Propuesta de lineamientos para la gestión de la cooperación en proyectos de software libre

Tesis de investigación presentada por

Alexánder Betancourth Moncada

Para obtener el título de

Magister en Software Libre

Dirigido por

Dr. Jorge Andrick Parra Valencia

Bucaramanga, Santander Enero de 2015

Facultad De Ingeniería De Sistemas

Grupo de investigación Pensamiento Sistémico

Universidad Autónoma de Bucaramanga

AUTORIZACION DEL DIRECTOR DE TESIS

Dr. Jorge Andrick Parra Valencia con cedula de ciudadanía 91288337 como director de tesis en el tema de investigación “Propuesta de Lineamientos para la Gestión de la Cooperación en Proyectos de Software Libre” para optar por el título de Magister en Software libre desarrollado por

Ing. Alexánder Betancourth Moncada, autoriza la presentación dado que reúne las condiciones para su defensa,

Bucaramanga – Santander, a los _____ días de Enero de 2015

Director de tesis

_____

Dr. Jorge Andrick Parra Valencia

Universidad Autónoma de Bucaramanga

AUTORIZACION DEL PRIMER EVALUADOR

Dr. Lilia Nayibe Gélvez Pinto como primer evaluadora de tesis en el tema de investigación “Propuesta de Lineamientos para la Gestión de la Cooperación en Proyectos de Software Libre” para optar por el título de Magister en Software libre desarrollado por Ing. Alexánder Betancourth Moncada, después de previa sustentación privada, autoriza la presentación dado que reúne las condiciones para su defensa,

Bucaramanga – Santander, a los _____ días de Enero de 2015

Calificador

_____

Dr. Lilia Nayibe Gélvez Pinto

Propuesta de lineamientos para la gestión de la cooperación en proyectos de software libre

Tesis de investigación presentada por

Alexánder Betancourth Moncada

Para obtener el título de

Magister en Software Libre

Dirigido por

Dr. Jorge Andrick Parra Valencia

Bucaramanga, Santander Enero de 2015

Facultad De Ingeniería De Sistemas

Grupo de investigación Pensamiento Sistémico

unab
Universidad Autónoma de Bucaramanga

CONFORMIDAD DE LA FACULTAD

La Universidad Autónoma de Bucaramanga y La Facultad de Ingeniería de Sistemas, después de sustentación pública realizada el día _____ de enero de 2015 del trabajo de tesis de investigación presentada por el Ing. Alexander Betancourth Moncada de título "Propuesta de Lineamientos para la Gestión de la Cooperación en Proyectos de Software Libre" elaborada dentro del campo de investigación del Grupo de Pensamiento Sistémico, para optar por el título de Magister en Software libre; Califica este trabajo por medio de la representación del director de tesis Dr. Jorge Andrick Parra Valencia, los evaluadores Dr. Lilia Nayibe Gélvez Pinto y la M.Sc. Claudia Isabel Cáceres Becerra han otorgado por _____ la calificación de:

unanimidad ó mayoría

Para constancia se firman en Bucaramanga – Santander, a los _____ días de Enero de 2015

Dr. Jorge Andrick Parra Valencia Director Tesis	Dr. Lilia Nayibe Gélvez Pinto Evaluadora Sustentación Final
M.Sc. Claudia Isabel Cáceres Becerra Evaluadora Sustentación Final	M.Sc. Wilson Briceño Pineda Decano Fac. de Ingeniería de Sistemas

Facultad De Ingeniería De Sistemas

Grupo de investigación Pensamiento Sistémico

Propuesta de lineamientos para la gestión de la cooperación en proyectos de software libre

Tesis de investigación presentada por

Alexánder Betancourth Moncada

Para obtener el título de

Magister en Software Libre

Dirigido por

Dr. Jorge Andrick Parra Valencia

Bucaramanga, Santander Enero de 2015

A Dios y mi familia que me dan las fuerzas.

Resumen

Esta investigación es un estudio de los diferentes mecanismos de cooperación utilizados en algunos proyectos representativos de software libre (Kernel GNU/Linux, FreeBSD, Gentoo/KDE, Joomla-CMS, PhapMyAdmin y Ruby on Rails) enmarcando los mismos en las teorías de cooperación utilizadas en diversos campos de las ciencias técnicas y humanas, con el fin de proponer lineamientos que mejoren la gestión de la cooperación en los proyectos de desarrollo de software libre que presenten características similares.

Agradecimientos

Agradezco a mi familia que ha puesto el estudio como una prioridad en mi vida, también a la Universidad Autónoma de Bucaramanga por la gestión tan importante para Colombia de traer este tipo de programas internacionales y así darles esta oportunidad de capacitación a los colombianos.

PROPUESTA DE LINEAMIENTOS PARA LA GESTION DE LA COOPERACIÓN EN PROYECTOS DE SOFTWARE LIBRE

Tesis de investigación

Tabla De Contenido

Prologo

Esta tesis de investigación está dirigida a buscar en la observación de comportamientos en la producción de software libre mecanismos de cooperación que presenten en algunos momentos de vida del proyecto deficiencias notables, comparando el tiempo de vida del proyecto con factores tales como el número de participantes, la cantidad de aportaciones individuales de los miembros más destacados, versiones en realización y versiones publicadas.

A través de la teoría de cooperación para dilemas sociales de gran escala se logra enmarcar estos mecanismos de cooperación para su correspondiente identificación en el sucesos apreciables en un momento dado del proyecto de desarrollo, por medio de este proceso de observación y relación de factores de comportamiento y aclarando las características circunstanciales que denotan un espacio de validez para este estudio se proponen unos lineamientos para la mejora de cooperación, de tal modo puedan ser evaluados en un amplio rango de proyectos de software libre similares. Y dejar abierta la perspectiva para futuras investigaciones que puedan especificar campos de acción más específicos dentro del mundo del software libre.

1. INTRODUCCIÓN

En este trabajo de investigación se plantearan los principales lineamientos de cooperación en los proyectos de software libre, haciendo un estudio de los diferentes mecanismos de cooperación utilizados en proyectos representativos del software libre y enmarcando los mismos en las teorías de cooperación utilizadas en diversos campos de las ciencias técnicas y humanas.

En los proyectos representativos de software libre (Kernel GNU/Linux, FreeBSD, Gentoo/KDE, Joomla-CMS, PhapMyAdmin y Ruby on Rails) se efectúan observaciones con la pretensión de aportar lineamientos para aplicar en proyectos de desarrollo de software libre, de forma que la gestión de la cooperación en estos proyectos sea mejorada notablemente, mitigando problemas visibles como la deserción, la falta de incentivos y la división de la comunidad.

1.1 ANTECEDENTES

El estudio de la cooperación en los proyectos de software libre presenta antecedentes visibles en el estudio de teorías de la cooperación tales como los dilemas sociales, en esta se estudia la interacción del individuo para lograr objetivos grupales que de alguna forma lo benefician directa o indirectamente.

1.1.1 Los dilemas sociales

Los dilemas sociales son esas situaciones en las que individualmente las ideas y actos propenden por lograr el bien personal, produciéndose varias opciones para el individuo como es la de colaborar o la de no colaborar, también existe la opción de colaborar parcialmente; es así como la situación colectiva se vuelve dependiente de la motivación y acción individual, por lo general el individuo es poco consiente de la dependencia de la situación colectiva con respecto a las decisiones e intereses personales.

Existen varios ejemplos de la teoría de juegos que esquematizan las diferentes situaciones de intereses personales de los individuos de un grupo, uno de ellos es el dilema del prisionero, en este se plantea la necesidad de colaborar grupalmente con una decisión unificada o el castigo por elegir obtener mejor retribución que su compañero.

En la teoría de juegos se estudia detenidamente el equilibrio de NASH, que describe una situación en la cual los individuos poseen una estrategia,

conociendo con anterioridad la estrategia de los otros individuos teniendo de esta forma la mejor manera de proceder en la situación, siendo poco o nada lo que ganaran si cambian su estrategia.

Podemos citar entonces comúnmente en los dilemas sociales algunas variables comunes para los problemas típicos, aquellas son:

Elección individual

Elección grupal

Estrategia individual

Estrategia de cooperación

Cada una de estas variables describe una parte del comportamiento colectivo en los dilemas de sociales de cooperación.

1.1.2 Los dilemas sociales de gran y pequeña escala

Los dilemas sociales independientemente de su escala poseen la similitud de motivación para el individuo, pero es de notar que los dilemas de pequeña escala difieren sustancialmente en las situaciones de dilemas de gran escala[1].

[1]Ver Jorge Andrick2012 , Evaluación de la Cooperación en Dilemas Sociales de Gran Escala, Pag. 23

Existen diferencias muy claras en la dinámica de cooperación en dilemas de gran escala con respecto a los de pequeña escala podemos decir que los de pequeña escala poseen la facilidad para el individuo de racionalizar fácilmente su comportamiento, motivación y resultados gracias a la observación de la colectividad. También hace que cada individuo pueda comprometerse con los intereses de la colectividad analizando a su vez la interdependencia que poseen mutuamente los individuos y el grupo de pequeña escala; Pero no pasa lo mismo con los dilemas de gran escala en los cuales si bien la motivación es la misma para los individuos y el colectivo, no son tan directamente observables las consecuencias como producto de un aporte individual, podemos tomar el caso de una pequeña empresa de comidas conformada por pocos miembros para los cuales existe suficiente motivación de hacer rápido y bien las actividades asignadas pues el resultado se observara casi inmediatamente, con directas responsabilidades para el individuo; Ahora podemos citar el caso de gran escala del cuidado del medio ambiente situación en la cual la dependencia de resultados gracias a los aportes individuales no es fácilmente evidente, también el compromiso entre el individuo y la colectividad es asumido sin la suficiente motivación.

Es así, que en los dilemas de gran escala la dependencia de resultados gracias a la cooperación individuo-grupo no es tan evidente motivo por el cual el individuo preferirá elegir sus opciones de un modo egoísta, en el caso del medio ambiente entonces deteriorando el mismo y desmejorando las posibilidades de sostenibilidad[2].

Cuando se sabe que muchos individuos actúan con reciprocidad en situaciones particulares, existe la ventaja de que cualquiera gane la reputación de ser

[2]Ver Jorge Andrick 2012 , Evaluación de la Cooperación en Dilemas Sociales de Gran Escala, Pag. 24

confiable y se comporte con reciprocidad. En el núcleo de una explicación conductual de niveles de cooperación mayores a los previstos, en la mayoría de los dilemas sociales se trata de conectar entre "la confianza que los individuos tienen en los demás, la inversión que los demás hacen en reputaciones confiables, y la probabilidad de que los participantes usarán normas recíprocas[3].

