novática
CEPIS UPGRADE Revista de la Asociación de Técnicos de Informática

Nº 190, noviembre-diciembre 2007, año XXXIII

sumario

Monografía del próximo número: "*Gobierno de las TIC*"

Novática, revista fundada en 1975 y decana de la prensa informática española, es el órgano oficial de expresión y formación continua de **ATI** (Asociación de Técnicos de Informática), organización que edita también la revista **REICIS** (Revista Española de Innovación, Calidad e Ingeniería del Software). ***Novática*** edita asimismo **UP**GRADE, revista digital de **CEPIS** (*Council of European Professional Informatics Societies*), en lengua inglesa, y es miembro fundador de **UP**ENET (**UP**GRADE ***E**uropean **NET**work*).

<http://www.ati.es/novatica/>
<http://www.ati.es/reicis/>
<http://www.upgrade-cepis.org/>

ATI es miembro fundador de **CEPIS** (*Council of European Professional Informatics Societies*) y es representante de España en **IFIP** (*International Federation for Information Processing*); tiene un acuerdo de colaboración con **ACM** (*Association for Computing Machinery*), así como acuerdos de vinculación o colaboración con **AdaSpain**, **AI2, ASTIC, RITSI** e **Hispalinux**, junto a la que participa en **ProInnova**.

Coordinación Editorial, Redacción Central y Redacción ATI Madrid
Padilla 66, 3º, dcha., 28006 Madrid
Tlfn.914029391; fax.913093685 <novatica@ati.es>
Composición, Edición y Redacción ATI Valencia
Av. del Reino de Valencia 23, 46005 Valencia
Tlfn./fax 963330392 <secreval@ati.es>
Administración y Redacción ATI Cataluña
Via Laietana 46, ppal. 1ª, 08003 Barcelona
Tlfn.934125235; fax 934127713 <secregen@ati.es>
Redacción ATI Andalucía
Isaac Newton, s/n, Ed. Sadiel,
Isla Cartuja 41092 Sevilla, Tlfn./fax 954460779 <secreand@ati.es>
Redacción ATI Aragón
Lagasca 9, 3-B, 50006 Zaragoza.
Tlfn./fax 976235181 <secreara@ati.es>
Redacción ATI Asturias-Cantabria <gp-astucant@ati.es>
Redacción ATI Castilla-La Mancha <gp-clmancha@ati.es>
Suscripción y Ventas <http://www.ati.es/novatica/interes.html>, ATI Cataluña, ATI Madrid
Publicidad
Padilla 66, 3º, dcha., 28006 Madrid
Tlnf.914029391; fax.913093685 <novatica@ati.es>
Imprenta: Derra S.A., Juan de Austria 66, 08005 Barcelona.
Depósito legal: B 15.154-1975 – ISSN: 0211-2124; CODEN NOVAEC
Portada: "Salida de la habitación 101" - Concha Arias Pérez / © ATI
Diseño: Fernando Agresta / © ATI 2003

De la catedral al bazar: un estudio empírico del ciclo de vida de los proyectos basados en comunidades de voluntarios

Andrea Capiluppi[1], Martin Michlmayr[2]
[1]Universidad de Lincoln, Reino Unido; [2]Universidad de Cambridge, Reino Unido

<acapiluppi@lincoln.ac.uk
<martin@michlmayr.org>

Traducción: Israel Herraiz Tabernero (Universidad Rey Juan Carlos)

Resumen: *en el pasado, algunos proyectos de software libre han tenido mucho éxito. A menudo, el éxito de un proyecto de software depende del número de desarrolladores que es capaz de atraer: una comunidad grande (el "bazar") encuentra y arregla más defectos en el software y añade más características nuevas mediante un proceso de revisión por pares. En este artículo estudiamos dos proyectos de software libre (Wine y Arla) desde un punto de vista empírico, con el fin de caracterizar el ciclo de vida del software, los procesos de desarrollo y la comunidad en la que se basa el proyecto.*

Palabras clave: *desarrolladores de software, evolución del software, fases, procesos de software, software libre.*

Autores

Andrea Capiluppi es Doctor en Informática por el Politécnico de Turín (Italia). Fue investigador visitante en el Grupo de Sistemas y Comunicaciones de la Universidad Rey Juan Carlos (Madrid) en octubre de 2003. Desde enero de 2004 hasta la actualidad ha sido investigador visitante en el Departamento de Matemáticas y Computación de la Open University (Reino Unido), trabajando junto con los doctores Juan Ramil, Neil Smith, Helen Sharp, Alvaro Faria y Sarah Beecham. Este compromiso ha sido renovado hasta diciembre de 2008. En enero de 2006, entró a formar parte de la Universidad de Lincoln como profesor senior.

Martin Michlmayr ha estado involucrado en varios proyectos de software libre desde hace más de 10 años. Fue coordinador de voluntarios para el proyecto GNUstep, y actuó como Director de Publicidad para Linux International. En el año 2000, se unió al proyecto Debian, y fue elegido más tarde *Debian Project Leader* (DPL), puesto que ocupó durante dos años. Martin posee títulos de Máster en Filosofía, Psicología e Ingeniería del Software, y un doctorado por la Universidad de Cambridge. En la actualidad trabaja para HP como experto en comunidades de software libre.

1. Introducción

Los proyectos de software libre más conocidos, tales como Linux [32], Apache [27] y FreeBSD [18] han tenido un éxito tremendo. Hasta ahora, las pruebas para caracterizar el éxito de un proyecto de software libre han sido anecdóticas: más usuarios y/o desarrolladores suponían más "ojos" para localizar fallos, los desarrolladores implementaban nuevas características de manera independiente, y los líderes de proyectos gestionaban una estructura prácticamente horizontal con los consecuentes costes de coordinación [28].

Estudios previos han proporcionado pruebas empíricas de los procesos de desarrollo empleados en proyectos de software libre que tienen éxito: por ejemplo, la definición de diferentes tipos de desarrolladores para los proyectos Apache y Mozilla, que justificaba diferentes niveles de esfuerzo [27], y concluía que el primer tipo, los desarrolladores principales (*core developers*), son los que contribuyen al éxito de un proyecto. También, el análisis de redes sociales ha mostrado cuáles son los costes de comunicación y coordinación en proyectos de software libre de éxito [21].

En todos estos casos, los proyectos que tienen éxito se estudian y se caracterizan, pero no se analiza cómo comenzaron. Por tanto, no se han realizado estudios empíricos que muestren si el proyecto se benefició siempre de la participación de un gran número de desarrolladores, o por el contrario la fase de bazar[1] se alcanzó tras años de desarrollo. Para cubrir este hueco, este artículo explora la evolución y los procesos de desarrollo de dos sistemas de software libre, el proyecto Wine (una implementación libre de Windows para Unix) y el sistema de ficheros Arla. El primero de ellos se ha extendido entre muchos desarrolladores, que también han contribuido extensamente. Arla, por el contrario, está todavía en la fase "catedral"[1] si lo comparamos con Wine: tiene menos desarrolladores que estén dirigiendo el desarrollo del proyecto.

