Modelo de desarrollo en proyectos de software libre y de código abierto [FOSS]: una mirada desde la teoría de la cooperación

Development model for Free and Open Source Software projects [FOSS]: a perspective from the theory of cooperation

COLCIENCIAS TIPO 3. ARTÍCULO DE REVISIÓN
RECIBIDO: MARZO 1, 2014; ACEPTADO: JUNIO 11, 2014

Byron Cuesta Quintero, M.Sc
bcuesta@unab.edu.co
Jorge Andrick Parra Valencia, Ph.D
japarra@unab.edu.co

Universidad Nacional Autónoma de Bucaramanga, Colombia
Grupo de Investigación en Pensamiento Sistémico
Facultad de Ingeniería de Sistemas
Programa de Maestría en Software Libre

Resumen

Este artículo reconoce el proceso de desarrollo de proyectos de software libre y de código abierto (FOSS) a partir de entender e interpretar las prácticas y actividades que son resultado de un proceso de software. Para este estudio se analizaron y reconocieron teorías de la cooperación en general, cooperación en el software libre y el modelo de desarrollo de algunos proyectos de éxito mundial como el núcleo del sistema operativo GNU/Linux, el entorno de escritorio Gnome, el servidor web Apache y la organización GNU enterprise [GNUe]. La revisión de la literatura permitió identificar como, a partir del modelo de cooperación en red que se utiliza en el desarrollo de software libre, se crean alianzas de desarrolladores y comunidades virtuales, que proporcionaron una visión para comparar las etapas de la metodología tradicional de desarrollo de software (modelo en cascada) y del software libre. Los resultados y las conclusiones presentan un conjunto de elementos que permiten comprender el proceso de desarrollo del software libre.

Palabras Clave
Software libre; cooperación; modelo de desarrollo de software; comunidad virtual; *kernel* de Linux.

Abstract

This article recognizes the process of developing projects of free and open source software (FOSS) from understanding and interpreting the practices and activities that are the result of a software process. For this study we analyzed and recognized theories of cooperation in general, cooperation in free software and model development of some projects worldwide success as the core of the GNU / Linux operating system, the GNOME desktop environment, the Apache web server and organization GNU Enterprise (GNUe). The literature review identified as possible from the model of cooperation network is used in the development of free software developers alliances and virtual communities provided a view to compare the stages of the traditional software development methodology created (Waterfall model) and free software. The results and conclusions presented a set of elements that allow us to understand the process of development of free software.

Keywords
Free software; cooperation; software development model; virtual community; Linux *kernel*.

I. INTRODUCCIÓN

Los proyectos de software libre se basan en la comunicación mediada por computador, utilizando como plataforma servicios de Internet que permiten crear comunidades virtuales de desarrollo de software, con participantes dispersos geográficamente en el mundo y con intereses y creencias comunes que permiten crear escenarios para contribuir al desarrollo y el mantenimiento de recursos comunes, a través del trabajo en equipo y el uso de la cooperación como mecanismo primordial para resolver conflictos (González, Seoane, & Robles, 2003; Elliott & Scacchi, 2004).

El software libre se construye mediante el aporte de un gran número de voluntarios que donan parte de su tiempo, a menudo de manera irregular, a este esfuerzo de cooperación (Mockus, 2000). Hoy en día, los resultados son equivalentes o mejores a los obtenidos en el software desarrollado de forma comercial; ha sido tal el éxito que algunos proyectos son financiados por empresas y algunos desarrolladores reciben pago. La disponibilidad del código fuente de forma libre hace que el modelo de desarrollo no tenga de forma arraigada características propias de los proyectos tradicionales –como un calendario asociado a una lista de entregables o un diseño a nivel de sistema explícito o reuniones de planificación de actividades cara a cara–, ya que los desarrolladores trabajan dispersos globalmente coordinando sus actividades a través de listas de correo (Mockus, Fielding, & Herbsleb, 2002). El éxito de los proyectos de software libre radica en que el código está escrito con más cuidado y mayor creatividad, debido a la pasión por programar de sus desarrolladores (Raymond, 1999).

Según Schweik y English (2007) los sitios de alojamiento de software libre como Sourceforge.net proporcionan información sobre cuándo se inició un proyecto y cuándo se lanzaron la primera versión y los *release* del software. De esta manera, permiten el seguimiento de los proyectos demostrando que el éxito o fracaso de los proyectos de software libre se asocia a tres atributos: la etapa evolutiva del proyecto, que corresponde a las etapas de iniciación –periodo de desarrollo, donde aún no se ha publicado el código– y crecimiento –periodo posterior a la primera versión publicada–;el tamaño del equipo desarrollador, donde un número de participantes pequeño permite generar escenarios de reciprocidad y confianza para la cooperación y facilita la comunicación (Koch & Schneider, 2002); y las medidas de éxito o fracaso (Stewart & Ammeter, 2002; Crowston, Annabi, & Howiston, 2003). De esta manera, si un proyecto no ha sido capaz de producir un primer lanzamiento en el primer año del proyecto, se considera como un fracaso en la iniciación y, por lo general, queda abandonado; en cambio, si un proyecto, en la etapa de crecimiento, ha producido varias liberaciones de un producto de software que realiza una tarea informática útil, se considera exitoso.