1.1.3 La reputación

En las relaciones cooperativas juega un papel importante la reputación ya que aunque las acciones grupales son una sumatoria de los procederes individuales, estos procederes se efectúan al interior de un grupo, hecho que conlleva a una valoración (reputación individual) por el historial acumulado dentro de la militancia de un grupo o comunidad. De otra forma la reputación grupal es muchas veces gran aliciente para participar de las actividades de una comunidad sin importar primariamente el tamaño, mientras que su reputación como comunidad sea altamente considerada en el entorno entonces el individuo se auto recompensara gracias al bien valorado reconocimiento externo de su comunidad o grupo.

[3]Ver Ostrom, 1998, EL GOBIERNO DE LOS BIENES COMUNES Ed. en español, Pag. 13

1.1.4 La confianza

La confianza juega un papel primordial en las situaciones tanto de dilemas sociales de pequeña escala como los de gran escala siendo una variable determinante a la hora de cooperar con el colectivo, de no cooperar o de condicionar la misma.

El diccionario de la Real Academia Española de la lengua define la confianza como:

Esperanza firme que se tiene de alguien o algo.

Seguridad que alguien tiene en sí mismo.

Se puede ver que la definición es bastante acertada para la teoría que se estudia en los dilemas sociales; también existe una definición en Wikipedia en la cual expone más concretamente el termino para este contexto, dice “En sociología y psicología social, la confianza es la creencia en que una persona o grupo será capaz y deseará actuar de manera adecuada en una determinada situación y pensamientos. La confianza se verá más o menos reforzada en función de las acciones. La psicología evolucionista supone que los seres humanos poseen habilidades para interpretar las intenciones de los demás, lo que facilita la reciprocidad (Barkowet al., 1992)[4].

[4]Ver Jorge Andrick 2012 , Evaluación de la Cooperación en Dilemas Sociales de Gran Escala, Pag. 18

1.1.5 Reciprocidad

Bien sea la reciprocidad una acción de corresponder devolviendo lo que se ha recibido, tal parece que en los proyectos de software libre viene siendo una relación altruista cuando simplemente se aporta a un proyecto con la mira de que la reciprocidad sea manifestada para el individuo en algún tipo de reconocimiento bien sea local dentro de los más allegados al proyecto mismo o de forma general para todo el proyecto como tal. Ahora bien si se toma en cuenta las formas asociadas de reconocimiento como pueden ser los créditos dentro del proyecto o ser parte del historial de aportaciones importantes dentro de este, entonces la relación quedaría bien satisfecha y planteada desde el principio cuando el individuo es acogido dentro de la comunidad para el desarrollo de alguna tarea particular.

1.1.6 El software libre

El software libre nació de la mano del propio software en la década de los años 60[5]. Entonces las gigantescas maquinas a las que llamaban computadoras hacían uso de programas cuyo código fuente estaba a la vista de todos (los que querían

[5]Universidad Rey Juan Carlos , Compilación de ensayos sobre software libre pag.1

verlo, por supuesto) y se podía distribuir libremente. Esto provocó que ya en esos tiempos, prehistóricos desde el punto de vista de la informática, existiera una pequeña comunidad de científicos y programadores que intercambiara código, a la vez que informes de errores e ideas. El software por entonces no era más que un valor añadido a las carísimas computadoras y se solía distribuir gratuitamente por los fabricantes.

La situación cambió radicalmente con el descenso del precio de las máquinas y sus componentes (el hardware) y la progresiva necesidad de un software más potente y con mayores funcionalidades. La ventaja competitiva que el intangible daba a las maquinas llego hasta el punto en el que incluso había gente que estaba dispuesta a pagar dinero por él. Esto que en sí no es necesariamente malo, provoco sin embargo un giro radical en la industria informática: las primeras compañías exclusivamente dedicadas a la creación de software aparecieron en el horizonte y se hicieron fuertes en el mercado. En aras de maximizar beneficios (económicos y estratégicos), una de sus tácticas habituales era limitar hasta más no poder lo que el usuario podía hacer con el software que creaban.

De repente, algo tan natural hasta pocas fechas antes como compartir un programa o su código se convirtió en una práctica deleznable y que atentaba no solo contra el creador del software, sino contra toda la industria del software y, por si acaso, también contra la sociedad y su bienestar.

No fue hasta mediados los años 80, cuando Richard Stallman formalizo las ideas básicas del movimiento del software libre que está revolucionando la industria del software. El software libre, tal y como lo conocemos hoy, dio sus primeros pasos con un manifiesto en favor de la libertad de expresión y un proyecto conocido hoy mundialmente, el proyecto GNU. Y con él, vio la luz probablemente una nueva forma de ver y entender el software y los bienes intangibles que se ha visto

acelerada con la masiva implantación de Internet en las postrimerías del siglo XX y principios del actual.

Ha sido el binomio Internet-software libre (junto con otros ingredientes secundarios) el que ha propiciado uno de los cambios más radicales de las últimas décadas. Nótese que es difícil imaginarse el éxito del uno sin el otro. La mayor parte de la infraestructura de Internet se sustenta sobre código libre, mientras que las posibilidades colaborativas que ofrece Internet han sido vitales para el pleno desarrollo del software libre como elemento tecnológico y filosófico. Sin embargo, mientras el cambio tecnológico basado en Internet ha tenido una fuerte implantación en el mundo occidental, la mentalidad ligada al software libre está tardando algo más en calar en la sociedad. Pero no cabe duda de que paulatinamente esté ganando en importancia.

Y es precisamente en este punto donde se ha encontrado un mundo que está empezando a asimilar estos cambios y lo que conllevan. Los apartados en este libro presentan y toman posición precisamente en algunos de los debates de más radiante actualidad que tienen que ver con estos aspectos.

1.2 PLANTEAMIENTO DEL PROBLEMA

La situación a estudiar es el uso reportado de los mecanismos de cooperación en la gestión del proceso de desarrollo de software libre, de tal forma se puedan descubrir algunas formas comunes de interacción entre los participantes de desarrollos en proyectos de software libre.

Después de estudiar cuales mecanismos de cooperación son los más utilizados y comparar la gama de resultados que estos producen en la vida de los proyectos de software libre se entrara proponer nuevos usos de los mecanismos de cooperación o también nuevos características que puedan presentar mejoras al proceso de desarrollo de software libre.

Aunque no todos los proyectos de software libre tienen como fin técnico los mismos objetivos podemos afirmar que la filosofía de cooperación es común a cualquier individuo de una comunidad en particular ya que el contexto de producción social se ve sustentado por la creencia en particular en la necesidad de un cubrimiento legal y filosófico por medio de una licencia de software abierto o libre en particular y con determinadas reglas de juego para la misma. Lo que alienta a los participantes a producir distintos segmentos de software independiente al objetivo técnico particular de cada uno.

Ahora bien la cooperación en el software libre no se limita únicamente al desarrollo concreto de código programado en algún lenguaje, en muchas oportunidades para que este código sea portado a plataformas dirigidas a usuarios en idiomas diferentes a los nativos del proyecto es necesaria la colaboración de personas que necesariamente no tienen que poseer conocimientos avanzados y ni siquiera básicos de programación de computadoras para ejercer los siguientes oficios bien valorados e indispensables en la comunidad del software libre:

Traductor: personas dedicadas a traducir manuales, traducción de interfaces de usuario entre otros.

Probador de software: que normalmente es la persona dedicada a probar versiones alfa, beta o en estudio, en diferentes condiciones o con nuevos tipos de hardware

Replicador: en cualquier programa se presentan fallas pero no en cualquier situación, estas son las personas encargadas de reproducir la situación para proveer exactamente el conjunto de características ambientales que reproducen la falla para reportarlas adecuadamente a un programador que se encargara de corregirla.

En estas condiciones es observable que el campo de la cooperación en el desarrollo de proyectos de software libre es un tema que debe ser abordado con suficiente entendimiento y certeza a fin de poder establecer claramente las relaciones que delinean las variadas formas de colaboración, motivación y dado en algunos casos con respecto a la empresa privada una visible forma de competición.

Las primeras preguntas que las personas usualmente hacen acerca del software libre son: ¿Cómo trabajan en el software libre? ¿Qué mantiene a un proyecto funcionando? ¿Quién toma las decisiones?, La meritocracia, el espíritu de cooperación, código que evoluciona por sí solo, etc. El hecho es, la pregunta no es fácil de contestar. Meritocracia, cooperación, y código de programas binarios son en realidad una parte del algo, pero eso es una corta explicación decir “como los proyectos se mantienen básicamente día a día, y no decir nada acerca de como los conflictos son evitados o resueltos”[6].

[6] Karl Fogel, Producing Open Source Software, How to Run a Successful Free Software Project, Pag 59

En el proceso de producción del software libre la pregunta principal es ¿Qué es lo que la sociedad necesita?, "Necesita información que esté verdaderamente a disposición de los ciudadanos; por ejemplo, programas que la gente pueda leer, corregir, adaptar, y mejorar"[7], no solamente ejecutar. Pero lo que los propietarios de software típicamente ofrecen es una caja negra que no podemos ni estudiar ni modificar. La sociedad también necesita libertad. Cuando un programa tiene un propietario, los usuarios pierden la libertad de controlar una parte de sus propias vidas. Y sobre todo la sociedad necesita incentivar el espíritu de cooperación voluntaria entre ciudadanos. Cuando los propietarios de software nos dicen que ayudar a nuestro prójimo de una manera natural es «piratería», están contaminando el espíritu cívico de nuestra sociedad.

Si bien partimos de la premisa que reconoce las cualidades novedosas de los ejercicios de participación y cooperación en los proyectos de software libre también denominados en la sociopolítica como parte de la filosofía del «conocimiento libre» o «abierto», no vamos a caer en la tentación, tan frecuente, de mitificarlo o idealizarlo. Aprender y entender las circunstancias y los efectos del conocimiento abierto sólo será posible si subrayamos con énfasis sus potencialidades pero también recalcamos con energía sus límites. Con el fin de ser totalmente justos y no caer en un extremo romántico de nube rosa en la cual todas sus características sean idealizadas, como puede suceder en un lado opuesto de la moneda sin subrayar adecuadamente las debilidades o limitaciones que se podrían minimizar o mejorar mediante procedimientos que adecuen positivamente las respectivas falencias[8].

