o, al menos, muchas de ellas. Utilizando de nuevo el ejemplo del antropólogo y sus comunidades indígenas, sería como tener un mapa de ubicación de los asentamientos sin la certeza de que en dichos lugares se sigan guardando los objetos y artefactos que desea analizar.

25. Los métodos mixtos consisten en aunar datos cuantitativos y cualitativos, y es la estrategia seguida para el análisis de los proyectos de ERNST *et al.* (2010).

A continuación la obtención de las *releases* de cada proyecto requirió de un trabajo de análisis y filtrado de proyectos por pares, ya que no todas las fuentes disponibles en los sitios web eran susceptibles de cotejar. Para ello, establecí un criterio comparativo a través de la historia de cada proyecto, consistente en:

1. Establecer la historia de cada fork y averiguar en qué momento y versión del código original sucedió. Una vez localizado este punto, continué el trabajo paralelo de las comunidades, descargando sus releases hasta la desaparición de alguno de los dos (en caso de que fuera así) o el momento actual.
2. Para el caso de proyectos en los que no he detectado la versión replicada o ninguna relación o sucesión evidente entre releases, opté por descargar las bases de código a partir del año del fork, siguiendo el mismo criterio de continuidad anterior. Es decir, para el caso de un proyecto O nacido en 2001 y que sufre un fork F en el año 2005, solo he descargado las versiones de O y F a partir de 2005 y avanzado hasta el abandono de alguno o la actualidad. A su vez, respecto a la numeración de las releases, he optado –como criterio general– por descargar las versiones principales de lanzamiento o el primer y último ejemplar de cada nueva funcionalidad. Puede consultarse la documentación de los materiales descargados en el anexo de investigación[26].

4.4. *Materiales obtenidos*

Una vez explicado el modo de recopilación de los datos y criterio de relación, voy a dar breve cuenta del material obtenido. Así, de los 53 proyectos a localizar, hubo varios en los que no encontré fuente ninguna. Esto puede deberse a que el proyecto ha sido abandonado y desparecido todo archivo, a que haya sido asumido por una empresa que se ha apropiado

26. Dicho «Anexo de Fuentes» está alojado en la red por evidentes razones de espacio (27 pp.). Es un documento público localizable en: https://docs.google.com/file/d/0B053tk8jYhuYkdhTVdJeGhXX2s/edit?usp=sharing.

del código (*hijacking*) o bien porque hubo una integración posterior de los códigos (*merge*). Estos casos han sido:

1. En el par Asterisk/Callweaver no encontré las fuentes del fork.
2. En el par Compiz/Beryl tampoco hay fuentes del fork ya que se fundieron en un nuevo proyecto llamado CompizFusion.
3. En el caso de GCC/EGCS también hicieron merge y ahora forman parte de un misma línea de desarrollo. A pesar de ello, si visitamos su página veremos que las releases son identificadas como GCC o EGCS dependiendo del fragmento temporal que quieras descargar.
4. Para PHPNuke y sus dos forks (PostNuke y RavenNuke) no he localizado versiones de ninguno de los forks.

De cara a los resultados, en todas estas ocasiones he considerado como superviviente al proyecto oponente, haya llegado su desarrollo hasta la actualidad o no. Del mismo modo, en los ejemplos en los que un proyecto original ha recibido más de un fork (tres casos: OpenOffice.org, Mambo y PHPNuke), he valorado sus forks de manera unitaria al considerarlos escisiones ante una misma comunidad original. En cualquier caso esto no ha sido significativo para los datos finales ya que en esas tres situaciones, los forks de OpenOffice hicieron merge entre ellos (Go-oo se unió a LibreOffice) y de las rupturas de PHPNuke no he encontrado las fuentes como comenté. El único caso es el de Mambo, en donde uno de sus forks continúa en actividad (Joomla), mientras que el otro parece abandonado aunque siga disponible su código en la red. Aquí sí he considerado las dos escisiones de manera unitaria de cara a los resultados de continuidad y tamaño. Por último, hay que indicar que el volumen de datos recogidos ha sido de 9,9 GB comprimidos y 27,4 GB extraídos para analizar.

4.5. *Herramientas de medición*

El programa utilizado para la medición del tamaño de los archivos a través de la cantidad de líneas de código, ha sido SLOCCount. Un programa creado en 2001 por David Wheeler para contar líneas físicas del código fuente de proyectos de gran tamaño, en concreto realizar métricas para el proyecto Red Hat Linux. Es utilizado para gran cantidad de programas ya que soporta multitud de lenguajes de programación como son Ada, Assembly,

distintas versiones de shell, C, C++, C#, Cshell, COBOL, Expect, Fortran, Haskell, Java, lex, LISP, makefiles, Modula3, Objective-C, Pascal, Perl, PHP, Python, Ruby, sed, SQL, TCL y Yacc.

Su desarrollo parece maduro y estable, a juzgar por la cantidad de estudios académicos y científicos que lo utilizan. Para nuestros proyectos su uso ha sido estándar, consistente en pasar por este programa cada uno de los archivos de fuentes e ir filtrando sus resultados. De todos los que arroja (estimación del trabajo personas/mes en las fuentes analizadas, estimación al coste de su desarrollo, etc.), solo he utilizado como comenté, la cifra de cantidad de líneas de código físicas[27].