El propósito de este artículo es detectar y caracterizar, empíricamente, las fases alcanzadas por los dos proyectos mencionados, con el fin de ilustrar si una de las fases aparece a continuación y como consecuencia de la otra, y de proponer una de esas fases como "éxito" para un proyecto de software libre. En tal caso, compartir la metodología empírica para llevar a cabo la transición entre fases podría ayudar a los desarrolladores a trabajar en los beneficios que presenta la fase de bazar.

A continuación, en la **sección 2,** se muestran los fundamentos teóricos, además de dos cuestiones a investigar, basados en comunidades de software libre. Además, se presenta una descripción del enfoque usado para adquirir y analizar los datos empleados en el estudio. Estos datos se emplean para com-

[1] **Nota del editor:** Como se cita posteriormente a lo largo del artículo, la distinción entre las fases de "catedral" y "bazar" en los proyectos de software libre proviene de Eric S. Raymond [28]. En Edukalibre <http://collab.edukalibre.org/docs/un_libro/sobre_gordo/ch07s02.html> nos cuentan que: "Dentro de lo que Raymond toma como el modelo de creación de catedrales no sólo tienen cabida los procesos pesados que podemos encontrar en la industria del software (el modelo en cascada clásico, las diferentes vertientes del Rational Unified Process, etc.), sino que también entran en él proyectos de software libre, como es el caso de GNU y NetBSD. Para Raymond, estos proyectos se encuentran fuertemente centralizados, ya que unas pocas personas son las que realizan el diseño e implementación del software [...].El modelo antagónico al de la catedral es el bazar. Según Raymond, algunos de los programas de software libre, en especial el núcleo Linux, se han desarrollado siguiendo un esquema similar al de un bazar oriental. En un bazar no existe una máxima autoridad que controle los procesos que se están desarrollando ni que planifique estrictamente lo que ha de suceder".

probar las cuestiones a investigar propuestas. En la **sección 3**, se describen las fases observadas en los dos sistemas desde el punto de vista de las actividades de los desarrolladores. Esta sección también muestra una descripción detallada de las actividades que sustentan el éxito de un proyecto de software libre, tal y como se han observado en los casos de estudio propuestos. La **sección 4** trata del trabajo relacionado en ésta y otras áreas, identificando cuáles son las contribuciones principales de este artículo, y discute algunas cuestiones adicionales que aparecen en el artículo y que necesitan más estudios futuros. Por último, la **sección 5** muestra las conclusiones sobre el proceso general y el ciclo de vida de un proyecto de software libre, además de algunas indicaciones para trabajos futuros.

2. Antecedentes teóricos

Uno de los autores de este artículo, en un trabajo previo [29], presentó una teoría para las diferentes actividades y fases del ciclo de vida de un proyecto de software libre. El objetivo era proporcionar un enfoque sistemático para el desarrollo de proyectos de software libre, es decir, incrementar la probabilidad de éxito en proyectos nuevos. En este artículo, el objetivo es evaluar de un modo empírico la teoría contenida en el mencionado trabajo, a través de dos casos de estudio, e informar de cuáles son las mejores prácticas de proyectos de software libre reales de éxito. Dado que algunos trabajos previos han mostrado que muchos proyectos de software libre deberían considerarse como fracasos [3][7], se muestra que estos proyectos no presentan algunas de las características mencionadas en [29], principalmente la transición entre el estilo cerrado (o "catedral") y abierto (o "bazar").

En su popular ensayo *The Cathedral and the Bazaar*, Eric S. Raymond [28] investiga las estructuras de desarrollo de proyectos de software libre, basándose en el éxito de Linux. La terminología "catedral" y "bazar" presenta tanto un enfoque cerrado, que se encuentra en la mayoría de entidades comerciales, donde las decisiones sobre un proyecto grande de software se toman mediante una gestión centralizada, como un enfoque abierto, donde una comunidad entera es la responsable de la gestión de todo el sistema.

En lugar de presentar estos dos enfoques como diametralmente opuestos (tal y como proponía originalmente Raymond), este artículo considera que son eventos complementarios dentro de un mismo proyecto de software libre. La **figura 1** muestra las tres fases básicas, las cuales están presentes en un proyecto de software libre con éxito, según la tesis defendida en este artículo. La fase inicial de un proyecto de software libre no opera en el contexto de una comunidad de voluntarios. Todas las características del estilo catedral (recogida de requisitos, diseño, implementación, *testing*) están presentes en esta fase, y también en el estilo de construcción típico de una catedral, es decir, el trabajo lo realiza un individuo o un pequeño grupo de desarrolladores aislados de la comunidad [5]. Este proceso de desarrollo muestra un control estricto y una planificación centralizada y realizada por el autor principal, que ha sido denominada "prototipado cerrado" por Johnson [17].

Según [29], un proyecto de software libre tiene que realizar una transición de la fase catedral a la fase bazar, con el fin de convertirse en un producto útil y de calidad (tal y como se muestra con la flecha en la **figura 1**). En esta fase, nuevos usuarios y desarrolladores se unen de manera continua al proyecto, escribiendo código, enviando parches y arreglando fallos. Esta transición se asocia con bastantes complicaciones: por ejemplo, algunos estudios [7] concluyen que la mayoría de proyectos de software libre nunca abandonan la fase de catedral y por tanto no acceden a la vasta cantidad de recursos que la comunidad de software libre ofrece en forma de mano de obra y habilidades.

2.1. Cuestiones a investigar

En este artículo, se analizan datos históricos sobre las modificaciones y adiciones de secciones a gran escala (subsistemas) o pequeña escala (módulos) a un sistema de software, con el fin de trazar cómo han evolucionado en el tiempo los casos de estudio. Se proponen dos cuestiones a investigar, que se validarán frente a los datos históricos. Esto se realiza en la siguiente sección, donde también se muestran los resultados. La primera cuestión se basa en la respuesta del proyecto a un estímulo, y la segunda se refiere al tipo de trabajo al que suelen dedicarse los desarrolladores cuando comienzan su colaboración con el proyecto. Estas cuestiones se formulan a continuación (se incluyen también las métricas necesarias para validarlas):

1) Cuestión 1: la fase bazar supone un crecimiento en el número de desarrolladores, que se unen en un ciclo auto-sostenido. El resultado obtenido en esta fase muestra un patrón de crecimiento similar. Los proyectos de software libre no se benefician de esta tendencia creciente durante la fase catedral.
2) Cuestión 2: Cuando un desarrollador nuevo se une al proyecto, tiende a trabajar primero en los módulos más nuevos, ya sea creando ellos mismos el módulo o contribuyendo a un módulo de reciente creación. Esto puede explicarse argumentando que un desarrollador no necesitaría conocer toda la funcionalidad ya existente en el sistema para desarrollar una parte nueva del sistema. Esta cuestión a investigar se emplea en este artículo para proponer cómo Wine pudo alcanzar la fase de bazar.