---

[7] Richard M. Stallman, Software Libre - Sociedad Libre, Por qué el software no debe tener propietarios, Pag 2

[8] Igor Sádaba, Dominio Abierto, Conocimiento libre y cooperación Pag 14

1.3 PREGUNTA DE INVESTIGACIÓN

En vista de los enfoques ofrecidos en la teoría de la cooperación es entonces apropiado preguntar con respecto a la dinámica de la producción del software libre lo siguiente:

¿Cuáles lineamientos de gestión de la cooperación en el desarrollo del software libre pueden extraerse del estudio de algunos casos representativos?, siendo que los proyectos de software libre difieren en sus características como funcionalidad del software objeto de producción, tiempo de vida, integrantes, toma de decisiones dentro de la comunidad de desarrollo y otras muchas.

En la vida de los proyectos de software libre la etapa inicial corresponde al momento en el cual la idea de conformación o enriquecimiento del proyecto sea tan atractiva que se pueda superar reveses en distintos aspectos no solo técnicos sino también netamente humanos lo que nos puede llevar en esta fase a pensar sobre el proyecto la siguiente pregunta: El proyecto, la interacción dentro del mismo, su organización y metas son lo suficientemente fuertes para *¿Permitir superar condiciones iniciales de confianza no suficientemente consistentes como para generar cooperación basada en confianza?* [9].

En lo que se puede coincidir es que de alguna forma técnica o empírica los proyectos de software que gozan de larga vida y han logrado crecer o mantenerse a flote tanto como comunidad de desarrollo, y también, como proyecto de software popularmente usado, deben de tener en común algunas pautas de cooperación

[9] Ver Jorge Andrick 2010 Constructo para la evaluación de la cooperación en dilemas sociales de gran escala,Pag.35

que la promueven de alguna manera, siendo las respuestas a estas preguntas un común a los proyectos mismos: *¿Son capaces de generar y mantener la cooperación en forma sostenible?, ¿ Son capaces de generar cooperación como norma de largo plazo?*[10] Ya que aunque es bien sabido que las grandes comunidades de desarrollo software libre tienden a estar jerarquizadas y la gran mayoría de colaboradores directos son colaboradores que permanecerán en el proyecto de forma esporádica, resulta entonces ser una cuestión a estudiar la forma en que las comunidades encuentran una rotación segura encaminada a corresponder en las tareas de desarrollo que comprenden el proyecto.

Ahora bien el factor humano es determinante en la supervivencia de un proyecto de software libre, pero existen variables técnicas que afectan enormemente el quehacer de los participantes en los proyectos de software libre, como lo son, la adaptación del software al constante movimiento de nuevo hardware producido por cientos de diferentes compañías. En otras situaciones la compatibilidad de licencias entre las partes de un software en desarrollo, aun en casos, raramente dicho de componentes o programas que tienen alguna clase de licenciamiento libre. Y a veces no el más trasmano de los trabajos de actualizar software con nuevas características o correcciones pero tal vez igual o más trabajoso que los anteriores. Así estas realidades en los proyectos de software libre son dificultades que se deben enfrentar a lo largo de la vida del proyecto, normalmente en muchas ocasiones para subsistir como proyectos triunfantes a través del tiempo. Es entonces que el individuo tendrá como una parte de sus razones atractivas para participar en un proyecto de software en particular, respuestas a los interrogantes *¿Son capaces de enfrentar los participantes las escaladas de tentación de*

[10] Ver Jorge Andrick 2010 Constructo para la evaluación de la cooperación en dilemas sociales de gran escala,Pag.35

desertar?[11] *¿Permiten la percepción de daño por alguna desventaja u obstáculo asegurar la cooperación en condiciones iniciales de cooperación no favorables?*

La vida de un proyecto de software libre se compone en salud operacional que corresponde al aspecto técnico del proyecto y la supervivencia al factor humano. "*La salud operacional es la habilidad en curso del proyecto para incorporar contribuciones nuevas de código y desarrolladores nuevos, y ser receptiva para incluir nuevos características. Supervivencia es la habilidad del proyecto para existir independientemente de cualquier patrocinador o participante individual — piense acerca de eso como la probabilidad que el proyecto continuaría aun si todo de sus miembros fundadores se moviera adelante para otras cosas. El éxito técnico no es difícil de lograr, pero sin una base robusta del desarrollador y una fundación social, un proyecto puede ser incapaz para manejar el crecimiento que el éxito inicial trae, o la partida de individuos carismáticos. Hay formas diversas para lograr esta clase de éxito. Algunos implican una estructura formal de autoridad, por cuáles debates son resueltos, los desarrolladores nuevos son invitadas a entrar (y algunas veces fuera), características nuevas tenidas prevista, etcétera".*[12]

[11] Ver Jorge Andrick 2010 Constructo para la evaluación de la cooperación en dilemas sociales de gran escala,Pag.36

[12] Ver Karl Fogel, Producing Open Source Software, How to Run a Successful Free Software Project, Pag 59

1.4 HIPOTESIS

Mediante la aplicación de lineamientos de gestión basados en mecanismos de cooperación es posible establecer el nivel de acción colectiva en los proyectos de desarrollo de software libre. Comprender los procesos de cooperación en los proyectos de software libre es indispensable para explicar los mecanismos tanto científicos, técnicos, humanos y empíricos necesarios para aumentar la efectividad de los mecanismos usados a lo largo de la vida de los diferentes proyectos de software libre, de esta forma se puede a su vez platear visiblemente los momentos de baja racha, dificultades y debilidades a lo largo de la existencia del mismo.

1.4.1 Cooperar para funcionar

Parece ser la ley número uno en los proyectos de software libre “cooperar para funcionar” significaría en el software libre lo que para la evolución de la vida fue el agua, tendiente a mejorar cada vez más el software del proyecto también se mejoran los conocimientos, la cohesión de la comunidad, los objetivos del proyecto mismo y en el argot del software libre la reputación, una primera condición necesaria para la adhesión de nuevos miembros al equipo.

En otros casos tal vez no planteados como parte de la continuidad de un proyecto sino más bien como el inicio de uno nuevo, la bifurcación, movimiento tendiente a aprovechar el código fuente, ya funcional elaborado en un proyecto en específico, ofrece a una comunidad en algunos casos algo diferente a una nueva dirección, tal

vez la evolución de las metas originales del proyecto de la comunidad desarrolladora pero también nuevas funcionalidades necesarias para la comunidad de usuarios, situación observable en las distintas versiones de GNU Linux, variedad en las cuales se pueden encontrar distribuciones orientadas a administradores de servidores, diseñadores gráficos, músicos y artistas, hasta versiones con orientación a la alta usabilidad por parte del usuario común. Tal vez la bifurcación de versiones de software libre sea resultado de un más alto sentido de cohesión y cooperación entre los líderes que emergen para crearla y darse al compromiso de mantenerlo con suficiente vida.

Se puede observar que en los proyectos de software libre existe un balance entre jerarquías existentes versus el número de participantes, podemos ver este tipo de organización y distribución burocrática como una situación análoga al dilema de *"la tragedia de los comunes"*[13], aunque los niveles organizativos no son un recurso agotable, si lo son los participantes con actividades específicas en cada proyecto que deben de ser coordinadas para poder llevar a cabo un desarrollo exitoso. De esta manera para los encargados de la planeación y organización de proyectos de software libre el liderazgo, el seguimiento y la evaluación que las personas cabezas de células, grupos o pequeñas comunidades de desarrolladores son capaces de llevar a cabo eficientemente suele ser un dilema a tener en cuenta con sumo cuidado. Siendo el recurso humano indispensable y no sustituible en el desarrollo de los proyectos, en otras ocasiones en la continuidad de tareas ya iniciadas o seguidas con cierta continuidad por algunos miembros en particular, se hace necesario que los encargados de coordinar grupos tengan capacidades más allá de la mera suficiencia técnica para cumplir con las metas de desarrollo ya que las personalidades y los caracteres salen a flote durante el proceso de desarrollo,

---

[13] Ver Peter Kollock, SOCIAL DILEMMAS: The Anatomy of Cooperation, Pag 188

toma de decisiones, en general dentro de cualesquiera actividad de un proyecto que obligue a la interacción humana para comunicarse; ya que en estas interacciones son visibles en ocasiones personas de variados tipos ya sean usuarios, desarrolladores o probadores de software que aunque no son directamente groseros en sus comunicaciones suelen abusar y manipular malamente los recursos de desarrollo, soporte o comunicación, cuestión que resulta en pérdida de tiempo y esfuerzo para la ejecución de las tareas, resolución de otros problemas que sean más decisivos para las actividades.[14]

En el siguiente cuadro se pueden ver tres situaciones organizativas que expresan el balance necesario o falta de este, en las jerarquías o liderazgo de producción en los proyecto de software libre:

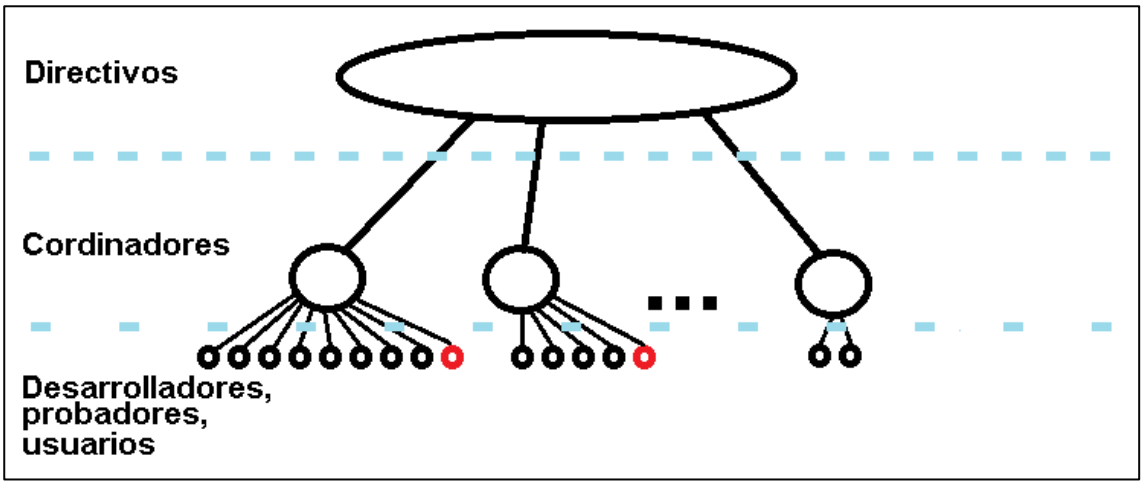

Figura 1 – *Desbalance en organización subgrupo desarrolladores*

[14] Karl Fogel, Producing Open Source Software, How to Run a Successful Free Software Project, Pag 94

La figura 1 muestra el desbalance en el número de desarrolladores, para ser coordinados por líder de tarea, siendo los pequeños círculos rojos personas con importante historial de obstrucción del normal desarrollo de las actividades y teniendo los pequeños círculos negros como el resto de la comunidad de usuarios y desarrolladores.