5. Resultados

5.1. *Evolución y continuidad*

A. *Evolución general*

Los primeros datos relevantes se relacionan con los resultados de continuidad, evolución o supervivencia de los proyectos (*outcomes*) y sirven para contrastar la tesis de David Wheeler sobre los escenarios posteriores a la ocurrencia de un fork. Recordemos que para este autor solo había cuatro posibles desenlaces: a) abandono del fork, b) fusión de ambos proyectos, c) abandono del original y d) éxito del fork. A ello le habíamos sumamos dos resultados más: el éxito o continuidad del original como reflejo necesario del primero de los anteriores (abandono del fork) y la desaparición o continuidad de ambos proyectos. Pues bien, si ordenamos la categoría sobre la base de las entradas de resultado que señalan Robles y Barahona en su memoria, en nuestros 25 pares de proyectos encontramos que:

27. Para saber más sobre este programa se puede ver: http://www.dwheeler.com/sloccount/.

Tabla 1.

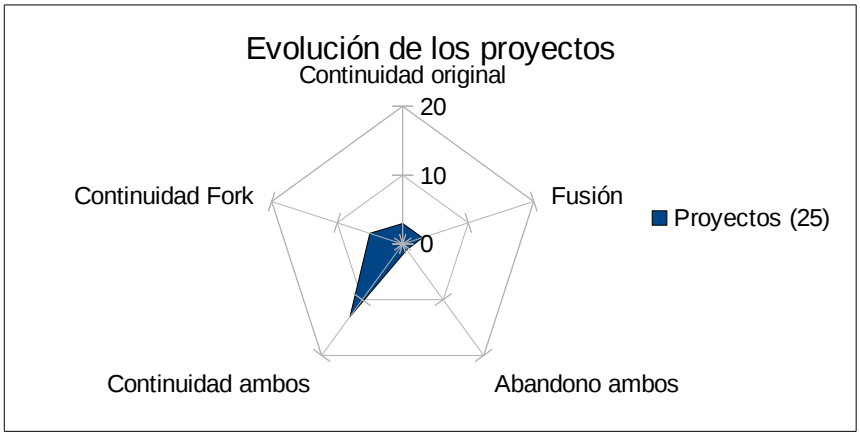
Resultados generales de continuidad

1. En 3 casos el proyecto original ha sobrevivido al fork	12%
2. En 5 casos el fork sobrevive al original	20%
3. En 13 casos, ambos proyectos continúan su desarrollo	52%
4. Son 3 los casos en que ha habido fusión (merge) posterior	12%
5. Hay 1 caso en que original y fork han sido abandonados	4%

Unos resultados que presentados en un gráfico de área, señalan la siguiente tendencia:

Fig. 2.

Evolución general de los proyectos



B. Versión empresa vs versión comunidad

Si ahora clasificamos los datos sobre la base de la distinción interna empresa-comunidad, las tendencias evolutivas son muy similares, como muestran las Figuras 3 y 4.

Fig. 3.

Evolución versión empresa

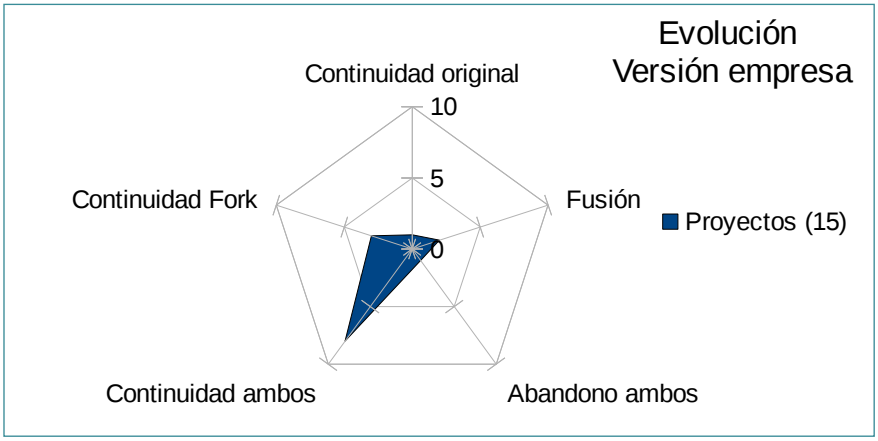
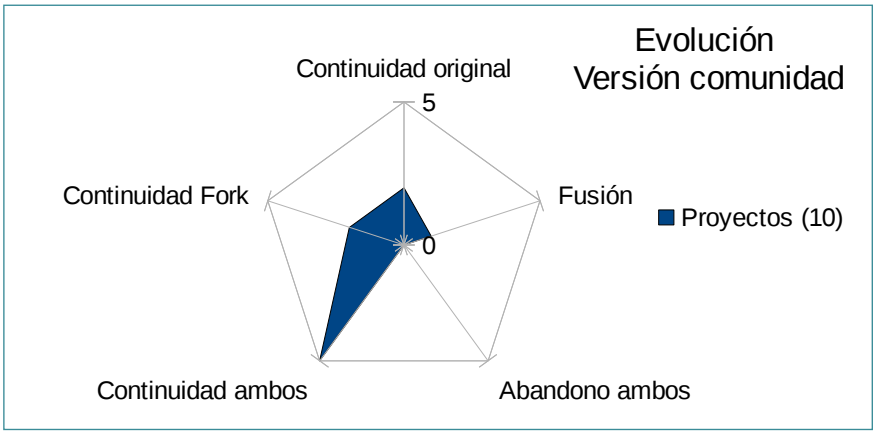


Fig. 4.