2.2. Metodología empírica

El enfoque empírico supone la extracción de todos los cambios que se encuentran tanto en la entrada (esfuerzo proporcionado por los desarrolladores) como en la salida (cambios y nuevo código en los subsistemas y módulos) del proyecto. A continuación, en lugar de analizar los repositorios CVS de los proyectos, se analiza el fichero de *ChangeLog*[2], que guarda toda la historia de cambios del proyecto. Estudios previos [10][22] muestran que diferentes prácticas

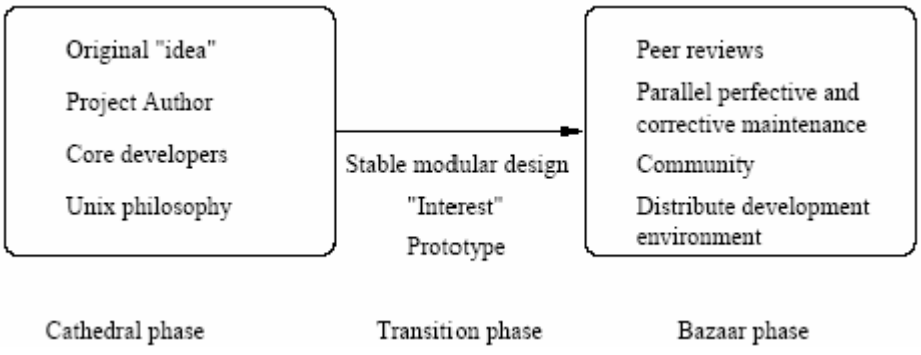

Figura 1. Ciclo de desarrollo de un proyecto de software libre.

de desarrollo tienen impacto en cuál es la mejor fuente de datos para analizar el proyecto; el fichero de *ChangeLog* proporciona más fiabilidad en los casos de estudio seleccionados [6][12][30].

Los pasos que hay que seguir para producir los datos necesarios para el estudio se resumen a continuación: se interpretan los datos de los ficheros de cambio y se extraen las métricas. Como parte del primer paso, se han escrito *scripts* en Perl para interpretar los datos contenidos en los ficheros de *ChangeLog*, y para extraer algunos campos de datos predefinidos. Los campos que se contemplan en este estudio son: el nombre del sistema, el nombre del módulo, el nombre del subsistema que contiene ese módulo, la fecha de creación o cambio y un identificador único (nombre y dirección de correo electrónico) del desarrollador responsable del cambio.

2.2.1. Extracción de datos

Los ficheros de *ChangeLog* analizados siguen un patrón muy regular, por lo que el análisis de los cambios producidos en la historia del proyecto puede realizarse muy fácilmente y de un modo casi automático. Para extraer los datos se han seguido los pasos siguientes:

1. Identificación de las fechas: en los casos de estudio se observó que cada cambio estaba delimitado por una fecha, usando el siguiente patrón o uno similar: AAAA-MM-DD, como en "2000-12-31". Cada cambio puede asociarse con uno o más desarrolladores; además, cada cambio puede asociarse con uno o más módulos. Sin embargo, sólo hay una fecha para cada cambio.
2. Módulos y subsistemas afectados: cada cambio afecta al menos a un fichero, y se guarda con una descripción en texto plano. En algunos casos, el mismo cambio afecta a varios ficheros: estas modificaciones presentan siempre la misma fecha. Los subsistemas se extraen como el directorio que contiene al fichero afectado por el cambio.
3. Detalles de los desarrolladores: Todos los cambios involucran al menos a un desarrollador, que se puede mostrar de varias maneras diferentes en la descripción del cambio. Si el cambio se debe a más de un desarrollador, todos los desarrolladores aparecen juntos en la descripción del cambio.
4. Cálculo de las métricas: Se calcularon tanto el esfuerzo de los desarrolladores como el trabajo producido creando nuevos módulos y corrigiendo módulos existentes.

2.2.2. Elección y descripción de las métricas

El análisis de los dos proyectos de software libre se realizó mediante tres tipos de métricas, que se usan de un modo diferente para discutir cada una de las cuestiones a investigar. La lista de métricas propuestas se muestra a continuación:

Métricas de entrada: el esfuerzo de los desarrolladores se evaluó contando el número de desarrolladores únicos (o distintos, por usar una terminología similar a SQL[3]) durante un intervalo específico de tiempo. La granularidad temporal que se ha seleccionado es la de meses: se pueden emplear diferentes enfoques, como contar los desarrolladores semanalmente o diariamente, pero creemos que un mes es una unidad temporal con una granularidad más indicada para extraer el número de desarrolladores activos. Estas métricas se han usado para evaluar la primera cuestión a investigar. Por ejemplo, durante febrero de 2006 el proyecto Wine tuvo 73 desarrolladores diferentes que escribieron código.

Métricas de salida: el trabajo producido se evaluó contando el número de cambios a los módulos o subsistemas durante el mismo intervalo de tiempo. No se han considerado métricas de granularidad fina, como por ejemplo líneas de código. Evaluar la producción de código por parte de los desarrolladores usando líneas de código hubiera supuesto importantes limitaciones al estudio[4]. En la siguiente sección se usará esta métrica como un indicador del trabajo de desarrollo en paralelo realizado en proyectos con éxito. Esta métrica se ha usado también para evaluar la primera cuestión a investigar. Siguiendo el ejemplo del párrafo anterior, en febrero de 2006 se detectó que en Wine había 820 módulos que habían sido modificados durante ese mes.

Métricas de nuevas entradas y salidas: el esfuerzo nuevo añadido al proyecto se midió como el número de desarrolladores que se unen al proyecto. Además, se aisló el trabajo que se debía a estos nuevos desarrolladores: el objetivo es determinar cuánto de este trabajo se centró en partes del sistema que existían previamente, y cuánto en partes nuevas. Estas métricas se han empleado para evaluar la segunda cuestión a investigar, esto es, para explorar si los desarrolladores nuevos tienden a trabajar en partes viejas o nuevas del sistema. Siguiendo el ejemplo de los párrafos anteriores, se detectó que durante febrero de 2006 se unieron 73 nuevos desarrolladores a Wine (esto es, los desarrolladores no fueron detectados en ningún cambio previo a esa fecha). Además, empíricamente se detectó que estos nuevos desarrolladores trabajan tanto en partes viejas como nuevas del sistema (añadidas ese mismo mes). Se observó que el 75% de su trabajo se producía en partes nuevas, y el 25% en partes que ya existían previamente.

2.3. Casos de estudio

La elección de los casos de estudio se realizó basándose en que uno (Wine) es un proyecto de éxito objetivo, mientras que el otro (Arla) parece haber sufrido problemas a la hora de reclutar a nuevos desarrolladores, alcanzando un tamaño mucho menor. Los dos proyectos se habían usado previamente para otros estudios empíricos, y se había estudiado ampliamente su estilo de desarrollo y patrón de crecimiento.

Los autores reconocen que los sistemas pertenecen a campos de aplicación muy diferentes: Wine es una herramienta para ejectuar aplicaciones Windows sobre Linux y otros sistemas operativos, mientras que Arla es un sistema de ficheros en red. El objetivo principal de estudio no era evaluar las razones exógenas detrás del éxito de estos proyectos

Atributo / Sistema	Arla	Wine
Entrada más antigua	Octubre 1997	Julio 1993
Última entrada	Marzo 2006	Marzo 2006
Número de cambios	7.000	88.000
Desarrolladores distintos (total)	83	880

Tabla 1. Resumen de la información relativa a los dos casos de estudio.