Se observa también que el número de usuarios/desarrolladores no es igual por cada coordinador, pero esto puede también ser interpretado como si el número de estos fuese necesario en cada subgrupo para lograr las tareas propuestas. También pudiese darse el caso de que los trabajos fuesen tan específicos que hubiere muy poca disponibilidad de personas para ejecutarlos.

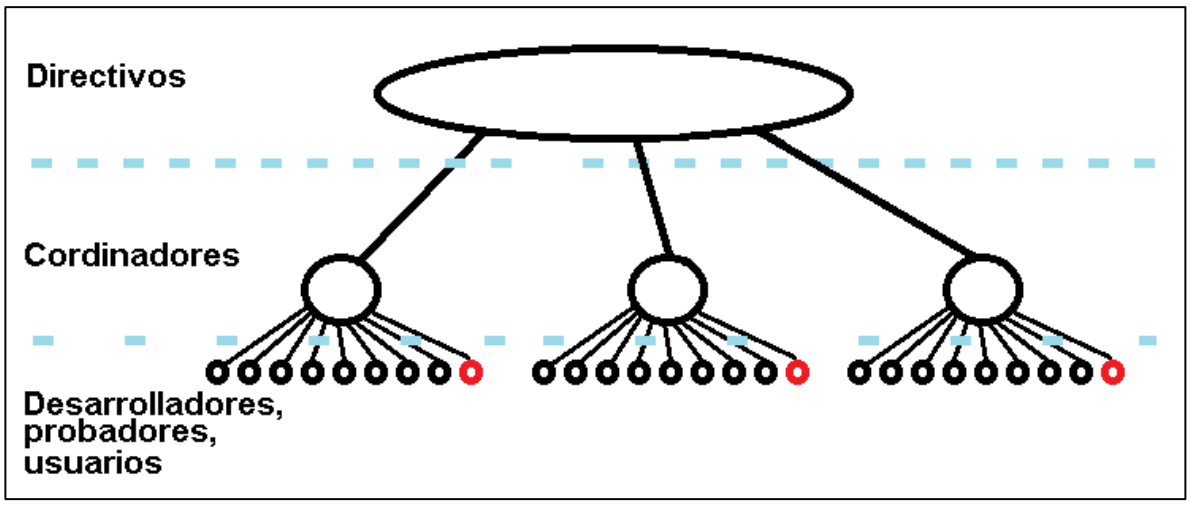

Figura 2 – *Desbalance en organización subgrupo desarrolladores*

La situación de figura 2 muestra el caso bien conocido se proyectos que tienen demasiados colaboradores, llegando al límite que en ocasiones obliga a dejar versiones de software sin soporte oficial o programado por proyectar que en un futuro próximo los pedidos de diversidad de casos asociados serán tantos que no es posible tener un número suficiente de responsables de gestionar y dar camino de solución a estos.

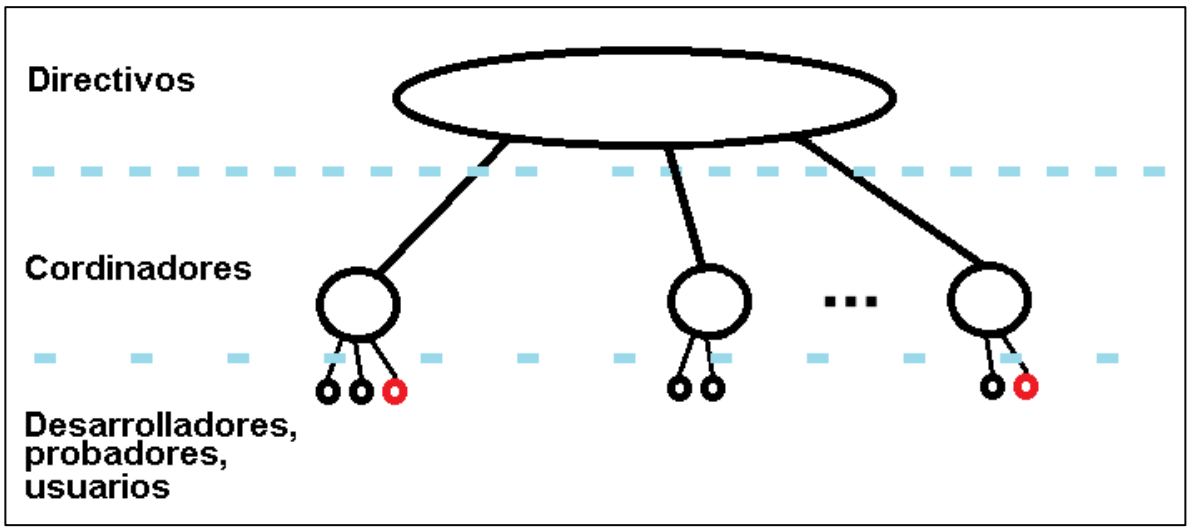

Figura 3 - *Desbalance en organización subgrupo desarrolladores*

La figura 3, muestra la situación en la cual los colaboradores son demasiado pocos, realidad que es interpretada desde el punto de vista de ser un proyecto en el cual se necesitan personas extremadamente capacitadas, también puede ser la situación en que el software es usado por un público demasiado exclusivo por sus características especiales. Y menos glamurosa pero no menos importante suele pasar también en proyectos en los cuales el número de participantes de la comunidad ha migrado a otro proyecto o por alguna otra razón el proyecto de desarrollo está llegando a su fin

1.5 OBJETIVOS

1.5.1 Objetivo general

Proponer lineamientos para la gestión de la cooperación en proyectos de desarrollo de software libre que minimicen la deserción y la baja cooperación.

1.5.2 Objetivos Específicos

- Revisar la literatura sobre el uso reportado de los mecanismos de cooperación en la gestión del proceso de desarrollo de software libre.

- Definir los casos representativos basados en el tiempo de vida del proyecto y la importancia dentro de la comunidad del software libre para el estudio de las dinámicas de cooperación.

- Derivar lineamientos para la gestión de la cooperación en proyectos de software libre desde el estudio de los mecanismos de cooperación identificados en la revisión de la literatura.

- Evaluar dichos lineamientos de aplicación mediante su confrontación con los fundamentos de la teoría de la cooperación y la teoría del desarrollo de proyectos de software libre.

1.6 LIMITACIONES

En el estudio de los comportamientos de cooperación para el desarrollo de software libre existen unos limitantes marcados tales como:

Aunque el código es abierto y las comunidades técnicamente son abiertas a colaborar para el mejoramiento del software, no siempre para la medición o estudio de los comportamientos, relaciones colaborativas que conllevan al desarrollo de las mismos, ya que en vista que los integrantes son individuos que no siempre son fáciles o rápidos de contactar.

Las muestras que puedan ser obtenidas pueden ser no lo suficientemente representativas, lo que conllevaría a una errónea interpretación o conclusiones fuera de lugar con respecto a la realidad del asunto tratado,

Aunque existen personas activas dentro de un proyecto y su interés de colaboración es bastante alto, por la poca periodicidad con la cual se le asignan tareas debido a la naturaleza misma de su labor, los periodos de pausa entre tareas suelen ser altos, a veces de varios meses y hasta años.

2. Revisión de la Literatura

2.1 Fuentes de información

En la búsqueda de material escrito lo suficientemente valido en calidad y en actualidad para el estudio de la gestión de la cooperación en el desarrollo del software libre, se han planteado los siguientes pasos para brindar la mayor confianza en esta parte del proceso.

2.3 Ordenamiento Bibliográfico

Se realizó la revisión de literatura organizándola por orden cronológico y seleccionando las que fundamentalmente son pertinentes al tema por mostrar alguna relación con el ejercicio de la cooperación en el desarrollo de software libre, o como estrategia solucionadora de problemas para un gran número de personas, teniendo en cuenta criterio de admiración y relevancia dentro de la comunidad científica.

2.4 Palabras claves para la búsqueda

Las palabras claves a buscar en el sistema Bibliográfico de la Universidad Nacional de Colombia (http://www.sinab.unal.edu.co/) fueron:

cooperation free software

Realizadas en el sistema de referencia bibliográficas SCOPUS mediante el menú "Referencias Bibliográficas → Herramientas bibliográficas → Botón Scopus",

Obteniendo los resultados de la búsqueda se ordenaron las listas primariamente por concepto de "cited by" -ranking de citaciones-, y por concepto de "relevance" -relevancia-, estas búsquedas se realizaron el día 24 de abril.

Otras búsquedas de referencias bibliográficas se efectuaron mediante los mismos términos:

Cooperación software libre cooperation

en el buscador google.com.co, también en google.com, ebooks google, siendo algunas resultados breves referencias a la información completa ubicada en otras páginas. La búsqueda se realizó entre las fechas del 11 al 17 de abril de 2013.

2.5 Criterios para inclusión de información

Después de haber visto que se debe de articular el tema con un tema que lo enmarque en una generalidad, el principal criterio que se aplicó para la selección de la información es que este se refiera directamente al ejercicio de cooperación como actividad para la producción de software libre, también se examinó si había relación ya fuera por conceptos históricos u otros similares como la maduración del concepto de software propietario, software libre e influencia de la cooperación; los anteriores con el fin de lograr la individualización del análisis de la cooperación en el desarrollo del software libre, y siendo el tema de los dilemas sociales la generalidad en la cual va enmarcada “la cooperación para el desarrollo del software libre”. Ahora el orden de importancia para los artículos es como se sugiere de forma cronológica, siendo los más actuales los principales a utilizar.

2.6 Programas para control y seguimiento de versiones

En la actualidad existen variadas herramientas para desarrollo de software, ahora bien cuando los programas se tornan en un tamaño monstruoso inmanejable por mero ordenamiento para un administrador de proyectos, es entonces que es necesario recurrir a un software especializado en gestionar código fuente para la asimilación del mismo por varios o tal vez miles de usuarios que intentaran hacer modificaciones sobre varias partes del código fuente que este gestione, al dia de hoy el concepto de programa de gestión de versiones ha ido evolucionando hasta llegar al nuevo concepto de *administrador de configuración de software (en ingles* Software Configuration Management*),* ya que en realidad es necesario que un software de control de versiones configure tantas situaciones como el control de usuarios, transacciones para la sustitución de código, apertura de preguntas y errores, y más comúnmente las versiones de un archivo de código fuente, las evoluciones de este archivo a través del tiempo, y el cruce de referencias con bibliotecas y similares.