II. ANTECEDENTES

A. Concepto de libertad en el software libre

Para Richard Stallman (1998) y la Free Software Fundation [FSF] (2014), el software libre es un asunto de libertad de los usuarios para ejecutar, copiar, distribuir, estudiar, modificar y mejorar el software (Gay, 2002). Para garantizar estas libertades, las distribuciones se realizan utilizando la licencia GNU *General Public License* [GPL], la cual permite al receptor del software ejercer esas libertades y definir restricciones, como por ejemplo dar crédito a los autores originales en caso de redistribución (González et al., 2003). Existen textos que definen las condiciones que debe cumplir una licencia para ser considerada de software libre, representadas en definiciones y directrices sobre software libre y programas de fuente abierta (SPI, 2014; OSI, s.f.; FSF, 2014).

La FSF promueve los principios del software libre con el objetivo de eliminar por completo la necesidad de utilizar programas con licencias de tipo propietarias. A partir de las creencias de la FSF ha surgido el movimiento de software libre, el cual se basa en el concepto de que el acceso al código fuente libre es fundamental para el fomento y la innovación futura en las ciencias de la computación (DiBona, Ockman, & Stone, 1999). De esta manera el software libre representa una ventaja social –al permitir a los usuarios cooperar–, y una ventaja ética –al respetar la libertad del usuario (Dempsey, Weiss, Jones & Greenberg, 1999).

Otro término relacionado con el software libre es el de programas de fuente abierta [*Open Source Software*), promovido por la *Open Source Initiative*. El término es distinto, ya que hace énfasis en la disponibilidad de código fuente, no en la libertad. El modelo de desarrollo del software *open source* tiene auge en el mundo de software comercial (González et al., 2003). El software comercial es distribuido únicamente como un binario, en forma ejecutable, es decir es un software de código cerrado,

donde sus desarrolladores reservan para sí, las habilidades de conocer el código fuente, modificarlo, distribuirlo y autorizar a otros hacer alguna de estas cosas. El software libre es desarrollado en un ambiente de colaboración en el cual la mayoría de los contribuyentes son voluntarios (Arulkumar & Chandra, 2012).

B. Equipos virtuales que forman comunidad

Los desarrolladores de software libre encuentran motivación en aprender y desarrollar nuevas habilidades, siendo intrínseca su pasión por programar, lo que hace que disfruten su trabajo (Hertel, Neidner & Hermann, 2003; Himanen, 2002; Ghosh, Glott, Krieger, & Robles, 2002). En las comunidades de software libre el desarrollador busca ser reconocido como colaborador de confianza impulsado por la necesidad de aportar algo al grupo y obtener reputación por ello (Feller & Fitzgerald, 2001; Stewart & Gosain, 2001; González et al., 2003). La reputación es el engranaje del sistema; va de lo individual a lo compartido, funciona de manera dual y motiva a los programadores a contribuir y, si tienen relevancia en el proyecto, a apropiarse de nuevas responsabilidades, definiendo patrones de colaboración e interacción (Cox, 1998; Robles, Scheider, Tretkowski, & Weber, 2001; Ghosh, Glott, Krieger, & Robles, 2002). El nivel de cooperación de un conjunto de individuos en una comunidad con bienes comunes aumenta a partir de la reciprocidad, creando identidades y normas de confianza para el grupo, que permiten mejorar la comunicación y resolver conflictos para asegurar acciones colectivas exitosas (Ostrom, 1997).

Los equipos de desarrollo de software libre pueden estar formados por cientos de desarrolladores que residen en diferentes continentes, y aun así cooperan para tener éxito utilizando canales de comunicación mediada por computador, sin la necesidad de una autoridad central o una estructura de gestión formal (Giaglis & Spinellis, 2012). En un ambiente de cooperación, los individuos de una comunidad se enfrentan a dilemas sociales, conflictos de racionalidad individual y colectiva, donde las acciones de una persona afectan a las demás (Kollock, 1998; Parra, 2010). Estos dilemas pueden lograr que no haya cooperación; surgen cuando los individuos, llamados *free-riders* quieren usar un bien común sin contribuir a su desarrollo o mantenimiento. Si todos los individuos optan por ser *free-riders* no se producirá un beneficio colectivo (Ostrom, 2001).

A medida que los desarrolladores participan y hacen su contribución a un proyecto de software libre, encuentran personas con valores y creencias en común, lo que permite crear redes profesionales de desarrolladores agrupados en comunidades que se comunican a través de sitios Web de software libre, en donde se tiene acceso a artefactos, discusiones de listas de correo o IRC (*kernel cousins*), formando equipos de trabajo para socializar, construir relaciones de confianza, compartir y aprender con otros. De esta forma los proyectos de software libre persisten sin una autoridad central en una organización virtual (Scacchi, 2002; Crowston & Scozzi, 2002; Smith & Kollock, 1999).