Evolución versión comunidad

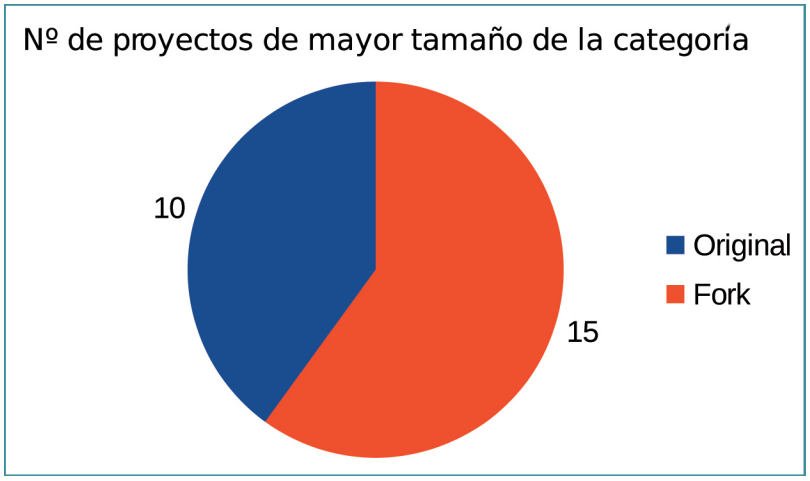


5.2. *Tamaños*

A. Tamaño general

Mediante el número de líneas de código que obtenemos con SLOCCount, vemos que en 15 casos comparados (el 60%) las fuentes de la comunidad escindida tienen mayor tamaño que las fuentes de código del original. Esta cifra resulta relevante como posible indicador de legitimidad en las motivaciones del fork, si la analizamos desde las consecuencias de la ruptura para la comunidad. Es decir, ante los juicios negativos sobre las motivaciones y consecuencias nefastas del forking, vemos que en un 60% de los casos la escisión no solo no ha supuesto la desaparición y muerte del proyecto, sino que su actividad ha sido previsiblemente mayor que la del grupo que quedó en el proyecto original. Bien sea por la falta de desarrolladores, porque el fork tenga un procedimiento más abierto de inclusión de mejoras e integrantes, o debido a que la comunidad escindida trabaja celosamente una vez ocurrida la división, el caso es que las comunidades alternativas no tienen nada que envidiar en principio a la actividad de los proyectos originales.

Fig. 5.
Relación de tamaño general

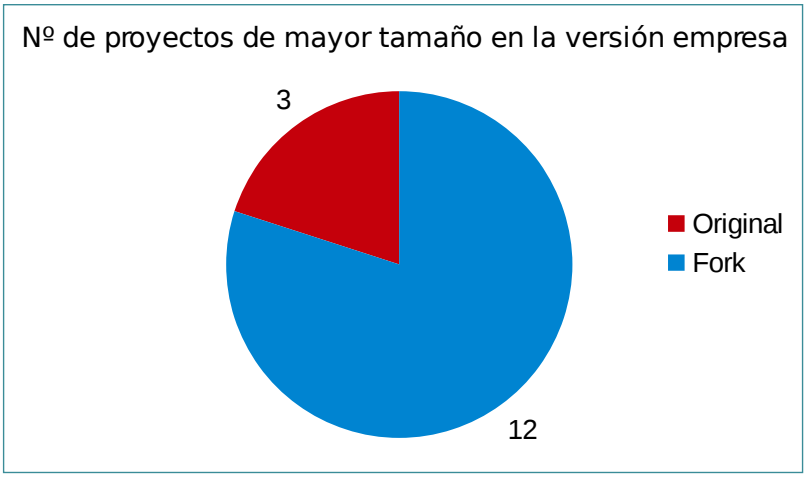


Esto puede sugerir que aquellas personas que impulsaron el fork, tenían de verdad buenas razones para ello, pues parece que han sido capaces de arrastrar o hacer comunidad en torno a su alternativa. Esto queda representado en la figura anterior.

B. Versión empresa vs versión comunidad

Pero si recurrimos nuevamente a los subtipos acordados, la situación cambia notablemente. En los casos en los que el fork fue una reacción ante una maniobra empresarial, de 15 proyectos detectados, son 12 las ocasiones (80%) en las que el fork tiene mayor tamaño, por 3 casos (20%) de los originales. Es lo que representan las Figuras 6 y 7.