Para este estudio nos interesa particularmente el uso que los usuarios le dan a este sistema ya que nos proveerá de una perspectiva directa y clara sobre el ritmo y tamaño de interacciones que tiene el usuario programador en el sistema que se usa para producir el producto, que el proyecto de software libre está creando. Cuestión que de otra manera seria prácticamente imposible obtener; los proyectos de software libre utilizan normalmente herramientas de control de versiones tales como *CVS, Subversion, SourceSafe, ClearCase, Darcs, Bazaar, Plastic SCM, Git, Mercurial, Perforce.*

Para propósito de celeridad y claridad en el presente estudio se estudiaran proyectos que utilicen github, ya que es una herramienta en línea que permite el

estudio del comportamiento de los usuarios programadores en periodos recientes de tiempo de forma pública, cuestión que es imprescindible para este estudio, ya que aunque algunos proyectos ofrecen sus fuentes de código de forma publica, no asi las interacciones de las comunidades están abiertas al público casi siempre o no de forma total , hecho que afecta totalmente el objeto de este estudio.

2.7 Marco Teórico

En este marco teórico se busca definir el área de producción del software libre y ubicarlo a su vez con la teoría de la cooperación desde el punto de vista de organización por medio de algunos casos que representan en buena parte la filosofía de cooperación para la producción en el software libre.

2.7.1 El software libre

En los años sesentas nace el software como como producto intangible que brinda una funcionalidad añadida a las maquinas, en esta misma década nace el software libre[15]. El código fuente del que hacen uso las computadoras esta libremente expuesto a los ojos de las personas que quieran estudiarlo, también se podía modificar con toda la libertad y distribuirlo libremente sin restricción alguna,

[15] Universidad Rey Juan Carlos, Sobre software libre Compilación de ensayos sobre software libre pag.1

es esos tiempos el software solo era una característica más, implementada por las grandes compañías como parte embebida de todo el hardware comercializado.

En los años setentas y ochentas los equipos computacionales comenzaron a entrar ampliamente en las empresas y los hogares, respectivamente, situación que con el tiempo requirió que el software presentara mejores funcionalidades y con una gama más amplia de posibilidades. Entonces después de comenzadas las competencias empresariales por desarrollos de software que presentara mejores posibilidades al usuario según su uso en la empresa, en el hogar o en múltiples escenarios, el software paso a ser una ventaja competitiva tanto para el vendedor como para el productor, es entonces que surge la demanda por nuevo software que potenciara enormemente las prestaciones que una máquina de computo podía ofrecer[16]. Los fabricantes de hardware entonces emprenden una campaña de limitación de capacidades de modificación y acceso para las funciones y demás añadidos que el usuario pudiese crear, modificar u operar por medio de nuevo software.

En esos momentos las practicas naturales de compartir para los desarrolladores de software pasan a ser casi delitos para el medio informático y para la industria del software, y poco después también para el mundo en general.

Es en los años ochenta cuando Richard Stallman inicia el movimiento de software libre con el proyecto bandera GNU que significa recursivamente GNU no es UNIX – GNU is no UNIX – en el cual se defiende la libertad de expresión por medio del software y se da forma a las licencias de software libre, las cuales tienen como su manifiesto y características especiales, distribuir el software desarrollado bajo las siguientes presunciones:

[16] Universidad Rey Juan Carlos, Sobre software libre Compilación de ensayos sobre software libre pág.1

1. La libertad de ejecutar el programa sea cual sea el propósito.

2. La libertad para modificar el programa para ajustarlo a tus necesidades. (Para que se trate de una libertad efectiva en la práctica, deberás tener acceso al código fuente, dado que sin él la tarea de incorporar cambios en un programa es extremadamente difícil.)

3. La libertad de redistribuir copias, ya sea de forma gratuita, ya sea a cambio del pago de un precio.

4. La libertad de distribuir versiones modificadas del programa, de tal forma que la comunidad pueda aprovechar las mejoras introducidas[17].

Es en la última década del siglo XX cuando el software libre emprende un camino hacia el éxito cuando encuentra en el internet la forma de comunicación perfecta en la cual la cooperación de varios miembros en la búsqueda de programas fabricados por las mismas comunidades para las cuales se orienta su uso, hace que el software libre gane mundialmente una escala significativa de confianza, teniendo como exponentes principales programas para internet como navegadores, servidores web, también sistemas operativos que se desempeñan como servidores de distintos tipos de servicios en empresas de diversos sectores.

En este siglo XXI es innegable el hecho de que el software libre ha sido catapultado en internet gracias a la facilidad de cooperación y distribuciones de la información para las comunidades de software libre.

[17] Richard M. Stallman, Software libre para una sociedad libre, pág. 19

2.7.2 GNU/Linux

El sistema operativo GNU/Linux nace como respuesta a la necesidad del movimiento GNU de tener en producción un sistema operativo libre en el cual pudiesen funcionar los programas desarrollados bajo el mismo tipo de licencia. En 1991 Linus Torvalds desarrolla una versión primaria de GNU/Linux gracias al mejoramiento de un sistema operativo que ya se distribuía con fines académicos para la época y que hoy por hoy aún se utiliza para este objeto, en los cursos de sistemas operativos, el autor de este maravilloso código fuente es el profesor Andrew Tanenbaum, este sistema operativo primario de nombre Minix era bastante limitado pero lo suficientemente bien diseñado para la escalabilidad, cuestión que aprovecho Linux Torvalds

Después de esto el proyecto GNU sigue utilizando en reemplazo de su propio sistema operativo no terminado – el HURD – el sistema operativo GNU/Linux en el cual pueden correr los programas que ya se habían desarrollado para producción como intérpretes de comandos, compiladores, bibliotecas y utilidades varias para programación y sistemas operativos.

2.7.3 Toma de decisiones en Linux

En los comienzos la metodología de desarrollo de GNU/Linux se tornaba de forma muy sencilla, una lista de correo en la cual se tomaban en cuentas todos los asuntos referentes al sistema, desde propuestas, errores, nuevas implementaciones para el desarrollo, también pasando por el envió de parches a la misma, increíblemente el parche era código totalmente explícito en el mensaje para que toda la comunidad pudiese estudiarlo, lo más notable era que los cambios y cuestiones relacionadas a estos eran desasidos en última instancia y hasta cierto punto por el mismo Linus Torvalds (hasta 2007[18]).

La forma de trabajo descrita es entonces en su desarrollo colectivo una verdadera muestra de cooperación, es de notar que las decisiones se toman en grupos de alta jerarquía, también podemos ver que el punto de decisión final lo hace Linus Torvalds, aunque la toma final de decisiones es autoritaria en la toma de decisiones sin embargo la dinámica sigue siendo cooperativa a pesar de esta característica

[18] Ver Jesús M. G., Joaquín S. P. y Gregorio R., Software libre, página 143

2.8 FreeBSD

FreeBSD surge como resultado de características y código fuente basado en 4.3BSD-lite que era parte del desarrollo puesto en marcha por la empresa AT&T, algunos de los módulos y funcionalidades originales no tuvieron cabida en FreeBSD ya que se tenían varios problemas legales para usar ciertas partes escritas del sistema.

En 1993 sale a la luz la primera versión de FreeBSD en CD-ROM, esto para mejorar la distribución, hacia personas que no poseían internet. El principal objetivo de FreeBSD era hacer portable el sistema. En noviembre de 1994 el sistema FreeBSD 2.0 es liberado sin la premura de problemas de propiedad intelectual por parte de la empresa Novell y la Universidad de Berkeley. Después de esta época FreeBSD ha pasado por muchas versiones estables y de desarrollo, al día de hoy es un éxito en las comunidades de software libre y también en algunas empresas.

La historia de FreeBSD está acompañada de otras distribuciones muy cercanas como son NetBSD y OpenBSD. NetBSD está basado en una gran variedad de software de libre distribución que incluye entre otros, a 4.4BSD Lite de la Universidad de California-Berkeley, a Net/2 (Berkeley Networking Release 2) el sistema de ventanas X del MIT y software de GNU. NetBSD ofrece entre sus características principales la seguridad, la estabilidad, compatibilidad en diferentes plataformas. En su filosofía esta escritura de código fuente de calidad, con implementación de estándares.

También dentro de la familia BSD se encuentra OpenBSD, su meta más importante es proveer un sistema libre de restricciones de tipo legal, desciende directamente de NetBSD ya que por diferencias entre Theo de Raadt y los demás miembros fundadores de NetBSD se plantea el proyecto OpenBSD, su eslogan principal trata de mostrar que es un sistema altamente seguro, en 2002 este rezaba así “Ningún fallo de seguridad remoto en la instalación por defecto en los últimos 6 años”. La filosofía de OpenBSD se resume como Libre, Funcional y Seguro. Libre ya que a través de sus múltiples auditorías a licencias del código fuente se logra garantizar esto, funcional pues existe un riguroso proceso para publicar las versiones, y seguro por las revisiones al código fuente.

2.8.1 Desarrollo en FreeBSD

El modelo de desarrollo utilizado por el proyecto FreeBSD está fuertemente basado en dos herramientas: el sistema de versiones CVS y el sistema de Informe de error GNATS. En tomo a estas dos herramientas gira todo el proyecto, como se puede comprobar por el hecho de que se ha creado una jerarquía a partir de las mismas. Así, sobre los *commiters* (aquellos desarrolladores con derecho a escritura al CVS) es sobre quien recae toda la soberanía del proyecto, bien directamente o bien indirectamente mediante la elección del *core group,* tal como se verá en el apartado siguiente[19].

[19] Ver Jesús M. G., Joaquín S. P. y Gregorio R., Software libre, página 143

2.8.2 Toma de decisiones en FreeBSD

El concepto de directores de FreeBSD, conocido popularmente como *core team,* se encarga de marcar la dirección del proyecto y de velar porque se cumplan sus objetivos, así como de mediar en caso de que haya conflictos entre *commiters.* Hasta octubre de 2000 era un grupo cerrado al que sólo se entraba a formar parte por Invitación expresa del propio *core team.* A partir de entonces, sus miembros son elegidos periódica y democráticamente por los *commiters.* La normativa más importante para la elección del *core team* es la siguiente:

Podrán votar los *commiters* que hayan realizado al menos un *commit* en el último año.