En los proyectos de software libre, al compartir el código fuente se genera interdependencia entre la red de desarrolladores de software sobre el uso del mismo código fuente, los artefactos, las herramientas de desarrollo y los sitios Web compartidos. Esta interdependencia crea un ecosistema de software y hace que el código fuente de un proyecto crezca de forma exponencial (Scacchi, 2002). Este resultado se conoce como *externalidad de red* (Ostrom, 1990); para este caso, aplica en la forma en que los aportes individuales suman al código fuente compartido generando externalidades positivas para el resto del grupo. Los dilemas de acción colectiva surgen en el caso de recursos de acceso libre, donde existe el riesgo de que no se generen suficientes incentivos para una acción colectiva de conservación del recurso común, que pueden repercutir en acciones individuales que crean externalidades negativas al resto del grupo (Ostrom, 1990; 2000).

En los proyectos de software libre y de código abierto [FOSS] exitosos, como el núcleo del sistema operativo GNU/Linux, el entorno de escritorio *Gnome*, el servidor web *Apache* y la organización *GNU Enterprise* [GNUe], se puede reconocer que los resultados a nivel de grupo no se ven afectados por elementos como que sus desarrolladores estén dispersos globalmente y que sus culturas sean diferentes (Mockus, 2000; Koch & Schneider, 2002; Germán, 2002; González et al., 2003; Elliott & Scacchi, 2004). Un equipo virtual es *un grupo de personas que trabajan de forma independiente con un propósito compartido a través del espacio, el tiempo y con límites organizacionales que utilizan la tecnología* (Lipnack & Stamps, 2000). Los proyectos de software libre se caracterizan por no seguir un modelo de organización típica, la falta de adopción de un modelo clásico de desarrollo de software, y la existencia de una cultura abierta, asociada a las creencias de las raíces del software libre. De esta manera, el desarrollo de software

libre se realiza por personas que forman equipos virtuales que usan tecnología para comunicarse y apoyar el trabajo distribuido en un ambiente de cooperación en red (Noll & Scacchi, 1999; Giaglis & Spinellis, 2012).

C. Procesos de desarrollo en proyectos de software libre

Para el desarrollo de cualquier producto de software se realizan una serie de tareas entre la idea inicial con la especificación del sistema y el producto final, que incluye el mantenimiento de éste después de que se utiliza. Un modelo de desarrollo de software representa un conjunto de actividades y resultados de un proceso de software (Sommerville, 2005).

El modelo de desarrollo en proyectos de software libre generalmente no adopta una metodología tradicional de ingeniería de software, ya que suele ser más informal, debido a que los desarrolladores realizan esas tareas voluntariamente y sin recompensa económica (Robles, 2002). Aun así, logran desarrollar software con un alto nivel funcional y operativo, con una distribución global de fácil uso para la comunidad (Scacchi, 2004; González et al., 2003).

En *La catedral y el bazar* Raymond (1999) describe algunas características del modelo de desarrollo de software libre, tomando como caso de estudio el *kernel* de Linux y hace énfasis en lo que diferencia este modelo de los de desarrollo propietario. Se explica que el núcleo Linux se desarrolló siguiendo un esquema similar al de un bazar oriental, en el cual no existe una máxima autoridad que controla los procesos que se están desarrollando ni planifica estrictamente lo que ha de suceder. Por otro lado, los roles de los participantes pueden cambiar de manera continua, de ahí el hecho de publicar versiones de prueba de forma temprana y frecuente. También muestra que el éxito de Linux dentro del mundo del software libre, es una sucesión de buenas maneras para aprovechar al máximo las posibilidades que ofrece la disponibilidad de código fuente y la interactividad, mediante el uso de sistemas y herramientas telemáticas (Robles, 2002).

En el proceso de desarrollo de proyectos de software libre, el análisis y la especificación de requerimientos no se lleva a cabo como una tarea necesaria que produce unos requisitos obligatorios a entregar, sino de forma implícita en las actividades de desarrollo del software. Al realizar la liberación del software, el proyecto es accesible a nivel mundial para participantes nuevos y antiguos, y a partir de ahí se generan discusiones utilizando la comunicación mediada por computador por medio de listas de correo, *chats* o acceso a artefactos de software, de manera que los requisitos aparecen como afirmaciones en los hilos de discusión (*kernel cousins*), sobre lo que el software debe o no hacer y acerca de quién va a asumir la responsabilidad para contribuir en el desarrollo de una nueva funcionalidad. (Scacchi, 2002). En los proyectos de software libre no suele ser común un documento que recoja los requisitos, tal como es normal en un modelo clásico de ingeniería del software (González et al., 2003), sino que, por lo general, se encuentra una carencia de diseño detallado; aunque en los proyectos grandes el diseño del sistema suele tener documentación, no se encuentra muy generalizado. Esto dificulta la participación de nuevos desarrolladores y, en algunos casos, hace que la reutilización de código sea escasa (González et al., 2003).

El desarrollo de software libre surge como una necesidad de algún desarrollador, quien lo diseña e implementa y, a medida que logra funcionalidad, lo libera y lo comparte con desarrolladores con las mismas necesidades (Narduzzo & Rossi, 2003; Ehrenkrantz, 2003); de esta manera, durante la implementación, los mismos desarrolladores siguen una programación a prueba y error hasta que se consiguen los resultados deseados desde el punto de vista subjetivo del programador.