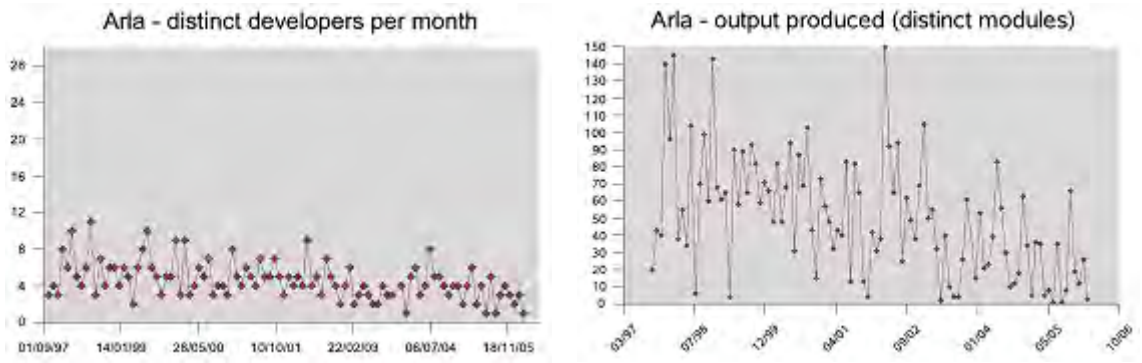

Figura 2. Número de desarrolladores (izquierda) y trabajo producido (derecha) en el caso de Arla.

a la hora de atraer desarrolladores (como por ejemplo la presencia de "gurús" en el proyecto, la buena reputación de la comunidad, etc. [9]). Al contrario, este estudio se centra en evaluar la presencia de tres etapas diferentes en proyectos que tienen éxito. El trabajo que se muestra en este artículo propone un marco teórico para proyectos de software libre, independientemente de su dominio de aplicación, y evalúa de manera empírica los mecanismos por los que se forman las comunidades alrededor de proyectos de software libre.

Se ha restringido la elección de las fuentes de información a dos tipos: los cambios realizados en el CVS y el fichero de *ChangeLog*. El repositorio CVS de Arla resultó estar *incompleto*, dado que no contenía la historia completa de la evolución del proyecto. Esto se debe con probabilidad al hecho de que el servidor CVS comenzó a usarse en algún momento posterior al comienzo del proyecto. Además, se observó que el repositorio CVS de Wine resultó ser *inexacto*: al consultar el número de desarrolladores en activo se obtuvo que había sólo 2 desarrolladores, mientras que los ficheros de *ChangeLog* contienen una cantidad mucho mayor de desarrolladores diferentes. Esto se debe probablemente a alguna restricción en el permiso de escritura en el repositorio. Debido a esto, era preferible usar los ficheros de *ChangeLog* que el registro de cambios del CVS.

La **tabla 1** muestra información acerca de los ficheros de *ChangeLog*, el tiempo de vida de los proyectos, y la cantidad de desarrolladores distintos, con el fin de caracterizar a los dos sistemas.

3. Resultados y discusión acerca de las fases

En esta sección, basándonos en datos empíricos de los dos casos de estudio, se discuten las dos cuestiones a investigar, y se evalúan las tres fases (catedral y bazar, separadas por una transición), tal y como se presentan en [29].

Además de esta evaluación, también se identifican algunas consideraciones prácticas para desarrolladores de software libre, de modo que se mejore el éxito evolutivo de sus proyectos, y se facilite la transición entre las fases de catedral y bazar.

3.1. Fase catedral

Una de las principales diferencias entre el software cerrado (tradicional) y el software libre es la propiedad del código. En entornos tradicionales, un grupo de individuos conduce el desarrollo, mientras que los usuarios ni contribuyen ni tienen acceso al código. En el software libre, potencialmente cualquiera tiene el derecho a acceder y modificar el código fuente de una aplicación. Creemos que un sistema libre típico presenta una fase de catedral en la primera parte de su historia evolutiva.

Sistema Arla – Entrada: La **figura 2** (izquierda) muestra la distribución de desarrolladores distintos por mes para el

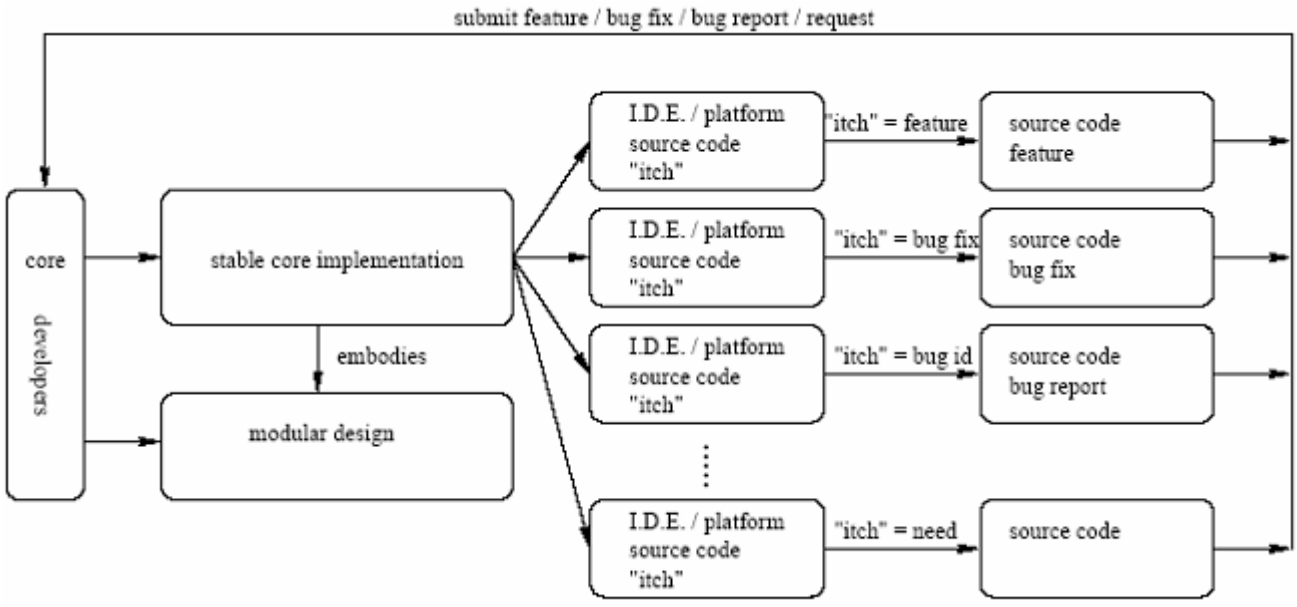

Figura 3. Fase de bazar (detalle).

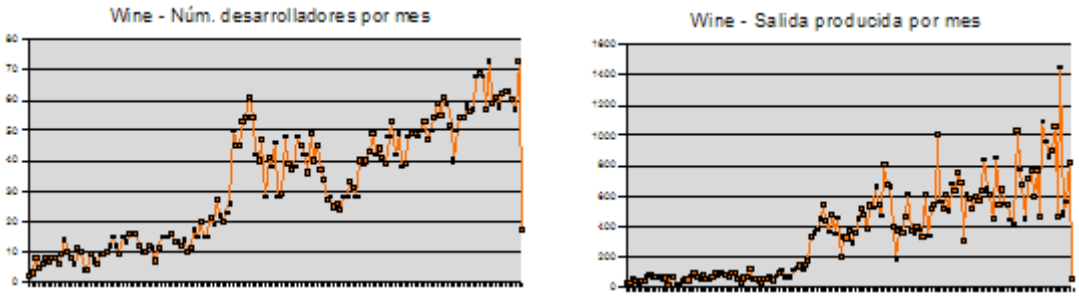

Figura 4. Número de desarrolladores (izquierda) y trabajo producido para el caso de Wine.

sistema Arla. A pesar de que más de 80 desarrolladores han contribuido con código, parches y arreglo de fallos al proyecto (véase **tabla 1**), el número de desarrolladores distintos por mes es mucho más bajo. De media, sólo 5 desarrolladores distintos trabajan cada mes en el proyecto. Como se dice en un párrafo anterior, los resultados empíricos no confirman la primera cuestión a investigar; la evolución de desarrolladores activos y distintos muestra un patrón regular y constante.