El Consejo de Directores se renovará cada dos años.

Los miembros del consejo de directores podrán ser "expulsados" con el voto de dos tercios de los *commiters.*

Si el número de miembros del consejo de directores es menor de siete, se celebrarán nuevas elecciones.

Se celebrarán nuevas elecciones si así lo pide un tercio de los *commiters.*

El cambio de normativa requiere un quòrum de dos tercios de los *commiters.*

Si bien la forma de tomar las decisiones en FreeBSD es democrática, esto en algunos casos hace que sea frecuente los retrasos en objetivos o actividades programadas para cierta fecha, la difícil sincronización entre los miembros hace

que las toma de decisiones y aun los desarrollos tomen más tiempo del requerido si todo el personal trabajara en su software en un sitio determinado[20].

NOTA: *Commiter significa en este contexto aprobador. Commit es aprobado*.

2.9 Las comunidades de software libre

En una comunidad de software libre la distinción principal es que los miembros se unen a esta para aportar en uno o varios proyectos de tal forma que se pueda contribuir a un bien común como lo es el software libre, aunque los proyectos de software libre tienen normalmente un objetivo principal, en la búsqueda de alcanzar este objetivo los miembros toman diferentes tareas para desarrollar como son las de programar fuentes, dar soporte a soluciones, probar programas en fase alfa, también traducir partes de un sistema a un lenguaje en particular. Es normal que una pequeña comunidad (cerca de 20 miembros) este orientada a cumplir un número limitado de objetivos de tal modo, mientras una comunidad numerosa (más de 50) en membresía posea un modo de organización más formalizada y capacidades más especializadas en sus miembros de tal modo se puedan desarrollar la mayoría de actividades pertenecientes de forma común a un proyecto de software libre.

[20] Ver Jesús M. G., Joaquín S. P. y Gregorio R., Software libre, página 148

En la filosofía del software libre se dice que un miembro se une a una comunidad de software libre por la necesidad de ver ciertas capacidades reflejadas en un software en particular, es por ello que en la filosofía del software libre el principal recurso es el usuario mismo, ya que *"en la economía del software libre son los usuarios quienes se encargan de este tipo de tareas y quienes proponen las correcciones[21] ".*

Las comunidades de software libre son más que solo la puesta en marcha de proyectos de programación, también realizan actividades como congresos, foros, capacitaciones y toda una serie de eventos que buscan fortalecer la integración de los miembros a la comunidad.

En el software libre se exponen dos vertientes de producción de software ejemplificadas por el escrito "la catedral y el bazar" de Richard M. Stallman, en la cual se habla de la catedral como el modelo privativo de software donde los usuarios solo pueden disponer de programas y capacidades de estos, que los altos jerarcas desean proveer sin espacio alguno a cambios; y el bazar como la forma en que los mismos usuarios pueden obtener lo que desean a través de la negociación y la búsqueda de características en programas de software que puede ser de utilidad no solo para ellos, sino también para otros, con la facilidad de compartir el producto recibido y modificado sin ninguna restricción.

Los desarrollos de software libre en ciertos proyectos están apoyados por un ente gubernamental o privado, que ayuda a desarrollar el proyecto, darle continuidad o hacer inversiones económicas que de otra forma serian tediosas de obtener. En esta misma forma de apoyo en el desarrollo se puede implementar el modelo de explotación, ya que las empresas privadas a veces se interesan en ciertos desarrollos particulares, como el caso de COREL que patrocino gran parte del

[21] Laurent Moineau y Aris Papathéodorou, Cooperación y producción inmaterial en el software libre. Enero 2000, Página 6

desarrollo de WINE como estrategia para portal el propio software de COREL hacia la plataforma GNU/Linux. Otra opción de explotación y a la vez producción de software libre es el pago por mejoras y soporte en la cual una empresa cliente paga por producir unas mejoras a un software libre en particular, en otras palabras por ponerlo a la medida de la misma, servicios asociados pueden ser la capacitación necesaria para el uso, la instalación y configuración, en fin todo lo que pueda estar relacionado al desarrollo o uso del software libre con la funcionalidad que se necesita.

2.10 Metodología de la investigación

En el estudio de esta investigación aplicaremos el método deductivo, ya que partiremos de datos, hechos y situaciones concretas en diversos campos para llegar a deducir particularidades relacionadas en la producción del software libre por medio de la cooperación, podemos citar entonces los métodos de las principales referencias.

En el libro “Evaluación De La Cooperación En Dilemas De Gran Escala” se puede observar el análisis de diferentes aspectos de las dinámicas de cooperación, sirviendo estas de aplicación a nuestro objeto de investigación. En el libro “Evaluación de Aplicabilidad de la Cooperación en la Crisis del Ozono” se analiza la dinámica de la cooperación de forma internacional para lograr plantear como esta sería un factor decisivo en la reducción de CFC en nuestro planeta. En la búsqueda de factores y practicas adecuadas para esta investigación para aplicar el método deductivo, el libro “Evaluación De La Cooperación En Dilemas De Gran Escala” nos demuestra entonces en toda su estructura la aplicación práctica del

método, analizando diferentes conflictos sociales e introduciendo una herramienta bastante novedosa como lo es la simulación, cosechando como resultado una serie de discusiones, y finalmente la sugerencia de disminuir por medio de ejercicios institucionales de cooperación la magnitud de los dilemas sociales.

El planteamiento escogido para esta investigación es fundamentarla en una estructura investigativa que exponga en la generalidad filosófica del software libre los mecanismos de cooperación y las dificultades inmediatas que estos enfrentan, siendo los puntos más sensibles para la gestión de la cooperación en el desarrollo de software libre como lo son *la alta tasa de deserción de los miembros y la baja cooperación en los proyectos de desarrollo de software libre.* En vista de tener de forma generalizada algunos comportamientos propios en las situaciones de producción de software libre, se procederá a la elaboración de tablas que desglosen de forma clara los puntos positivos y negativos de la estrategia que se sigue. De esta forma se podrán emprender propuestas de lineamientos que mejoren de alguna manera visible las dinámicas de funcionamiento.

Después de concluir con una serie de explicaciones a situaciones de funcionamiento del software libre, en encontraran las particularizaciones de estas, expresándolas como mecanismos utilizados en la realización de tareas en ciertos proyectos de software –casos representativos-. Estas demostraciones prácticas en los mecanismos que se utilizan en el software libre servirán entonces para trazar y comparar los lineamientos propuestos en la generalización del estudio, explicando que aunque los lineamientos no sean medibles en este estudio si sean objeto confiablemente gracias a la explicación valida de su uso o desuso en casos representativos del software libre.

2.10.1 Enfoque de la investigación

Para la realización de esta investigación se realizara un estudio netamente cualitativo que deje abiertas una serie de planteamientos que puedan ser estudiados y evaluados a futuro con datos cuantitativos, pueda ser con técnicas estadísticas de estudio de comportamientos, modelos de simulación u otra forma de esquematización matemática.

Esta tesis tiene una fase descriptiva, la consistencia de esta será encontrar características claves que influyen en la forma de desarrollo del software libre y que contribuyen a alterar de alguna forma la cooperación y la tasa de deserción en estos proyectos[22].

Esta tesis intentara una fase explicativa de las causas de las situaciones de desarrollo en los proyectos de software libre, siendo los lineamientos que se produzcan argumentos que se puedan comprobar para algunos casos representativos de proyectos de producción de software libre, causas y efectos de los comportamientos asociados[23].

---

[22] Roberto Hernández Sampieri, Carlos Fernández Collado, Pilar Baptista Lucio, Metodología de la Investigación, 4ta Edición, México D.F, 2006, 1998,1991, Pag. 108.

[23] Roberto Hernández Sampieri, Carlos Fernández Collado, Pilar Baptista Lucio, Metodología de la Investigación, 4ta Edición, México D.F, 2006, 1998,1991, Pag. 124

2.10.2 Método de muestreo

En el presente trabajo no se realizara muestreo de datos, se efectuara un estudio histórico sobre las cantidades de aportaciones contra los periodos de tiempo de vida en los proyectos de software libre, se analizaran variables importantes que influyen en la calidad de cooperación traducidas como volumen de aportaciones y la tasa de deserción de miembros cooperantes en los proyectos software libre. Exponiéndose las condiciones particulares que influyen en el presente estudio como genero de producto de software, versiones asociadas y situaciones sincrónicas de las actividades.

2.10.3 Prueba de la hipótesis

En esta tesis se pretenden establecer lineamientos que mengüen la falta de cooperación que se pueda presentar en los proyectos de software libre, aunque siendo el estudio de tipo cualitativo se pueden dar a conocer casos de estudio en los cuales los lineamientos propuestos tengan una importancia visible en la superación de estas problemáticas de tal modo se pueda efectuar futuramente una validación más rigurosa por medio de herramientas cuantitativas, como modelos matemáticos formales y simulaciones por computador.

2.11 Bibliografía propuesta

En la siguiente tabla se expone la bibliografía principal para esta revisión literaria con su correspondiente justificación.

Titulo	Tipo	Autor	Pertinencia
Cultures and organizations, Software of the mind	Libro	Geert Hofstede	Analiza la cooperación en la interculturalidad
Por qué el software no debe tener propietarios	Articulo	Richard Stallman	Explica la creación de software libre como la necesidad del usuario a cooperar para su propia satisfacción
Constructo para la evaluación de la cooperación en dilemas sociales de gran escala	Tesis Doctoral	Jorge Andrick Parra Valencia	Organización de Metodología para analizar dilemas sociales de gran escala, aplicable al situaciones del software libre
Escrituras digitales: Tecnologías de la creación en la era virtual	Libro, Texto electrónico	Virgilio Tortosa	Analiza diferentes aspectos de la nueva integración global
Cooperación y producción inmaterial en el software libre. Elementos para una lectura política del fenómeno GNU/Linux	Articulo	Aris Papathéodorou, Laurent Moineau	Un recuento histórico detallado del inicio del software libre como resultado de la estrecha colaboración por desarrollar software no privado
Software libre para una sociedad libre	Libro, Texto electrónico	Richard M. Stallman	Justificaciones de porque el software debe ser de libre y producto de la cooperación
Producing Open Source Software: How to Run a Successful Free Software Project	Libro	KarlFogel	Texto que explica de forma técnica la organización y requisitos de

			proyectos de software libre
How Open is the Future? Economic, Social & Cultural Scenarios inspired by Free and Open Source Software	Libro	Marleen Wynants & Jan Cornelis (Eds)	Algunas aplicaciones prácticas reales de la filosofía del software libre
Dominio abierto. Conocimiento libre y cooperación	Libro	Igor Sádaba (Ed.)	Análisis desde diferentes aspectos del crecimiento de la cultura de conocimiento compartido y las redes
SOCIAL DILEMMAS: The Anatomy of Cooperation	Articulo	Peter Kollock	Explica algunos modelos de dilemas en cooperación y detalla soluciones
Software libre	Libro	Jesús M. González Barahona Joaquín Seoane Pascual Gregorio Robles	Tratado sobre software libre, recuento histórico, técnico entre otros

Tabla 1 – Descripción bibliográfica

2.12 Criterios de selección de casos representativos

En este trabajo se estudian casos representativos del software libre, ya que para un estudio general con definición de variables estadísticas, se tendrían líneas de desarrollo algo diferentes y que de una u otra forma intervienen en el rendimiento y comportamiento de los individuos en los proyectos de software libre, siendo

comunidad de desarrolladores de software libre un conjunto no homogéneo de personas (en ocupación, motivaciones y geográficamente[24]).