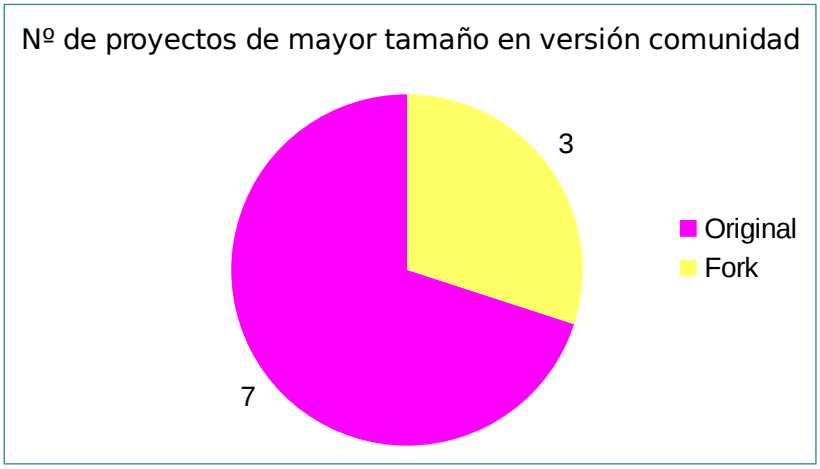
Fig. 6.
Relación de tamaño versión empresa



Sin embargo, en los proyectos de la versión comunidad la situación se invierte. De 10 proyectos incluidos, en 7 casos (70%) el tamaño juega a favor de las iniciativas originales y solo en 3 casos (30%) el fork presenta más líneas de código.

Fig. 7.

Relación de tamaño versión comunidad



En último lugar, al plasmar gráficamente la cantidad de líneas de código, recogidos los proyectos por pares y siguiendo la división empresa-comunidad, obtenemos las figuras 8 y 9.

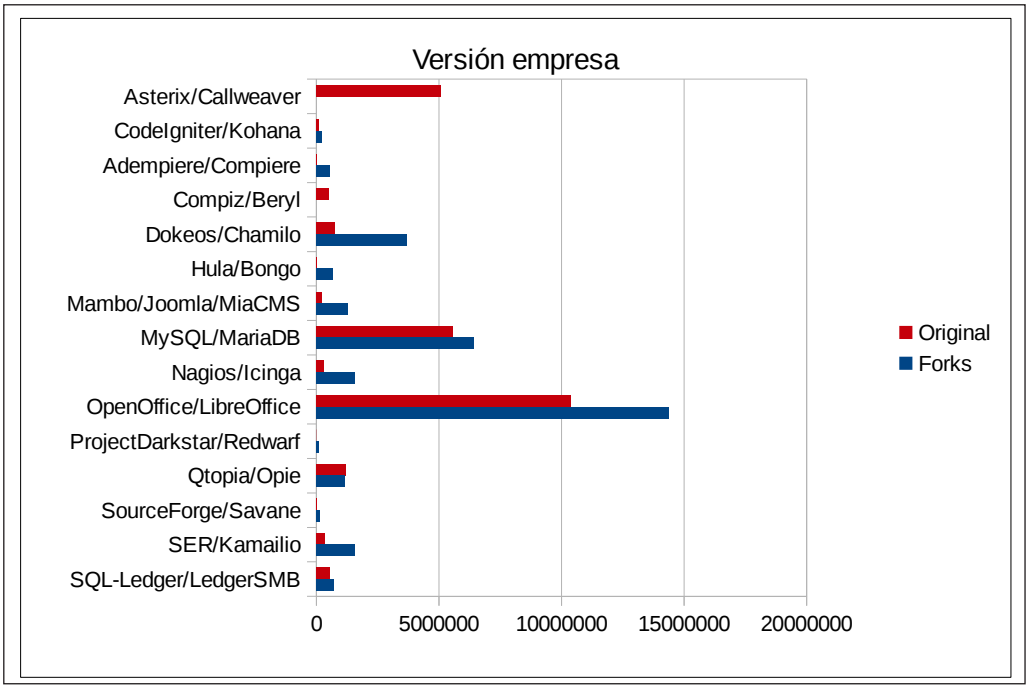
6. Conclusiones para la sociología del Software

Lo primero que hay que señalar es que si recuperamos por un momento la evolución temporal de la figura 1, vemos que la emergencia de forks en esta categoría no sigue el ritmo de crecimiento de proyectos de código libre que los investigadores Amit Deshpande y Dirk Riehle presentan en su artículo «The Total Growth of Open Source» (2008). Así, mientras que el crecimiento de proyectos de software libre es exponencial, la aparición y surgimiento de rupturas experimenta en esta familia altibajos evidentes. Por tanto no hay relación directa entre la cantidad de proyectos iniciados cada año y el número de conflictos aparecidos.

En segundo lugar hay que remarcar lo ya avanzado en los resultados de continuidad presentados en la tabla 1 y figura n.º 2. Y es que queda desmentida para esta categoría la tesis de David Wheeler respecto a la continuidad

Fig. 8.