Sistema Arla – Salida: La **figura 2** (derecha), al contrario, muestra el número de módulos y subsistemas distintos en los que los desarrolladores de Arla han trabajado cada mes desde el comienzo del proyecto. La distribución es muy regular, lo que podría implicar que cuando nuevos desarrolladores se unen al proyecto no se expanden por áreas nuevas, sino que trabajan en funcionalidad ya existente, junto a los desarrolladores principales. Esto se comprobará en la sección dedicada a la fase de transición. Estos resultados, es decir, un patrón de la salida producida constante y que no crece, confirman que la primera cuestión a investigar no se verifica en el caso de Arla.

Aunque estos resultados no implican necesariamente que Arla sea un fracaso si lo comparamos con Wine (como se podría pensar al mirar el número total de desarrolladores en la **tabla 1**), sí que plantean algunas preguntas interesantes: por ejemplo, debería estudiarse por qué sólo un pequeño y constante grupo de desarrolladores está contribuyendo mediante código. Una explicación posible de este (reducido) éxito al reclutar a nuevos desarrolladores puede ser que los potenciales desarrolladores perciben el sistema como maduro [8], y por tanto se necesita poco trabajo en el proyecto. Se han encontrado problemas similares en el pasado en OpenOffice.org y Mozilla: estos sistemas representan dos aplicaciones extremadamente complejas y requerían una enorme inversión en su estudio antes de que los desarrolladores pudieran empezar a contribuir.

En las siguientes secciones se evalúan consejos prácticos sobre cómo un proyecto de software libre puede afrontar problemas como los que se están encontrando en Arla, y beneficiarse de los esfuerzos de un grupo de desarrolladores mayor.

3.2. Fase bazar

El objetivo de muchos proyectos de software libre es alcanzar una etapa en la que una comunidad de usuarios pueda contribuir de manera activa al desarrollo posterior del proyecto. Algunas de las características clave de la fase de bazar se muestran en la **figura 3** y pueden resumirse así:

- Contribuciones: el estilo bazar hace que el código fuente esté disponible públicamente, y las contribuciones se fomentan de manera activa, sobre todo de personas que sean usuarios del software. Las contribuciones pueden venir de muchas maneras diferentes y en momentos diferentes. Los usuarios sin perfil técnico pueden sugerir nuevos requisitos, escribir documentación y tutoriales, o poner de manifiesto problemas de usabilidad (que se representan como aportaciones de bajo nivel en la **figura 3**)
- Calidad del software: Las inspecciones exhaustivas y en paralelo del código proporcionan unos mayores niveles de calidad. Estas inspecciones las realiza una comunidad grande de usuarios y desarrolladores. Estos beneficios son consistentes con los principios de la Ingeniería del Software: el proceso de depuración de un proyecto de software libre es sinónimo de la fase de mantenimiento del ciclo de vida tradicional de un proyecto de software.
- Comunidad: una red de usuarios y desarrolladores revisa y modifica el código asociado con un sistema de software. El viejo dicho *"el trabajo compartido es más llevadero"*[5] describe las razones por las que algunos proyectos de software libre tienen éxito [27].

Sistema Wine – Entrada: La **figura 4** (izquierda) muestra la distribución de desarrolladores distintos por mes para el sistema Wine. En total, más de 800 desarrolladores han contribuido con código, parches y arreglando fallos (véase la **tabla 1**). Aunque el proyecto tiene un período de vida más largo, que podría haber facilitado el crecimiento del número de desarrolladores, mediante el número de desarrolladores se puede identificar una clara división entre la primera fase (catedral) y la última (bazar). Alrededor de julio de 1998, el sistema Wine experimentó una evolución masiva en el número de desarrolladores distintos involucrados en el proyecto. La sostenibilidad de esta nueva fase de bazar se demuestra por el incremento continuo de nuevos desarrolladores en el proyecto. Wine proporciona las evidencias empíricas para responder a la primera cuestión a investigar, esto es, un patrón creciente de desarrolladores activos señala la presencia de la fase de bazar. La sostenibilidad de la fase de bazar es visible en la cantidad de desarrolladores activos participando en la evolución del sistema, que cambia cada mes.

Sistema Wine – Salida: La fase de bazar se caracteriza por un proceso abierto en el que la entradas proporcionadas por voluntarios definen la dirección del proyecto, incluyendo la lista de requisitos. La implementación inicial se basa principalmente en los requisitos del autor del proyecto. En la fase de bazar, el proyecto se beneficia de la participación de un amplio espectro de usuarios (con diferentes requisitos), que trabajan juntos para incrementar la funcionalidad y el atractivo del software.

En el proyecto Wine se logra de una manera satisfactoria este proceso de desarrollo en paralelo. Durante la investigación de este sistema, se puso de manifiesto la evolución del alcance del proyecto, a través de la cantidad de módulos distintos en los que los desarrolladores trabajaban cada mes. En la **figura 4** (derecha) se muestra la cantidad de módulos y subsistemas distintos en los que los desarrolladores han trabajado desde el comienzo del proyecto: la distribución crece de manera brusca justo cuando se observa un crecimiento en el número de autores distintos. Esto significa que el proyecto se está expandiendo hacia nuevas áreas gracias a los nuevos desarrolladores que se unen de manera constante. El patrón de crecimiento de desarrolladores activos sostiene el crecimiento de la salida producida: como en el párrafo

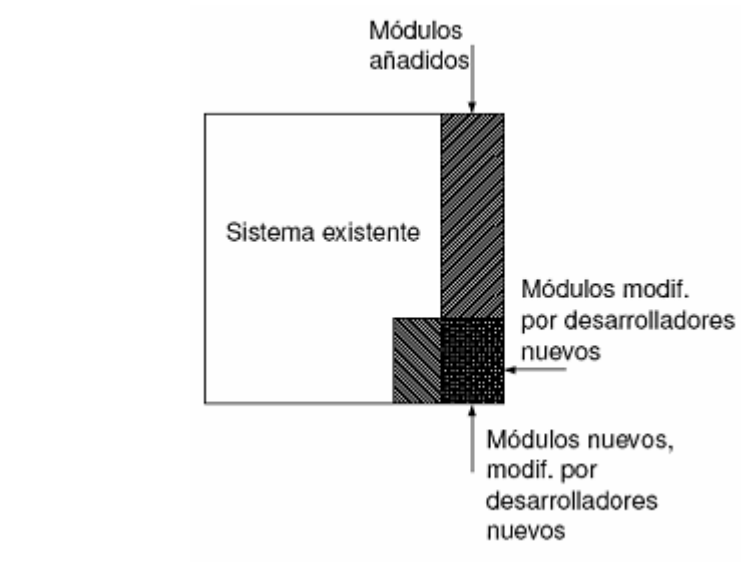

Figura 5. Diseño de la segunda cuestión a investigar.

anterior, la primera cuestión a investigar ayuda a señalar la presencia de la fase de bazar cuando ocurre ese patrón de crecimiento.