Para el mantenimiento del software las comunidades de software libre utilizan herramientas de control de versiones [CVS], para administrar repositorios de software libre en la Web, ellas permiten tener un mecanismo centralizado para coordinar el desarrollo de software libre y así controlar las mejoras que se incorporan a los archivos de software. Cada equipo del proyecto o administrador del repositorio CVS debe decidir y resolver conflictos para saber si la nueva versión aplica para liberarla como una versión estable. De esta manera los cambios están a disposición de la comunidad cada vez que los desarrolladores liberan versiones alfa ó beta (Mockus, 2000; Koch & Schneider, 2002; Elliott & Scacchi, 2004).

D. Caso de estudio: ciclo de vida de un parche del kernel de Linux

Los desarrolladores del *kernel* de Linux usan un proceso de liberación basado en tiempos sin un riguroso plan de entregas; cada nueva versión del núcleo ocurre en promedio cada dos o tres meses e incluye un modelo de desarrollo que integra continuamente cambios importantes soportados en miles de líneas de código (Corbet, 2013). La

fusión de parches para una nueva versión del *kernel* sigue un procedimiento relativamente sencillo. Al principio de cada ciclo de desarrollo se abre lo que se denomina una ventana de fusión, la cual permite combinar parches con nuevas características al *kernel*. La ventana de fusión permanece abierta por un tiempo aproximado de dos semanas, al final del cual *Linus Torvalds* declara la ventana fusión cerrada y libera el primero de los *release candidate* [RC] *kernel*s, una versión beta del núcleo, con el potencial de ser un producto final. Aquí inicia el tiempo requerido para estabilizar la siguiente versión del *kernel* a liberar, lo que incluye la elección de únicamente parches que corrigen problemas significativos, hasta considerar que el *kernel* es lo suficientemente estable y hacer el lanzamiento final. A partir de una versión estable, el mantenimiento se hace a través de un equipo de mantenedores que ocasionalmente pueden lanzar actualizaciones de versiones estables (Corbet, 2013). Generalmente los mantenedores buscan errores por arreglar en la línea principal del *kernel* de Linux, en las listas de regresiones y *bug tracks* –sistemas de seguimiento a errores que permiten publicar fallos de la línea principal del núcleo de Linux–, ejemplo de ello es el sitio web https://bugzilla.kernel.org/.

La comunidad se reúne en su conjunto a través de listas de correo, por lo que es importante que los desarrolladores eviten ruido en las listas con mensajes que aporten poco o nada a la comunidad. Es importante que los desarrolladores conjuguen en reglas o normas para evitar que el nivel de ruido pueda distraer la comunidad de desarrollo en su conjunto (Corbet, 2013). Según Ostrom (1990) las comunidades exitosas tienen un conjunto de reglas que gobiernan cómo utilizar los recursos comunes y definen quién es el responsable de la producción y el mantenimiento de los bienes colectivos. Las reglas son una característica de las comunidades que cooperan, todos se benefician si todos los desarrolladores aprenden las reglas necesarias para interactuar con el grupo que desarrolla la línea principal del *kernel* de Linux a través de listas de correo; de esta forma, los desarrolladores principales están tentados a ignorar las preguntas de desarrolladores novatos, si estos participan en las listas de correo sin antes aprender las normas de participación (Corbet, 2013).

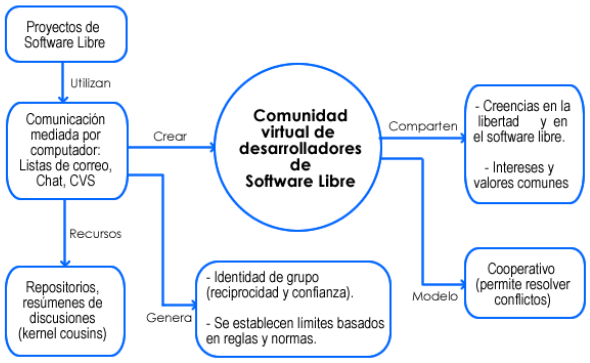
III. RESULTADOS

A. Equipos virtuales que forman comunidad

Las comunidades virtuales de software libre están formadas por personas con ideas afines que comparten los siguientes elementos (ver Figura 1):

- forman una red de desarrolladores con competencias, valores y creencias de libertad en común (DiBona et al., 1999; Dempsey et al., 1999; Elliott & Scacchi, 2004; Stallman, 2002);
- crean identidades de grupo que les permiten cooperar para el desarrollo y mantenimiento de un software de acceso libre y resolver conflictos, apoyados en normas de confianza (Kollock, 1998; Ostrom, 1990; Ostrom, 2000; Koch & Schneider, 2002; Mockus, 2000; Ostrom & Walker, 2003);
- se comunican a través del uso de recursos y servicios de Internet como las listas de correo, *chats*, sitios web que alojan repositorios y herramientas para el control de versiones (Fogel, 1999; Elliott & Scacchi, 2004; Crowston & Scozzi, 2002; Koch & Schneider, 2002; Mockus, 2000; Corbet, 2013); y
- comparten código fuente, lo que permite aprender de otros y beneficiarse mutuamente (Stewart & Gosain, 2001; González et al., 2003.