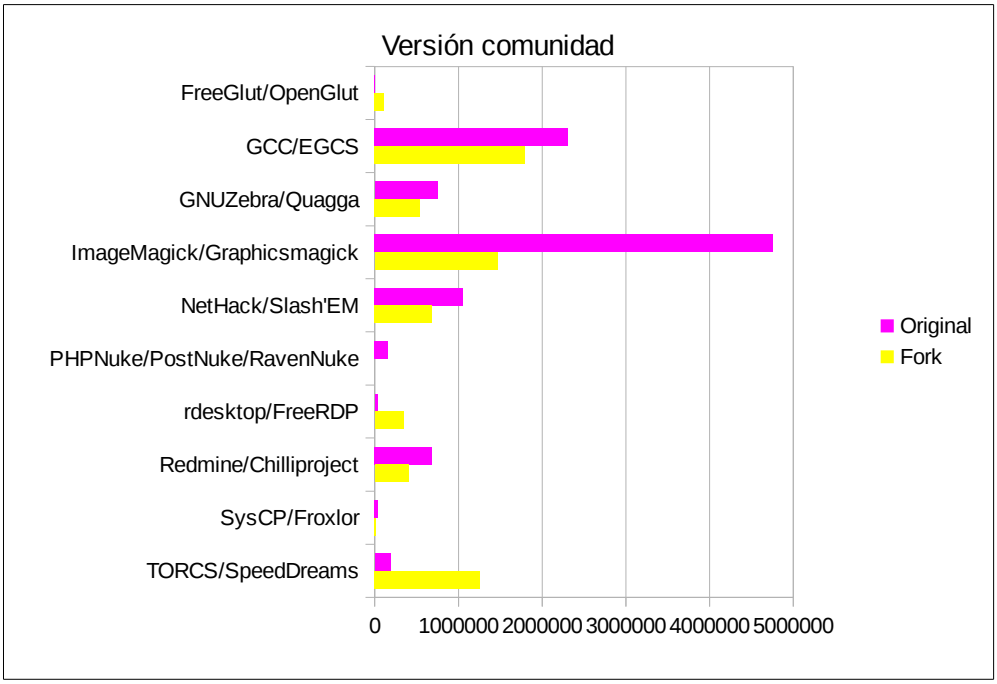
Líneas de código versión empresa



o evolución de los proyectos que sufren un fork. Una rectificación doble, ya que no solo hay más escenarios de los cuatro propuestos, sino que su afirmación sobre el porcentaje de abandono también es falsa. En nuestro grupo, el cese de actividad en proyectos originales es mayor al cese en sus forks y, como acabamos de mostrar, la tendencia evolutiva mayoritaria es la continuidad de ambas versiones. Algo presentado en la Figura 2 y sin especial variación para las Figuras 3 y 4. Por ello, aunque se pueda replicar el dato diciendo que su tesis se refiere al total de forks registrados (en un cierto período de tiempo pongamos), el caso es que su principio no es válido para una de sus secciones aisladas. Máxime cuando hay que tener en cuenta que el *carácter* de la categoría dispuesta podría hacer esperar una solución contraria; esto es, que cuando las comunidades son lideradas por

Fig. 9.

Líneas de código versión comunidad



empresas, estas deberían ser capaces de mantener el ritmo de desarrollo con más facilidad que cuando la discordia irrumpe en colectivos formados exclusivamente por gente voluntaria.

Sin embargo, los resultados presentados en los Gráficos 6 y 7 muestran lo contrario. Cuando una empresa pierde manos voluntarias, el grupo alternativo es capaz de mantener o mejorar el crecimiento del código disponible en un 80% de los casos. Por contra, en las ocasiones en que el fork ocurre en comunidades sin dirección empresarial, solo en el 30% de los casos el fork supera al original. Todo ello puede relacionarse con la definición de fork utilizada, en donde subrayábamos la importancia de la dimensión social de este tipo de replicación. Si el software libre es una actividad íntimamente vinculada a la producción en comunidad, las motivaciones

presentadas para promover una ruptura, han de ser suficientemente avaladas por sus pares. Así ante la inminente separación de un proyecto en otra versión competidora, es crucial ver si son capaces de movilizar a su comunidad. Dicho de otro modo: las justificaciones o razones que tengan para afrontar aquella «lucha y acritud» de que hablara Raymond, han de tener el suficiente respaldo entre su comunidad si quieren que parte de ella (o toda) les siga y su proyecto tenga éxito. Esta es al menos una de las lecturas que podemos aventurar a raíz de los datos obtenidos.

En tercer lugar, las preocupaciones en contra de la sostenibilidad del software que suponía el fork, tampoco se han visto confirmadas. Ya vimos en la tabla 1, que el porcentaje de forks que continúan una vez que el original ha desaparecido es mayor que los casos inversos. Si a esto le sumamos la cifra de la continuidad de ambos, vemos que en casi 3/4 de las escisiones (el 72%) el código fuente sigue disponible. Si a esto le añadimos las tres ocasiones en las que hubo fusión, podemos concluir que el forking no es la causa de desaparición de códigos fuente en esta categoría, sino solo la desaparición de una determinada configuración de comunidad. De este modo concluimos que todos aquellos juicios que tan dura y negativamente juzgaron el papel del fork, no pueden ser fácilmente respaldados. Ni el fork conduce a la muerte de los proyectos, ni su promoción muestra a sus impulsores como «niños petulantes» a juzgar por la comunidad que previsiblemente consiguen arrastrar.