3.3. Fase de transición: nuevas vías de desarrollo

El marco teórico representado en la **figura 1** asigna un papel fundamental a la fase de transición, dado que requiere de una drástica reestructuración del proyecto, especialmente en la manera en la que es gestionado. Un aspecto importante es comenzar la fase de transición en el momento correcto. Este paso es crucial y muchos proyectos no logran superar este obstáculo [11]. Dado que durante la fase de transición es cuando hay que atraer a los voluntarios, el prototipo tiene que ser funcional pero a la vez todavía debe necesitar de desarrollo [17][28][2].

Si el prototipo no tiene suficiente estabilidad o funcionalidad, los voluntarios potenciales puede que no se unan al proyecto. Por otro lado, si el prototipo está demasiado avanzado, los nuevos voluntarios no tienen demasiados incentivos para unirse al proyecto porque el código ya existente es complejo o las características que estos desarrolladores requieren han sido implementadas ya. En los dos casos, añadir direcciones futuras de desarrollo al sistema puede proporcionar a los potenciales desarrolladores vías para el desarrollo del proyecto.

Basándonos en la segunda cuestión a investigar, cuando los nuevos desarrolladores se unen a un proyecto, tienden a trabajar en módulos nuevos más que en módulos viejos. Como consecuencia de esto, los desarrolladores principales deberían expandir el sistema original hacia nuevas direcciones para proporcionar nuevo código sobre el que trabajar: esto fomentaría el reclutamiento de desarrolladores nuevos y facilitaría la fase de transición.

Para evaluar esta cuestión, se diseñó un experimento: primero, se extraen los módulos nuevos añadidos cada mes. En paralelo, se extrae la cantidad de desarrolladores nuevos cada mes. Finalmente, las partes en las que los nuevos desarrolladores han trabajado se definen como el porcentaje de módulos nuevos que han sido tocados por estos desarrolladores. La **figura 5** muestra un resumen gráfico de este proceso. Se extrajeron los resultados empíricos para los dos sistemas, Arla y Wine. Se muestran en un diagrama de caja, que se extiende para todas las versiones de los dos sistemas. La **figura 6** describe, en porcentaje, la cantidad de módulos nuevos tocados por los desarrolladores nuevos.

Transición lograda – Wine: este sistema revela que cuando desarrolladores nuevos entran en el proyecto tienden a trabajar con menores dificultades en partes nuevas del código mejor que en partes antiguas. De hecho, más del 50% (en media) del trabajo que realizan los desarrolladores nuevos se produce en módulos añadidos el mismo mes, ya sea por los desarrolladores principales o por ellos mismos (**figura 6**, derecha). Además, la media era mayor cuando se consideraba sólo la fase de bazar en Wine.

El primer resultado se confirma dibujando la cantidad de módulos nuevos creados por los desarrolladores (**figura 7**, derecha). Se detecta un patrón creciente, similar al patrón de evolución global del sistema (**figura 4**): cuando desarrolladores nuevos se unen al proyecto trabajan en las partes más nuevas del sistema, mientras que los desarrolladores principales sostienen la comunidad del proyecto añadiendo continuamente módulos nuevos.

Transición no lograda – Arla: este sistema proporciona un diagrama de caja mucho más interesante. La tendencia de los desarrolladores nuevos es claramente trabajar en algo nuevo mejor que en algo antiguo (**figura 6**, izquierda). La diferencia principal con Wine es que, para la mayoría de los períodos no hay desarrolladores nuevos que se unan al proyecto. Basándonos en las suposiciones de la segunda cuestión a investigar, los desarrolladores nuevos todavía prefieren comenzar partes nuevas, o trabajar en partes que se han añadido recientemente: en cualquier caso, este proyecto no puedo superar la fase de transición al no reclutar desarrolladores nuevos. Por tanto, podemos concluir que los desarrolladores de Arla fallaron al no crear nuevas direcciones para el proyecto mediante la creación de módulos o subsistemas nuevos (**figura 7**, izquierda). Se observa un patrón decreciente, que confirma que los desarrolladores nuevos (y la comunidad alrededor del proyecto), aunque querían participar en el proyecto, no fueron adecuadamente estimulados por los desarrolladores principales.

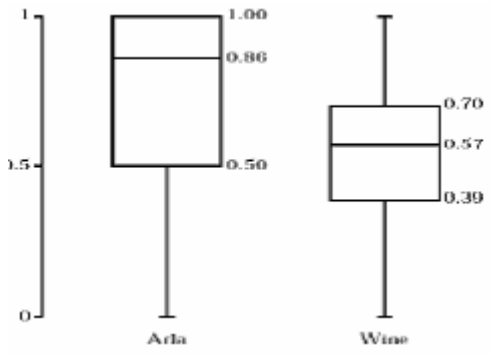

Figura 6. Descripción del esfuerzo dedicado por los desarrolladores nuevos.

En resumen, considerando la segunda cuestión de investigación planteada en los párrafos anteriores, encontramos evidencias similares para los dos proyectos: cuando un desarrollador se une a un proyecto de software libre, tiende a trabajar (añadir, modificar) en módulos nuevos más que en partes previamente existentes. Como conclusión para estos resultados, los desarrolladores principales del proyecto deberían perseguir de manera activa la transición a la fase de bazar: los desarrolladores nuevos tienen que ser estimulados añadiendo nuevas ideas o direcciones al proyecto.

4. Trabajo relacionado

En esta sección relacionamos este trabajo con otros en diferentes campos, específicamente en el estudio empírico de software y en la evaluación de esfuerzos. Dado que este trabajo se enmarca también en el campo más amplio del estudio de la evolución de proyectos de software libre, también son relevantes para este trabajo los estudios empíricos sobre el software libre.

Los estudios más tempranos sobre la evolución de software se realizaron sobre el sistema propietario OS/360 [4]. El estudio inicial se realizó sobre 20 versiones del OS/360, y los resultados de estas investigaciones y otras posteriores sobre software propietario, incluyen la clasificación SPE de programas y un conjunto de leyes de evolución de software [20].

Este trabajo se ha realizado de manera similar, pero evaluando tanto la entrada (esfuerzo) proporcionada al proyecto, como la salida (cambios en el código) lograda. Las cuestiones de investigación planteadas en este artículo derivan de [29], y se basan en la presencia de dos fases diferentes en el ciclo de vida de los proyectos de software libre, denominadas fases de catedral y de bazar [28]. Este hecho contrasta con la afirmación de Raymond de que la fase de bazar es típica de proyectos de software libre [15][28]: se realizó una evaluación empírica estudiando dos proyectos de software de tamaño grande, de los cuales sólo uno había realizado la transición a la fase de bazar y atraído a una comunidad grande de desarrolladores. Los autores piensan que tradicionalmente se ha puesto demasiado énfasis en proyectos de éxito que no necesariamente representan a una comunidad de software libre en su totalidad [13][15][16][26]. Pocos proyectos logran hacer la transición a la fase de bazar, atrayendo a una comunidad grande de desarrolladores activos en el proceso.