Para hacer de este estudio una forma realmente efectiva de análisis de la cooperación en el software libre se ha pensado como parte de la metodología exponer en proyectos, que tienen características claves, las dinámicas de desarrollo que pueden ser mejoradas para lograr una tasa de deserción baja y un mejor grado de cooperación en casos en los cuales esta vaya en deterioro.

Los principales criterios se han escogido como puntos necesarios para validar la aplicabilidad y la fiabilidad de los lineamientos a plantear, siendo que forman por su trayectoria un ejemplo modelo de la filosofía del software libre. Así de esta manera como principales criterios tendríamos en la búsqueda de un tema para estudio de cualquier interés particular, así tenemos en este estudio los siguientes criterios:

Tiempo de vida del proyecto (antigüedad)**:** En vista de la fugacidad con que el ser humano se integra en un grupo social, de retira, vuelve nuevamente o definitivamente se retira[25]. El ejemplo de *la radio como gran instrumento de difusión comunicativa demostrado con el tiempo*, muestre lo imprescindible de examinar con ejemplos de amplia antigüedad[26] el caso del software libre.

---

[24] Ver Jesús M. G., Joaquín S. P. y Gregorio R., Software libre, página 62

[25] Ver Geert Hofstede., Software libre, Cultures and organizations, Software of the mind, pagina 17

[26] Ver Richard Stallman, Por qué el software no debe tener propietarios, pagina 3.

Cantidad de participantes: Se propondrán casos representativos en los cuales el número de integrantes no sea fácilmente reproducible en un experimento real, ya que algunos proyectos pueden tener integrantes titulares que formen parte de una cúpula reconocida, pero en realidad su membresía no es exactamente cuantificable, ni controlable[27].

***Reconocimiento del proyecto*:** Aunque no es de negar que existen programas de software libre que llegan a funcionar de unas maneras más complejas que otros tantos, y hasta requieren un nivel de ingenio muy superior. También es cierto que en el mundo del software libre existen proyectos bandera más por su popularidad que por su complejidad. Y dado que la escogencia de un caso representativo implica ser un símbolo del software libre como un todo, la popularidad del proyecto será tenida en cuenta como factor de importancia

[27] Ver Jesús M. G., Joaquín S. P. y Gregorio R., Software libre, página 145 y 151

3. Estudio de casos

Los casos de estudio a tener en cuenta son ampliamente reconocidos no solo en el mundo del software libre, también en ambientes empresariales y más tempranamente en ambientes caseros. Esto asegura que la búsqueda de características lo suficientemente representativas de los proyectos de desarrollo de software libre sea alcanzada en este estudio. No obstante la funcionalidad de estos programas, variadas formas de desarrollo y de organización los diferencia, la forma económica de sostenimiento puede ser también unas diferencias pero no necesariamente; En el cuadro siguiente se desea proponer siete casos de estudio

	Descripción	**Año de inicio**	**Volumen de participación**	**Reconocimiento del proyecto**
kernel LINUX	Núcleo primario del sistema operativo Linux y todas sus distribuciones	1991	Más de 3700 desarrolladores, más de 6 millones de usuarios. Fuente: https://github.com/torvalds/linux	Muy alto en la comunidad de software libre, alto en empresas y bajo en el usuario casero.
Berkeley Software Distribution (BSD)	Sistemas operativos y distribuciones basadas en las primeras versiones de UNIX elaboradas es Universidad de	1999	Auspiciado por más de 482 donantes y grandes empresas de mercado del internet Fuente: página principal	Bueno en grandes empresas, muy bajo en el ambiente del software libre y

	California en Berkeley.			usuario casero
K Desktop Environment (GENTOO / KDE)	Paquete de escritorio multipropósito, ampliamente personalizable.	1996	65 colaboradores, Más de 60 traductores al español, más de mil en otros idiomas. Tomado de http://es.l10n.kde.org/colaboradores.php	Muy alto en las comunidades de software libre, casi desconocido en otros ambientes
Joomla Content Management System	Sistema para gestión de contenidos con numerosas funcionalidades adicionales.	2005	266 Desarrolladores, fuente: https://github.com/joomla/joomla-cms Usado en más de 112 idiomas	Alto en la comunidad de software libre, alto en las empresas relacionadas al web. Bajo en el usuario común.
PhpMyAdmin	Manejador Web de bases de datos orientado a la administración del motor MySQL	1998	425 colaboradores según https://github.com/phpmyadmin/phpmyadmin	Alta en el ámbito del desarrollo web y manejo de bases de datos
Ruby on Rails	Framework para desarrollo de aplicaciones web que sigue el modelo vista-controlador	2005	2544 colaboradores según https://github.com/rails/rails más de 3500 según su web oficial http://rubyonrails.org	Ampliamente reconocido en el ámbito de aplicaciones web de tipo GNU

Tabla 2 – Descripción proyectos casos de estudio

A continuación se expondrán los casos de estudio seleccionados, con datos estadísticos que muestran la interacción de los miembros con el proyecto, cuestión que ayudara a descifran varias interacciones comunes que involucra a los miembros del proyecto.

3.1 Caso de estudio kernel GNU/Linux

En este grafico se puede observar el número de aportaciones de los miembros del proyecto a la rama principal del mismo contra el paso del tiempo en el último año

Figura 4 – Tiempo de vida GNU/Linux vs Cantidad de Commits

Dividiendo el grafico en 3 partes, desde antes del año 2013, durante todo el periodo 2013, y en el periodo que inicia en 2014, encontramos que existen una depresiones marcada en la fecha 2013 y fecha 2014, esto puede deberse al cambio de año y el receso por el ánimo de varias festividades de cambio de año, también a algún factor temporal existente en común para los colaboradores por estos periodos.

Otra observación en todo el grafico es que las aportaciones de los miembros se mantienen en todo momento, podemos citar un promedio de aportaciones de algo más de 200 diarias en cualquier momento, lo que nos conduce a la pregunta de si la masa de miembros genera tal grado de aportaciones, entonces ¿puede ser el mismo grado de trabajo o de contacto para los miembros, o equivalente?, ahora bien, en las siguientes graficas se pueden observar varios casos de individuos y sus aportaciones.