7. Relaciones con la filosofía de la tecnología

He intentado poner de relieve que la dinámica y evolución del *forking* es un proceso que tiene interesantes implicaciones para el estudio del comportamiento y de los valores de las actuales comunidades de software libre. Unas conclusiones que enlazan con la corriente de pensamiento contemporáneo que desde el marco de la filosofía de la tecnología –con propuestas que van desde la axiología hasta los estudios sociales de la ciencia– exige la necesidad de analizar el carácter concreto de las actividades científicas. Y es que si hoy en día la tecnología es uno de los principales vectores del cambio social, los valores que guían su actividad deben sin duda despertar el interés de nuestros estudios filosóficos.

En mi caso, la intención de indagar entre las comunidades de software libre, se debe a que las identifico como el primer exponente de

posicionamiento y asunción –o al menos el más conocido– de la dimensión política en la producción de tecnología moderna. De este modo, las observaciones sobre el comportamiento en la producción del código o software libre como acción social o actividad eminente colectiva, pueden ayudarnos a encontrar patrones interpretativos para algunos de los comportamientos y reacciones de nuestra actual sociedad red y su relación con la producción de nuevos tipos de conocimiento. Muchos de sus contenidos se han extendido a otros sectores de la sociedad que, sin ser específicamente tecnológicos, convergen con algunos de los valores de este movimiento. Así nos encontramos a menudo ante la proliferación de proyectos e iniciativas que podemos colocar bajo el epígrafe de «Cultura Libre» –o todo lo que suele moverse bajo la nomenclatura *open*– con propuestas sociales, culturales o políticas, que entienden que sus procesos internos han de ser lo más transparentes posibles, de cara a facilitar su mejoría a través de la intervención y colaboración de agentes no implicados en su diseño inicial, distribuir las decisiones sobre la dirección de su evolución, o convertirse en un material viral y reutilizable por terceros.

Es por ello que mi trabajo no pretende ser una mera recopilación de datos, sino que propongo hacer avanzar las investigaciones e interpretaciones sobre los valores que guían la actividad creadora de dichos colectivos. Una dirección que como ya destaqué al inicio del texto, señala que hay una *axiología hacke*r aún por detallar[28], realizable a través de la comunicación entre este tipo de colectivos y los estudios sobre actividades científicas –o tecnocientíficas– de los ambientes estrictamente filosóficos. Un acercamiento que nos permitiera ver cuáles son y qué evolución experimentan las motivaciones no solo técnicas sino de otra índole que rigen estas comunidades tan características.

Un planteamiento que enlaza por ejemplo con la propuesta formulada por Echeverría (2002) y que apoyándose en Quintanilla (1988) propugna el análisis de valores nucleares y periféricos en las actividades tecnocientíficas. Y es que asumiendo el indudable pluralismo que debe nutrir la actividad concreta de las comunidades de código libre, parece que mantuvieran en el tiempo un evidente núcleo axiológico. Un núcleo que no es otro que el de

28. En la línea por ejemplo de lo avanzado por Pekka Himanen en *La ética del hacker y el espíritu de la era de la información,* ampliada a otro tipo de valores, como pudieran ser los epistémicos.

la proclamada libertad, entendida en el territorio del software como libertad de uso y copia, y que dichas comunidades siguen reclamando cuando comprobamos que responden con un fork a los posibles intentos de dirección propietaria de sus esfuerzos por parte de empresas, o cuando separan el destino de su trabajo al entender que la gestión de las principales decisiones no está siendo todo lo participativa que debiera.

Del mismo modo otros autores españoles como Broncano, relacionan el análisis de la tecnología con una vertiente más manifiestamente política. Para este autor «si la democracia es el proyecto y la posibilidad de la determinación colectiva y libre del futuro, el control social de las decisiones tecnológicas es uno de los territorios donde se define esa posibilidad» (Broncano, 2000, 226). Una posibilidad que en el mundo moderno se relaciona con un *sujeto plural* «que solamente es legítimo si cada parte [en la toma de decisiones] respeta los valores internos, constitutivos, de las otras partes, y si el acuerdo surge de un proceso *público* de formación de un consenso estable» (Broncano, 2000, 229)[29]. Un análisis por tanto, que aunque brevemente aquí esbozado, parece conectar en profundidad con el comportamiento de estas comunidades de creación de código.

De este modo y partiendo por ejemplo desde estas conexiones, el presente texto ha querido indicar una senda de trabajos que mediante la posibilidad real de las incursiones empíricas, nos permitan ofrecer aproximaciones axiológicas o sociológicas a este tipo de colectivos. Una invitación a incluir este tipo de estudios entre nuestra producción filosófica, dado el evidente interés para nuestro mundo tecnológico actual. Un terreno que además no tiene aún una vía definitiva de acceso, por lo que parece pertinente seguir estudiándolo en el futuro.

29. La cursiva es mía.

8. Bibliografía

BAR, Moshe y FOGEL, Karl (2003): *Open Source Development with CVS*, Scottsdale, Paraglyph Press. Disponible en: http://cvsbook.red-bean.com/OSDevWithCVS_3E.pdf.