Tener un bazar grande alrededor del proyecto tiene muchas ventajas, como la capacidad de incorporar información de retorno de una base de usuarios y desarrolladores muy diversa. En cualquier caso, esto no implica que los proyectos que no han alcanzado la fase de bazar sean necesariamente fracasos: no quiere decir que no hayan tenido éxito o que sean de baja calidad. Es interesante señalar que al contrario de lo que afirma Raymond, existen algunas aplicaciones, como *GNU coreutils* y *tar*, que siguen claramente un modelo de catedral y forman parte de todos los sistemas Linux. De manera similar, existen muchos proyectos desarrollados por una única persona, con competencias excelentes, que muestran altos niveles de calidad. Debido a la falta de mejores teorías y de investigación empírica, la calidad de un proyecto de software se supone que se produce debido al proceso de revisión por pares en el bazar [1][26][28]. Sin embargo, no todos los proyectos de alta calidad presentan un bazar grande o un proceso de revisión por pares.

Un proyecto en la fase de catedral puede ser muy exitoso y tener mucha calidad [31]. Sin embargo, existen algunas restricciones que un proyecto en la fase de catedral debe afrontar, además de los problemas potenciales que serían menos graves si el proyecto tuviera una comunidad grande. Por ejemplo, aunque es posible para un único desarrollador escribir una aplicación de alcance limitado (como un cargador de arranque), sólo una comunidad puede completar el proyecto para llevarlo a entornos más amplios (como un entorno de escritorio). Además, un proyecto escrito por un único desarrollador puede ser de alta calidad pero también asume el riesgo de fracaso debido a que se sustenta en una única persona que trabaja como voluntario [23][25]. Tener una comunidad alrededor del proyecto lo hace más sostenible.

Estos argumentos muestran la falta de investigación en algunas áreas relacionadas con proyectos de software libre. Aunque se han asumido en el pasado algunos modelos para todos los proyectos de software libre, parece que está cada vez más claro que existe mucha variedad en los procesos de desarrollo [9][19][14]. Se necesitan mejores teorías acerca del éxito y la calidad de proyectos de software libre [24], además de comparaciones entre proyectos con diferentes grados de éxito y calidad. Finalmente, no debemos asumir que la fase de bazar es necesariamente la óptima para todos los proyectos de software libre, o que no está asociada con algunos problemas. Se acepta de manera general que es mejor que un proyecto de software libre sea abierto, pero si el proyecto es demasiado abierto puede estar demasiado expuesto a desarrolladores incompetentes o personas que desaniman a los que más contribuyen.

5. Conclusiones y trabajo futuro

Hasta este momento se han estudiado proyectos de software libre que tienen éxito, pero sin proporcionar pruebas empíricas de cómo han logrado este éxito. Para cubrir esta carencia, este artículo presenta el estudio empírico de dos proyectos de software libre, Arla y Wine, para ilustrar las diferentes fases en su ciclo de desarrollo y las comunidades que se forman alrededor de ellos. Se analizaron los ficheros de *ChangeLog*, de modo que se grabaron todos los cambios y nuevo código realizados por los desarrolladores durante varios años.

La principal hipótesis de este artículo es que las fases de catedral y bazar, tal y como las propuso y describió inicialmente Raymond [28], no son mutuamente excluyentes: los

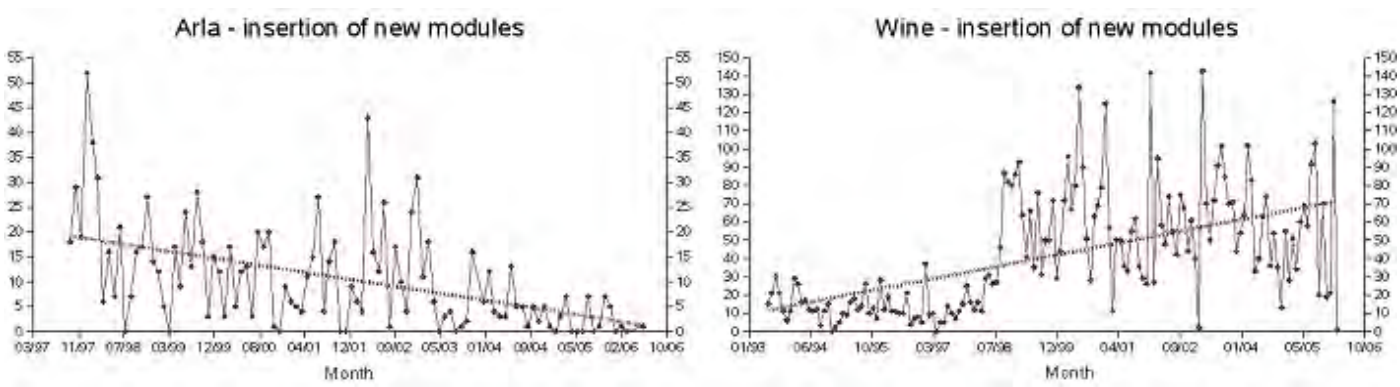

Figura 7. Creación de módulos nuevos en Arla y Wine.

proyectos de software libre comienzan en una fase de catedral, y de manera potencial migran a una fase de bazar. La fase de catedral se caracteriza por un desarrollo cerrado realizado por un grupo pequeño o un único desarrollador. La fase de bazar explota las ventajas de tener un número grande de voluntarios que contribuyen al proceso de desarrollo informando de defectos, solicitando nuevas características, arreglando fallos o proporcionando nuevas funcionalidades. La transición entre las dos fases es también una fase en sí misma, que necesita ser acondicionada mediante las acciones específicas del grupo de desarrolladores principales o del autor del proyecto. Esta fase de transición es fundamental para lograr un verdadero éxito y que el proyecto resulte popular.

Se propuso una cuestión a investigar para estudiar las diferencias entre las fases de catedral y bazar: el primer sistema (Arla) ha permanecido durante todo su ciclo de vida en una fase de catedral, debido a que sólo aportaba esfuerzo un limitado número de desarrolladores. Esto no debe entenderse como un signo de fracaso de un proyecto de software libre, sino como una oportunidad potencialmente perdida para establecer una comunidad próspera alrededor del proyecto. Por el contrario, el segundo sistema (Wine) sólo muestra una fase inicial similar a la de Arla: una segunda fase, más larga, presenta un número de desarrolladores activos creciente y una expansión continua del sistema.

Mediante la segunda cuestión a investigar, se centró el estudio en las preferencias de los desarrolladores nuevos que se unen al proyecto: los resultados de los dos proyectos muestran que los desarrolladores nuevos prefieren trabajar en módulos añadidos recientemente más que en módulos existentes previamente. En el caso del sistema Wine, los desarrolladores principales facilitaron la transición de fase añadiendo nuevos módulos en los que los desarrolladores pudieran trabajar. Por el contrario, los desarrolladores nuevos en el caso de Arla, aunque tuvieran ganas de trabajar en código nuevo, no encontraron nuevas direcciones en el proyecto, de modo que no se logró atraer a un número suficiente de nuevos desarrolladores.