Figura 1. Comunidad virtual de desarrolladores de software libre



B. Procesos de desarrollo en proyectos de software libre

El modelo de proceso de desarrollo del software libre no está sujeto a las directrices específicas de los modelos tradicionales o genéricos (González et al., 2003), como los enfoques cascada, evolutivo y basado en componentes; aun así, las actividades o etapas son más cercanas a un desarrollo evolutivo y de iteración por procesos, el cual se caracteriza por una implementación inicial que se expone a los comentarios del usuario y a la entrega de versiones incrementales con funcionalidades del sistema (Sommerville, 2005).

Uno de los modelos de desarrollo cercanos al software libre, que está expresada como una metodología ágil

derivada del enfoque incremental por enfocarse en el software en vez de su diseño y documentación, es la programación extrema (Cuesta & Parra, 2013).

La Tabla 1 compara las etapas o actividades de desarrollo de software tradicional (modelo en cascada) y los proyectos de software libre.

Tabla 1. Etapas de software tradicional vs software libre

	Tradicional (Sommerville, 2005).	Software libre
Análisis y especificación de requerimientos	Los requerimientos se definen a partir de consultas con los usuarios. Entonces, se definen en detalle y sirven como una especificación del sistema.	Los requisitos aparecen como afirmaciones en los hilos de discusión sobre lo que el software debe o no hacer y acerca de quién va a asumir la responsabilidad de contribuir en el desarrollo de la nueva funcionalidad (Scacchi, 2002).
Diseño	Diseño detallado con alta documentación. Se establece una arquitectura completa y se identifican y describen las abstracciones fundamentales del sistema.	El software surge como una necesidad de algún desarrollador, quien lo desarrolla e implementa y, a medida que logra funcionalidad, lo libera y comparte (Narduzzo & Rossi, 2003; Ehrenkrantz, 2003).
Implementación y pruebas	Durante esta etapa, el desarrollo del software se lleva a cabo como un conjunto ó unidades de programas. La prueba de unidad implica verificar que cada una cumpla su especificación.	Publicar versiones de prueba de forma temprana y frecuente, de esta manera, durante la implementación, los mismos desarrolladores siguen una programación a *prueba y error* hasta que consiguen los resultados deseados (Raymond, 1999; González et al., 2003).
Mantenimiento	El sistema se instala y se pone en funcionamiento práctico. El mantenimiento implica corregir errores no descubiertos en las etapas anteriores del ciclo de vida, de manera que permita mejorar la implementación de las unidades del sistema.	Para el mantenimiento del software, las comunidades utilizan herramientas de control de versiones [CVS], que permiten tener un mecanismo centralizado para coordinar el desarrollo y controlar las mejoras que se incorporan a los archivos de software (Mockus, 2000; Koch & Schneider, 2002; González et al., 2003; Elliott & Scacchi, 2004; German, 2002).

C. Caso de estudio: ciclo de vida de un parche del kernel de Linux

En el caso de estudio sobre el *kernel* de Linux, los parches no se enlazan directamente a la línea principal, en su lugar, hay un ciclo de desarrollo para asegurar que cada parche sea revisado y tenga la calidad que sugiere un cambio que es deseable tener en la versión estable del *kernel* a liberar (Corbet, 2013). La Tabla 2, muestra las etapas de desarrollo de un parche para la línea principal del *kernel* de Linux.

Tabla 2. Etapas de desarrollo de un parche del *kernel* de Linux (Corbet, 2013)

Etapa	Descripción
Diseño	Especifica los requerimientos reales para el parche y la forma en que se cumplirán los requisitos establecidos. Este trabajo es a menudo realizado sin la participación de la comunidad.
Revisión temprana	Los parches se publican en la lista de distribución esperando comentarios de la comunidad. Para la gestión de código fuente se usa el software de control de versiones GIT y herramientas relacionadas como *Mercurial* o *Quilt* que permiten a los mantenedores rastrear la lista de parches, para encontrar información de autoría, metadatos y parches del repositorio no incluidos en la línea principal del *kernel*.
Revisión más amplia	Cuando el parche se acerca a ser incluido en la línea principal del *kernel*, debe ser aceptado por un mantenedor del subsistema correspondiente, quien lo muestra en el árbol de código del subsistema; este proceso debe conducir a la más amplia revisión del parche y al descubrimiento de cualquier problema resultante de su integración con otros.
Fusión en la línea principal	Con el tiempo, si el parche no tiene quejas de revisión será seleccionado y fusionado en el repositorio de la línea principal gestionado por Linus Torvalds. En este momento también pueden surgir comentarios o problemas.
Última versión estable	El número de usuarios que puedan ser afectados por el parche es ahora grande, así que, una vez más, pueden surgir nuevos problemas.
Mantenimiento a largo plazo	El desarrollador original debe mantener, a largo plazo, la responsabilidad por el mantenimiento del código después de su fusión. Si así no fuere, dejaría una muy mala impresión en la comunidad de desarrollo.

IV. DISCUSIÓN

El software libre usa un modelo de cooperación en red, basado en esfuerzos de participantes que desarrollan el software a distancia, influenciados por elementos de colaboración, para construir el código fuente (Dempsey et al., 1999). La Figura 1, representa una comunidad virtual de software libre.