BARNES, Bruce H. y BOLLINGER, Terry (1991): «Making reuse cost-effective», *IEEE Software*, 8 (1), pp. 13-24. Disponible en: http://www.idi.ntnu.no/emner/dif8901/papers2003/P-r12-barnes91.pdf.

BEZROUKOV, Nicolai (1999): «A second look at the Cathedral and the Bazar», *First Monday*, 4 (12). Disponible en: http://firstmonday.org/ojs/index.php/fm/article/view/708/618.

BITZER, Jürgen y SCHRÖDER, Philipp J. H. (2005): «The impact of entry and competition by open source software on innovation activity», *International Business Section*, Working Paper 2005-12. Disponible en: http://128.118.178.162/eps/io/papers/0512/0512001.pdf.

BRONCANO, Fernando (2000): «El control social de la tecnología y los valores internos del ingeniero», en *Mundos Artificiales: Filosofía del Cambio Tecnológico*, México, Editorial Paidós.

CAPILUPPI, Andrea; BOLDYREFF, Cornelia y STOL, Klaas-Jan (2011): «Successful reuse of software components: A report from the open source perspective», University of Limerick Institutional Repository. Disponible en: http://ulir.ul.ie/handle/10344/1745.

CROWSTON, Kevin y HOWISON, James (2005): «The social structure of Free and Open Source software developmnet», *First Monday*, 10 (2). Disponible en: http://firstmonday.org/ojs/index.php/fm/article/viewArticle/1207/1127.

DE LA CUEVA, Javier (2013): «Software Libre, ciudadanía virtuosa y democracia», incluido en el blog *Derecho de Internet*. Disponible en: http://derecho-internet.org/node/577.

DESHPANDE, Amit y RIEHLE, Dirk (2008): «The Total Growth of Open Source», en *Proceedings of the Fourth Conference on Open Source Systems*, Springer Verlag. Localizable en: http://dirkriehle.com/publications/2008-2/the-total-growth-of-open-source/.

DIBONA, Cris; OCKMAN, Sam y STONE, Mark (1999): «Introduction», en DIBONA, C.; OCKMAN, S. and STONE, M. (eds.), *Open Sources. Voices from the Open Source Revolution*, O'Reilly, pp. 1-17. Disponible en: http://oreilly.com/openbook/opensources/book/intro.html.

ECHEVERRÍA, Javier (2001): «Tecnociencia y Sistema de Valores», en LÓPEZ CEREZO, J. A. y SÁNCHEZ RON, J. M. (eds.), *Ciencia, Tecnología, Sociedad y Cultura en el cambio de siglo*, Madrid, Biblioteca Nueva/OEI, pp. 221-230.

ECHEVERRÍA, Javier (2002): *Ciencia y Valores*, Barcelona, Editorial Destino.

ERNST, Neil A.; EASTERBROOK, Steve y MYLOPOULOS, John (2010): «Code forking in open-source software: a requirements perspective», en *arXiv.org*, abs/1004.2889. Disponible en: http://arxiv.org/abs/1004.2889.

FELLER, Joseph y FITZGERALD, Brian (2000): «A Framework Analysis of the Open Source Software Development Paradigm», en *21st International Conference of Information Systems*, pp. 58-69, Atlanta. Disponible en: http://ifipwg213.org/system/files/p58-feller.pdf.

FOGEL, Karl (2005): *Producing Open Source Software-How to run a successful free software project*, O'Reilly. Disponible en: http://producingoss.com/.

FUNG, Kam H.; AURUM, Aybücke y TANG, David (2012): «Social Forking in Oper Source Software: An Empirical Study», en *Proceedings of the CAISE'12 Forum at the 24th International Conference on Advanced Information Systems Engineering (CaiSE)*, pp. 50-57. Disponible en: http://ceur-ws.org/Vol-855/paper6.pdf.

GAMALIELSSON, Jonas y LUNDELL, Björn (2012): «Long-Term Sustainability of Open Source Software Communities beyond a Fork: A Case Study of LibreOffice», *Open Source Systems: Long-Term Sustainability,* 8th IFIP WG 2.13 International Conference, OSS 2012, Hammamet, Tunisia, September 10-13, 2012. Proceedings, pp. 29-47. Disponible en: http://link.springer.com/chapter/10.1007%2F978-3-642-33442-93.

HAEFLIGER, Stefan; VON KROGH, Georg y SPAETH, Sebastian (2008): «Code Reuse in Open Source Software», *Management Science*, vol. 54, n.º 1, pp. 180-193. Disponible en: http://mansci.highwire.org/content/54/1/180.abstract.

HIMANEN, Pekka (2001): *La ética del hacker y el espíritu de la era de la información*, Barcelona, Editorial Destino.

ISODA, Sadahiro (1995): «Experiences of a software reuse project», *Journal of Systems and Software*, volumen 30, pp. 171-186. Disponible en: http://dl.acm.org/citation.cfm?id=211872.

KUUSIRATI, Henri y SEPPÄNEN, Janne (2012): «Forks in Open Source Software Projects». Disponible en: https://wiki.oulu.fi/download/attachments/28092087/ossd_2012_kuusirati_sepp anen.pdf.