Proponemos como trabajo futuro la replicación de este estudio en otros proyectos de software libre, especialmente en aquéllos que pertenezcan al mismo dominio de aplicación: los resultados tal y como se han obtenido en este estudio han analizado la comunidad desde un punto de vista neutral, esto es, sin considerar factores exógenos. El próximo paso será introducir estos factores en el estudio, y analizar proyectos grandes que están compitiendo en este momento por un recurso escaso, los desarrolladores.

Referencias

[1] A. Aoki, K. Hayashi, K. Kishida, K. Nakakoji, Y. Nishinaka, B. Reeves, A. Takashima, Y. Yamamoto. A case study of the evolution of jun: an object-oriented open-source 3d multimedia library. En actas de la *23rd International Conference on Software Engineering*, pp. 524-533, Toronto, Canada, 2001.

[2] B. Arief, C. Gacek, T. Lawrie. Software architectures and open source software – where can research leverage the most? En actas del *1 st Workshop on Open Source Software Engineering*, Toronto, Canada, 2001.

[3] R. Austen, G. Stephen. Evaluating the quality and quantity of data on open source software projects. En actas de la1st International Conference on Open Source Systems, Génova, Italia 2005.

[4] L. A. Belady, M. M. Lehman. A model of large program development. *IBM Systems Journal, 15(3)*:225-252, 1976.

[5] M. Bergquist, J. Ljungberg. The power of gifts: Organising social relationships in open source communities. *Information Systems Journal, 11(4)*:305-320, 2001.

[6] A. Capiluppi. Models for the evolution of OS projects. En actas de la *International Conference on Software Maintenance*, pp. 65-74, Amsterdam, Países Bajos, 2003.

[7] A. Capiluppi, P. Lago, M. Morisio. Evidences in the evolution of OS projects through changelog analyses. En actas del *3rd Workshop on Open Source Software Engineering*, Portland, OR, EEUU, 2003.

[8] A. Capiluppi, M. Morisio, J. F. Ramil. Structural evolution of an open source system: A case study. En actas del *12th International Workshop on Program Comprehension* (IWPC), pp. 172-182, Bari, Italia, 2004.

[9] K. Crowston, J. Howison. The social structure of free and open source software development. *First Monday, 10(2)*, 2005.

[10] M. Fischer, M. Pinzger, H. Gall. Populating a release history database from version control and bug tracking systems. En actas de la *International Conference on Software Maintenance*, pp. 23-32, Amsterdam, Países Bajos, 2003.

[11] K. F. Fogel. *Open Source Development with CVS*. The Coriolis Group, Scottsdale, Arizona, 1ª edición, 1999. ISBN 1-57610-490-7.

[12] D. M. German. An empirical study of fine-grained software modifications. pp. 316-325, Chicago, IL, EEUU, 2004.

[13] D. M. German. Using software trails to reconstruct the evolution of software. *Journal of Software Maintenance and Evolution:Research and Practice, 16(6)*:367-384, 2004.

[14] D. M. German, A. Mockus. Automating the measurement of open source projects. En actas del *3rd Workshop on Open Source Software Engineering*, Portland, OR, EEUU, 2003.

[15] M. W. Godfrey, Q. Tu. Evolution in open source software: A case study. En actas de la *International Conference on Software Maintenance*, pp. 131-142, San Jose, CA, EEUU, 2000.

[16] J. Howison, K. Crowston. The perils and pitfalls of mining SourceForge. En actas del *International Workshop on Mining Software Repositories* (MSR 2004), pp. 7-11, Edimburgo, UK, 2004.

[17] K. Johnson. *A descriptive process model for open-source software development*. Tesis de máster, Department of Computer Science, Universidad de Calgary, 2001. <http://sern.ucalgary.ca/students/theses/KimJohnson/thesis.htm>.

[18] N. Jorgensen. Putting it all in the trunk: Incremental software engineering in the FreeBSD open source project. *Information Systems Journal, 11(4)*:321-336, 2001.

[19] S. Koch, G. Schneider. Effort, cooperation and coordination in an open source software project: GNOME. *Information Systems Journal, 12(1)*:27-42, 2002.

[20] M. M. Lehman, L. A. Belady, editors. *Program evolution: Processes of software change*. Academic Press Professional, Inc., San Diego, CA, EEUU, 1985. ISBN:0-12-442440-6.

[21] L. Lopez, J. G. Barahona, I. Herraiz, G. Robles. Applying social network analysis techniques to community-driven libre software projects. *International Journal of Information Technology and Web Engineering, 11(4)*:321-336, 2006.

[22] T. Mens, J. F. Ramil, M. W. Godfrey. Analyzing the evolution of large-scale software: Guest editorial. *Journal of Software Maintenance and Evolution, 16(6)*:363-365, 2004.

[23] M. Michlmayr. Managing volunteer activity in free software projects. En actas del *2004 USENIX Annual Technical Conference, FREENIX Track*, pp. 93-102, Boston, EEUU, 2004.

[24] M. Michlmayr. Software process maturity and the success of free software projects. En K. Zielinski and T. Szmuc (editores), *Software Engineering: Evolution and Emerging Technologies*, páginas 3-14, Cracovia, Polonia, 2005. IOS Press. ISBN: 978-1-58603-559-4.

[25] M. Michlmayr, B. M. Hill. Quality and the reliance on individuals in free software projects. En actas del *3rd Workshop on Open Source Software Engineering*, pp. 105-109, Portland, OR, EEUU, 2003.

[26] M. Michlmayr, F. Hunt, D. Probert. Quality practices and problems in free software projects. En M. Scotto and G. Succi (editores), *Proceedings of the First International Conference on Open Source Systems*, pp. 24-28, Génova, Italia, 2005.

[27] A. Mockus, R. T. Fielding, J. D. Herbsleb. Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology, 11(3)*:309-346, 2002.

[28] E. S. Raymond. *The Cathedral and the Bazaar*. O'Reilly & Associates, Sebastopol, CA, EEUU, 1999. ISBN: 1565927249.

[29] A. Senyard, M. Michlmayr. How to have a successful free software project. En actas de la *11th Asia-Pacific Software Engineering Conference*, pp. 84-91, Busan, Corea del Sur, 2004. IEEE Computer Society.

[30] N. Smith, A. Capiluppi, J. F. Ramil. Agent-based simulation of open source evolution. *Software Process: Improvement and Practice, 11(4)*:423-434, 2006.

[31] I. Stamelos, L. Angelis, A. Oikonomou, G. L. Bleris. Code quality analysis in open-source software development. *Information Systems Journal, 12(1)*:43-60, 2002.

[32] L. Torvalds. The Linux edge. In C. DiBona, S. Ockman, and M. Stone, editors, Open Sources: *Voices from the Open Source Revolution*, pp. 101-111. O'Reilly & Associates, Sebastapol, CA, EEUU, 1999. ISBN: 1-56592-582-3.

Notas

[2] N. T.: En este fichero se escribe una entrada describiendo los cambios que se realizan en cada nueva versión del software.

[3] N.T.: *distinct* se usa en SQL para seleccionar registros que son distintos en su identificador.

[4] Las líneas de código producidas por un desarrollador están sesgadas por sus habilidades, por el lenguaje de programación y, en general, por el contexto de las modificaciones.

[5] N.T: *Many hands make light work* en el texto original.