La creencia colectiva en el software libre y otros elementos comunes asociados a la cultura de los participantes definen la reciprocidad, a partir de escenarios de confianza, para cooperar por la creación y el

mantenimiento del software, representado en un recurso común que debe ser acompañado por una serie de normas que definen el límite para gestionar el bien, en una comunidad virtual donde todos los participantes esperan el máximo beneficio neto positivo (Ostrom, 1990).

La comunicación mediada por computador permite el registro de las discusiones de los desarrolladores, las que, de esta manera, se convierten en un mecanismo de consulta persistente que identifica las creencias culturales de los desarrolladores y las normas de grupo, con lo cual se crean las comunidades virtuales y una cultura organizacional que permite resolver conflictos. Los conflictos aparecen cuando se desarrolla software, y son parte del trabajo de cooperación de los participantes de las comunidades de software libre, donde los proyectos se gestionan de manera informal (Elliott & Scacchi, 2004). Estos conflictos, que están representados en los dilemas sociales, pueden ser generados por participantes que son denominados *free-riders* y ocurren cundo ellos se aprovechan del trabajo de otros y no contribuyen a un esfuerzo conjunto (Kollock & Smith, 1996).

La Tabla 1, muestra una comparación entre las etapas ó actividades de desarrollo de software tradicional (modelo en cascada) y los proyectos de software libre; se evidencia el formato estructurado del modelo tradicional –donde el resultado de una etapa lleva a la otra, de manera secuencial– (Sommerville, 2005), y el del software libre, en el cual no hay una planificación de actividades *bien* detallada y documentada (Raymond, 1999), sino que se asemeja a un desarrollo iterativo donde hay una implementación inicial del software con las funcionalidades del caso, que permite liberarlo a través de algún mecanismo de comunicación como la lista de correo (Narduzzo & Rossi, 2003; Ehrenkrantz, 2003), a partir de donde los requisitos aparecen como afirmaciones en los hilos de discusión sobre lo que el software debe o no hacer (Scacchi, 2002). Como se libera temprano, durante la implementación, los desarrolladores siguen una programación a *prueba y error* hasta que consiguen los resultados deseados (Raymond, 1999; González et al., 2003). Para el mantenimiento del software, cada equipo del proyecto o administrador del repositorio CVS debe tomar decisiones y resolver conflictos, para saber si la nueva versión aplica para ser liberada como una versión estable, a partir de escenarios de confianza (Scacchi, 2004). En el desarrollo de software propietario se hace más compleja la liberación temprana de una parte de la nueva versión, ya que puede afectar el valor en el mercado de la versión completa del software (Michlmayr & Fitzgerald, 2012). En los proyectos de software libre hay equipos de voluntarios que están conectados y distribuidos geográficamente en el mundo trabajando juntos en el desarrollo de versiones de software con millones de líneas de código, las cuales siguen un modelo asociado al desarrollo incremental (González et al., 2003).

La Tabla 2, muestra las etapas de desarrollo de un parche del *kernel* de Linux; sus actividades permiten certificar un diseño sólido del archivo resultante, el cual es examinado en la lista de correo correspondiente por otros desarrolladores –en un proceso de revisión reciproco y de cooperación que permite verificar si es o no, candidato para fusionarse con el árbol de la línea principal– y así garantizar la calidad del código, para determinar el éxito final del proyecto. Los desarrolladores de la línea principal del *kernel* de Linux, siempre van están preocupados por la estabilidad del sistema; por lo tanto, ante un error en un módulo específico, lo primero que hacen es reconocer el problema, después realizan una discusión con la comunidad para resolver conflictos y, con base en este análisis buscan una solución para implementarla, garantizando el mantenimiento a largo plazo de forma responsable por parte del *mantenedor* del parche (Corbet, 2013). Los resultados sugieren un conjunto de elementos que especifican el modelo de desarrollo del software libre y la manera en que se organizan sus desarrolladores en comunidades virtuales.

V. CONCLUSIONES

El proceso utilizado en los proyectos de software libre y de código abierto [FOSS] emerge, con respecto a los procesos tradicionales de la ingeniería del software, con un enfoque basado en el software, de una manera más informal, que permite realizar liberaciones rápidas, a menudo en un entorno de cooperación en red, que le permite a los participantes desarrollar software con un alto nivel funcional y operativo para la comunidad (Scacchi, 2004; González et al., 2003). Los proyectos de software libre se organizan a través de comunidades virtuales de desarrollo, con participantes dispersos globalmente, que utilizan la comunicación mediada por computador para crear identidades de grupo, donde el acceso al código fuente, como recurso común, permite realizar acciones colectivas que repercuten en la cooperación para mitigar conflictos (Kollock & Smith, 1996; González et al., 2003).

La línea principal del *Kernel* de Linux fue construida por los desarrolladores alrededor de una cadena de confianza que se encuentra representada en la comunidad virtual, la cual se reúne en su conjunto a través de listas de correo y define, para el mantenimiento de los parches del *kernel*, una serie de actividades de desarrollo de software para garantizar su calidad (Corbet, 2013).