LERNER, John y TIROLE, Jean (2002): «Some simple economics of open source». *JOURNAL OF INDUSTRIAL ECONOMICS*, 50 (2), pp. 197-234. Disponible en: https://intranet.cs.aau.dk/fileadmin/user_upload/Education/Courses/2010/ENT/Lerner Tirole_2002.pdf.

MATEOS GARCÍA, Juan y STEINMUELLER, Edward W. (2003): «Applying the Open Source Development Model to Knowledge Work», *INK Open Source reasearch working paper n.º 2*. Disponible en: http://ideas.repec.org/p/sru/ssewps/94.html.

MOCKUS, Audrey; FIELDING, Roy y HERBSLEB, James (2002): «Two Case Studies Of Open Source Software Development: Apache And Mozilla», *ACM Transactions*

on Software Engineering and Methodology. Disponible en: http://mockus.us/papers/mozilla.pdf.

MOEN, Rick (1999): «Fear of Forking-How the GPL Keeps Linux Unified and Strong», *Linuxcare*, noviembre. Disponible en: http://linuxmafia.com/faq/Licensing_and_Law/forking.html.

MOODY, Glyn (2009): «Who owns commercial open source and can forks work?», *Linux Journal*, 23 de abril. Disponible en: http://www.linuxjournal.com/content/who-owns-commercial-open-source-%E2%80%93-and-can-forks-work. (2011): «The Deeper Significance of LibreOffice 3.3», *ComputerWorld UK*, enero. Disponible en: http://blogs.computerworlduk.com/open-enterprise/2011/01/the-deeper-signific ance-of-libreoffice-33/index.htm.

NEVILLE-NEIL, George V. (2011): «Think before you fork», *Communications of the ACM*, 54, pp. 34-35, junio. Disponible en: http://dl.acm.org/citation.cfm?doid=1953122.1953137.

NYMAN, Linus (2011): «Understanding Code Forking in Open Source Software». Disponible en: http://ossconf.org/2011/oss2011-doctoralconsortium.pdf#page=72.

NYMAN, Linus y MIKKONEN, Tommi (2011): «To Fork or Not to Fork: Fork Motivations in SourceForge Projects», *International Journal of Open Source Software and Processes*, volumen 3, pp. 1-9. Disponible en: http://www.igi-global.com/journal/international-journal-open-source-software/1123.

NYMAN, Linus *et al.* (2011): «Forking: the Invisible Hand of Sustainability in Open Source Software», *Proceedings of SOS 2011: Towards Sustainable Open Source*. Disponible en: http://tutopen.cs.tut.fi/sos11/papers/cr4.pdf.

NYMAN, Linus *et al.* (2012): «Perspectives on Code Forking and Sustainability in Open Source Software: the Invisible Hand of Sustainability in Open Source Software», en *The proceedings of the 8th International Conference on Open Source Systems*. Disponible en: http://flosshub.org/content/perspectives-code-forking-and-sustainability-open-s ource-software.

NYMAN, Linus y LINDMAN, Juho (2013): «Code Forking, Governance, and Sustainability in Open Source Software», T*echnology Innovation Management Review*, vol. January, pp. 7-12. Disponible en: http://timreview.ca/sites/default/files/article_PDF/NymanLindman_TIMReview_January2013.pdf.

QUINTANILLA, Miguel Ángel (1988): *Tecnología: un enfoque filosófico*, Madrid, Fundesco.

RAYMOND, Eric S. (1996): *The Jargon File.* Localizable en: http://www.catb.org/jargon/.

ROBLES, Gregorio y GONZÁLEZ-BARAHONA, Jesús M. (2009): «Evolution of the core team of developers in libre software projects», en *MSR '09 Proceedings of the 2009 6th IEEE International Working Conference on Mining Software Repositories*, pp. 167-170. Disponible en: http://dl.acm.org/citation.cfm? id=1591154.

ROBLES, Gregorio; GONZÁLEZ-BARAHONA, Jesús M. (2012): «A Comprehensive Study of Software Forks: Dates, Reasons and Outcomes», en *OSS 2012 The Eighth International Conference on Open Source Systems*. Disponible en: http://flosshub.org/content/comprehensive-study-software-forks-dates-reasons-and-outcomes.

RUBENS, Paul (2010): «Open Source Software comes to a fork in the code». Disponible en: http://www.serverwatch.com/trends/article.php/3896671/Open-Source-Software-Comes-to-a-Fork-in-the-Code.htm.

SCACCHI, Walter (2010): «Computer game mods, modders, modding, and the mod scene», *First Monday*, 15(5). Disponible en: http://firstmonday.org/ojs/index.php/fm/article/view/2965.

VISEUR, Robert (2012): «Forks impacts and motivations in free and open source projects». Disponible en: http://www.robertviseur.be/download/ijacsa-rv-umons-forks-2012.pdf.

WHEELER, David A. (2007): «Why Open Source Software/Free Software (OSS/FS, FLOSS, or FOSS)? Look at the Numbers!». Disponible en: http://www.dwheeler.com/oss_fs_why.html#forking.

WINNER, Langdon (1987): *La Ballena y el reactor: una búsqueda de los límites en la era de la alta tecnología*, Barcelona, Gedisa.