VI. REFERENCIAS

Arulkumar, N. & Chandra, S. (2012). The impact of release management and quality improvement in open source software project management. *India Applied Mathematical Sciences, 6*(62), 3051-3056

Corbet, J. (2013). *A guide to the kernel development process*. Recuperado de https://www.kernel.org/doc/Documentation/development-process/

Cox, A. (1998).Cathedrals, bazaars and the town council. Recuperado de http://news.slashdot.org/story/98/10/13/1423253/featurecathedrals-bazaars-and-the-town-council

Crowston K. & Scozzi B. (2002). Open source software projects as virtual organizations: competency rallying for software development. *IEEE Proceedings Software, 140* (1), 3–17

Crowston, K., Annabi, H., & Howiston, J. (2003). Defining open source project success. En Proceedings *of the 24th International Conference on Information Systems (ICIS 2003)*, Seattle, Wash, (pp.327-340)

Cuesta, B. & Parra, J. (2013). *Curso e-learning para el mejoramiento de las competencias de cooperación en el desarrollo de proyectos de software libre orientado a los lineamientos metodológicos de la programación extrema* [tesis de maestría]. Universidad Autónoma de Bucaramanga, Colombia

Dempsey, B.J., Weiss, D., Jones, P., Greenberg, J. (1999). *A quantitative profile of a community of open source Linux developers* [technical report TR-1999–05]. School of Information and Library Science, University of North Carolina at Chapel Hill

DiBona, C., Ockman, S., & Stone, M. (1999). *Open sources: voices from the open source revolution*. Sebastol, CA: O'Reilly

Ehrenkrantz, J. (2003). Release management within open source projects. En Proceedings of the 3rd Workshop on Open Source Software Engineering at the 25th International Conference on Software Engineering, Portland, USA

Elliott, M. & Scacchi, W. (2004). Free software development: cooperation and conflict in a virtual organizational culture. En *Free/open source software development*, (pp.152-172). Hershey, PA: Idea

Feller, J. & Fitzgerald, B. (2001). *Understanding open source software development*. Reading, MA: Addison-Wesley

Fogel, K. (1999). *Open source development with CVS*. Scottsdale, AZ: Coriolis Open

Free Software Foundation (2014, junio 26). *Free software definition*. Recuperado de http://www.gnu.org/philosophy/free-sw.html

Gay, J. (Ed.). (2002). *Free software, free society: Essays by Richard M. Stallman*. Cambridge, MA: Free Software Foundation

German, D.M. (2002). The evolution of gnome. En *Proceedings of the 2nd Workshop on Open Source Software Engineering at the 24th International Conference on Software Engineering*, Florida, USA

Ghosh, R., Glott, R., Krieger, B., & Robles, G. (2002). Survey of developers. En Free/libre and open source software [parte 4, reporte final]. Maastricht, The Netherlands: International Institute of Infonomics University of Maastricht. Recuperado de http://www.flossproject.org/report/FLOSS_Final4.pdf

Ghosh, R., Glott, R., Krieger, B., & Robles, G. (2002). Free/libre and open source software: Survey and study. Maastricht, The Netherlands: International Institute of Infonomics, University of Maastricht

Giaglis, G. & Spinellis, D. (2012). Division of effort, productivity, quality, and relationships in FLOSS virtual teams: Evidence from the FreeBSD project. *Journal of Universal Computer Scienc*e, *18*(19): 2625-2645. doi:10.3217/jucs-018-19-2625

González, J., Seoane, J., & Robles, G. (2003). *Software libre*. Barcelona, España: Universitat Oberta de Catalunya

Hertel G, Neidner S, & Hermann S. (2003). Motivation of software developers in open source projects: an internet based survey of contributors to the Linux kernel. *Research Policy*, 32(7), 1159-1177

Himanen, P. (2002). The hacker ethic and the spirit of the information age. New York, NY: Random House

Koch, S. & Schneider, G. (2002). Effort, co-operation and co-ordination in an open source software project: GNOME. *Information Systems Journal, 12*(1), 27-42

Kollock, P. & Smith, M. (1996).Managing the virtual commons: cooperation and conflict in computer communities. En S. Herring (Ed.), Computer-mediated communication: Linguistic, social, and cross-cultural perspectives (pp. 109-128). Amsterdam, The Nethederlans: John Benjamins

Kollock, P. (1998). Social dilemmas: The anatomy of cooperation. *Annual Review of Sociology, 24*(1), 183-214

Lipnack, J. & Stamps, J.(2000). *Virtual Teams: People Working Across Boundaries with Technology*. New York, NY: John Wiley and Sons

Michlmayr, M. & Fitzgerald, B. (2012). Time-based release management in free and open source (FOSS). *Projects, 4*(1), 1-19

Mockus, A. (2000). A case study of open source software development: the apache server. En, *Proceedings of the 22nd International Conference on Software Engineering*. New York, NY: ACM

Mockus, A., Fielding, R.T., & Herbsleb, J. (2002). Two case studies of open source software development: Apache and Mozilla. *ACM Trans. Software Eng. and Methodology, 11*(3), 309-346

Narduzzo, A. & Rossi, A. (2003). Modularity in action: GNU/Linux and free/open source software development model unleashed [Quaderno DISA n. 78]. Recuperado de http://flosshub.org/system/files/narduzzorossi.pdf

Noll, J. & Scacchi, W. (1999). Supporting software development in

virtual enterprises. *Journal do Digital Information, 1*(4). Recuperado de http://www.usc.edu/dept/ATRIUM/Papers/DHT-VE98.html

Ostrom, E. & Walker, J. (2003). Trust and reciprocity: interdisciplinary lessons for experimental research. New York, NY: Russell Sage Foundation

Ostrom, E. (1990). Governing the commons: The evolution of institutions for collective action. New York, NY: Cambridge University

Ostrom, E. (1997). A behavioral approach to the rational choice theory of collective action: Presidential address, American Political Science Association. *The American Political Science Review*, *92*(1), 1-22

Ostrom, E. (2000). Collective action and the evolution of social norms. *The Journal of Economic Perspectives*, *14*(3), 137-158

Ostrom, E. (2001). Social dilemmas and human behavior. En *Economics in Nature*, (pp.23-41). New York, NY: Cambridge University

Parra, J. (2010). *Constructo para la evaluación de la cooperación en dilemas sociales de gran escala* [tesis de doctorado]. Universidad Nacional de Colombia, Bogotá

Raymond, E. (1999). The cathedral and the bazaar. *Knowledge, Technology & Policy, 12*, 3, 23-49

Robles, G. (2002). *Ingeniería del software libre, una visión alternativa a la ingeniera del software tradicional*. Recuperado de http://es.tldp.org/Presentaciones/200211hispalinux/robles/robles-ponencia-hispalinux-2002.pdf

Robles, G., Scheider, H., Tretkowski, I., & Weber, N. (2001). *Who is doing it? Knowing more about libre software developers*. Recuperado de http://widi.berlios.de/paper/study.pdf

Scacchi, W. (2002). Understanding the requirements for developing open source software systems. *IEEE Proceedings- Software, 149*(1), 24-39

Scacchi, W. (2004).Free and open source development practices in the game community. *Journal IEEE Software,21*(1), 59-66

Schweik, C. & English, R. (2007).Tragedy of the FOSS commons? Investigating the institutional designs of free/libre and open source software projects. Washington DC: IEEE Computer Society

M. Smith, & P. Kollock. [Eds.]. (1999). *Communities in cyberspace*. Londres, UK: Routledge

Sommerville, I. (2005). *Ingeniería de software* [7a ed.]. Madrid, España: Pearson

SPI (2014, abril 29). *Debian free software guidelines*. Recuperado de http://www.debian.org/social_contract.html#guidelines

Stallman, R. (1998). The GNU operating system and the free software movement. En C. DiBona, S. Ockman, & M. Stone [Eds.], *Open sources: Voices from the open source* revolution, (pp. 53-70). Sebastopol, CA: O'Reilly

Stallman, R. (2002). *Free software, free society: Selected essays of Richard M. Stallman*. Boston, MA: GNU

Stewart, K.J. & Ammeter, T. (2002). An exploratory study of factors influencing the level of vitality and popularity of open source projects. En L. Applegate, R. Galliers, & J.I. DeGross [Eds.], *Proceedings of the 23rd International Conference on Information Systems*, Barcelona, (pp. 853-857).

Stewart, K.J. & Gosain, S. (2001).An exploratory study of ideology and trust in open source development groups. En *Proceedings of the 22nd International Conference Information Systems*, New Orleans, LA (paper 63). Recuperado de http://aisel.aisnet.org/cgi/viewcontent.cgi?article=1142&context=icis2001

The Open Source Initiative [OSI] (s.f). *The open source definition*. Recuperado de http://opensource.org/osd

CURRÍCULOS

Byron Cuesta Quintero. Ingeniero de Sistemas (2002), Especialista en Informática Educativa (2005), y Especialista en Práctica Docente Universitaria (2009) de la Universidad Francisco de Paula Santander; Magíster en Software Libre (2013) de la Universidad Autónoma de Bucaramanga [UNAB]. Pertenece al Grupo de Investigación en Pensamiento Sistémico de la UNAB. Administrador de los sistemas de información y profesor catedrático del Departamento de Ingeniería de Sistemas de la Universidad Francisco de Paula Santander, Ocaña. Sus áreas de interés son las bases de datos y los sistemas operativos libres.

Jorge Andrick Parra Valencia. Ingeniero de Sistemas (1997) y Magíster en Informática (2003) de la Universidad Industrial de Santander [UIS]; Doctor en Ingeniería - Área Sistemas (2010) de la Universidad Nacional de Colombia - Sede Medellín. Profesor titular del Programa de Ingeniería de Sistemas, líder del Grupo de Investigación en Pensamiento Sistémico de la Universidad Autónoma de Bucaramanga. Presidente de la Comunidad Colombiana de Dinámica de Sistemas (2011-2012, 2012-2013), Secretario del Comité de Ética Institucional en Investigación y Director del Programa de Gestión de Sistemas de Información - Modalidad Virtual. Sus áreas de interés son la cooperación y la promoción de la cooperación para la solución de dilemas sociales.