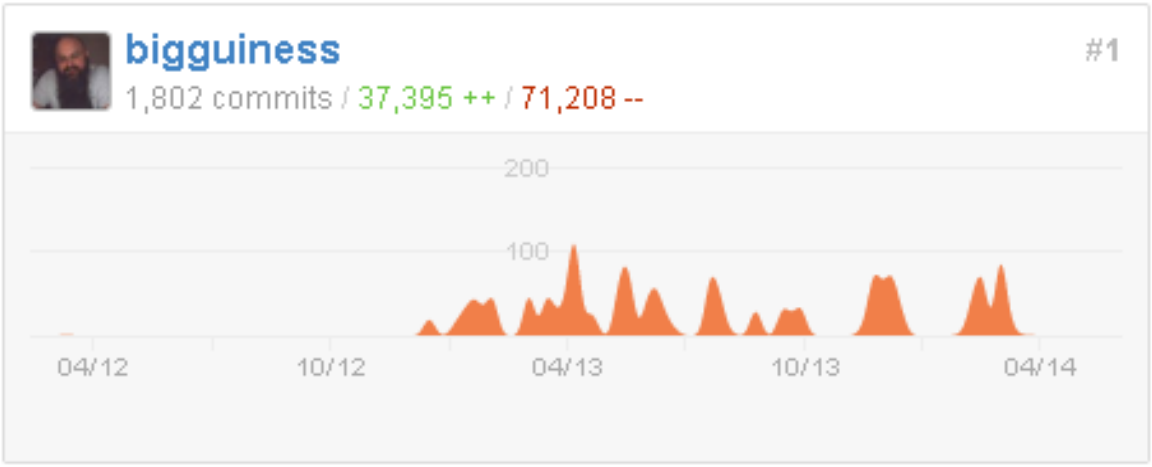

Figura 5 – Tiempo de vida GNU/Linux vs Cantidad de Commits desarrollador #1

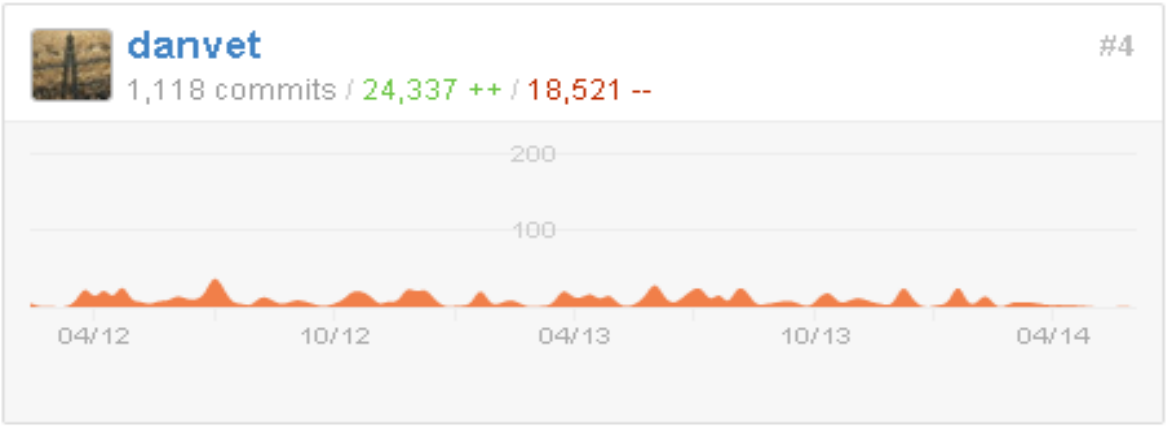

Figura 6 – Tiempo de vida GNU/Linux vs Cantidad de Commits desarrollador #4

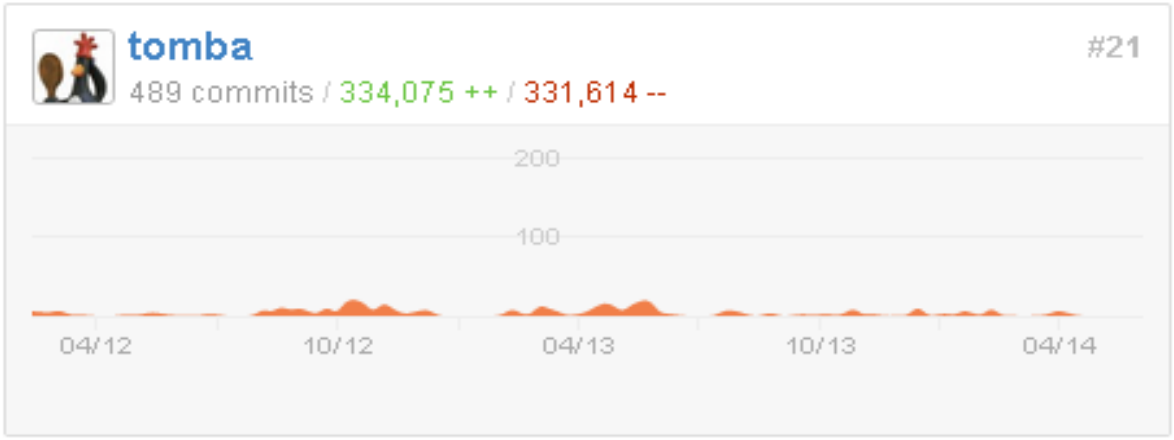

Figura 7 – Tiempo de vida GNU/Linux vs Cantidad de Commits desarrollador #21

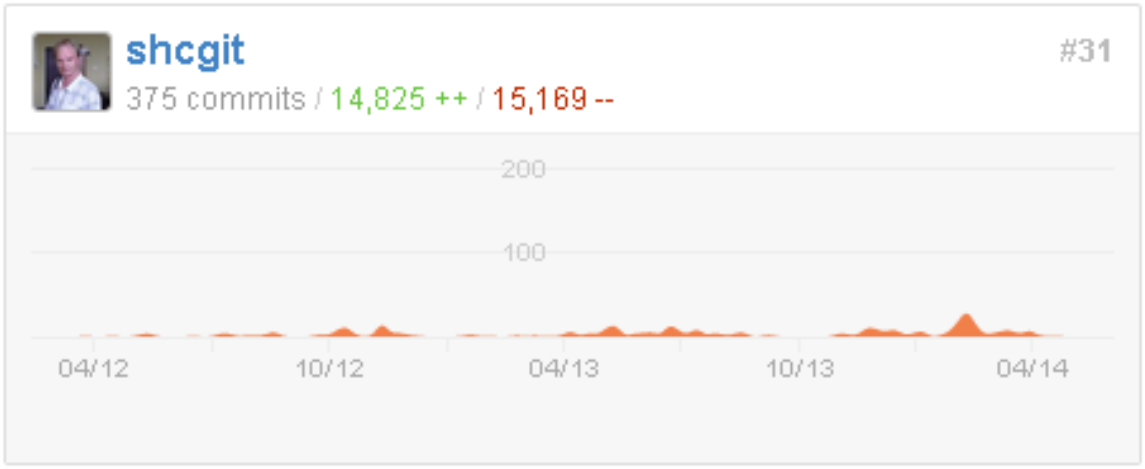

Figura 8 – Tiempo de vida GNU/Linux vs Cantidad de Commits desarrollador #31

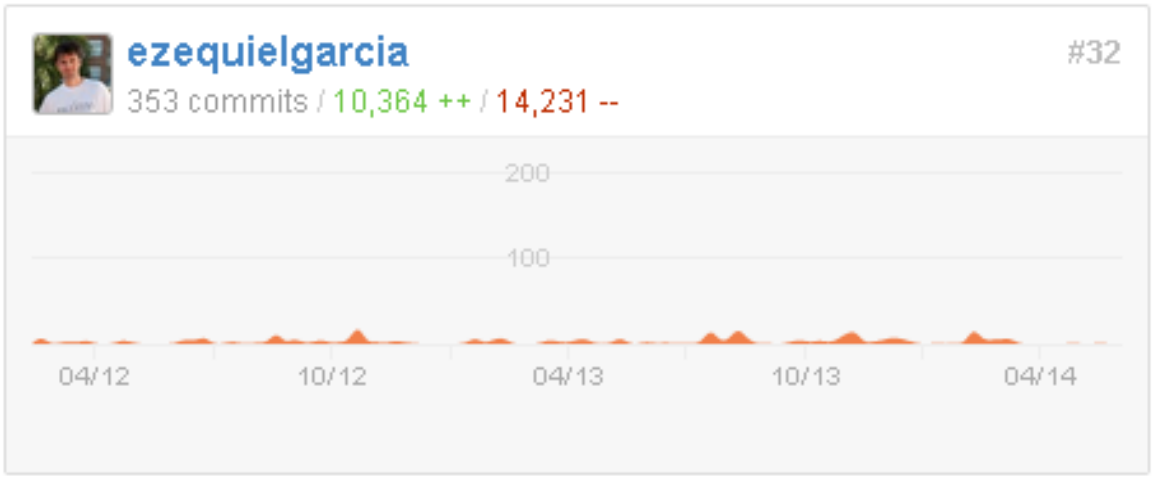

Figura 9 – Tiempo de vida GNU/Linux vs Cantidad de Commits desarrollador #32

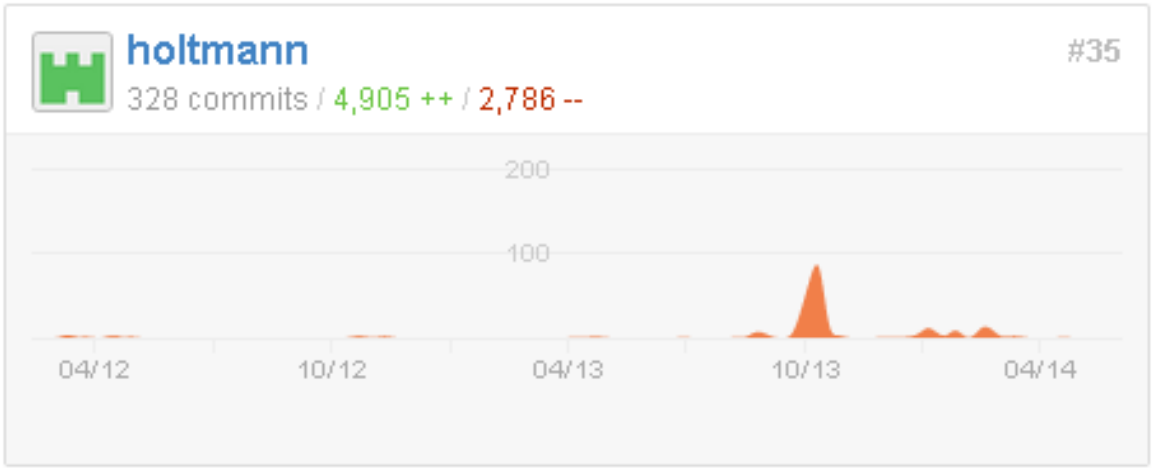

Figura 10 – Tiempo de vida GNU/Linux vs Cantidad de Commits desarrollador #35

Algunas observaciones de los cuadros de participación de los usuarios más activos del proyecto Linux son:

- Aunque el volumen de aportaciones de los usuarios al proyecto es diferente y varia en razón de variables personales, que no son objeto del presente estudio, tampoco de los datos que se pueden obtener, el hecho a simple vista verificable es que *estas personas de una forma u otra están ligadas a la comunidad pues mantienen en todo el tiempo la actividad en el proyecto, sin que sea su periodicidad la misma en todos los casos*, o independiente que la cantidad de aportaciones sea casi cero (0) en algunos momentos.

- Si bien los picos de aportaciones son raros son más comunes las aportaciones constantes y sostenidas a través de cortos periodos de tiempo, lo que se puede traducir como el planteamiento y alcance de pequeñas tareas es un ejemplo de los mejores contribuyentes al proyecto Linux.

- Se puede contrastar el caso de estudio Linux para la situación de commits efectuados por los miembros con la situación de adiciones de nuevo código que está orientada a la generación de nuevas soluciones o características del software en desarrollo en este caso el kernel Gnu/Linux, la siguiente grafica muestra las operaciones de adición de código al proyecto.

Figura 11 – Tiempo de vida GNU/Linux vs Cantidad de Adiciones

Se puede apreciar que los valles de la gráfica oscilan entre 0 y 20k lo que significa que las adiciones al código fuente son normalmente de 20.000 operaciones por día, lo que supone una estrecha forma de colaborar entre los miembros para estar al ritmo de desarrollo del proyecto, lo que les permitiría observar de una forma u otra los aportes producidos en el proyecto, ya sean adicionados al código de la rama principal o queden en espera.

En el gráfico de operaciones de eliminación se puede apreciar la relación directa que guarda el volumen de operaciones de borrado con el volumen de operaciones de adición al código fuente de la rama principal, es visible que los picos y los valles casi se corresponden de manera igual en las operaciones de eliminación y de adición de código.

Figura 12 – Tiempo de vida GNU/Linux vs Cantidad de Eliminaciones

El comportamiento más visible en la cooperación interna de desarrollo de este proyecto es deducible observando que aunque las gráficas estadísticas anteriores tienen algunos picos pronunciados, poseen muchos valles de forma constante, sin que ninguno de estos llegue a un punto cero (0), lo que significa que aunque el proyecto tenga unos lapsos mínimos de cooperación entre los miembros, *estos miembros se mantienen activos dentro de la comunidad del proyecto disminuyendo su cantidad de trabajo o el número de aportaciones que producen, pero estando ligados a la rama o tareas que desarrollan*, a través de las siguientes graficas se muestra la actividad de los miembros más activos dentro del proyecto para cada una de la operaciones principales, situación de eliminación y situación de adición de código.

3.1.1 Algunos historiales de aportación en situación adicionar código

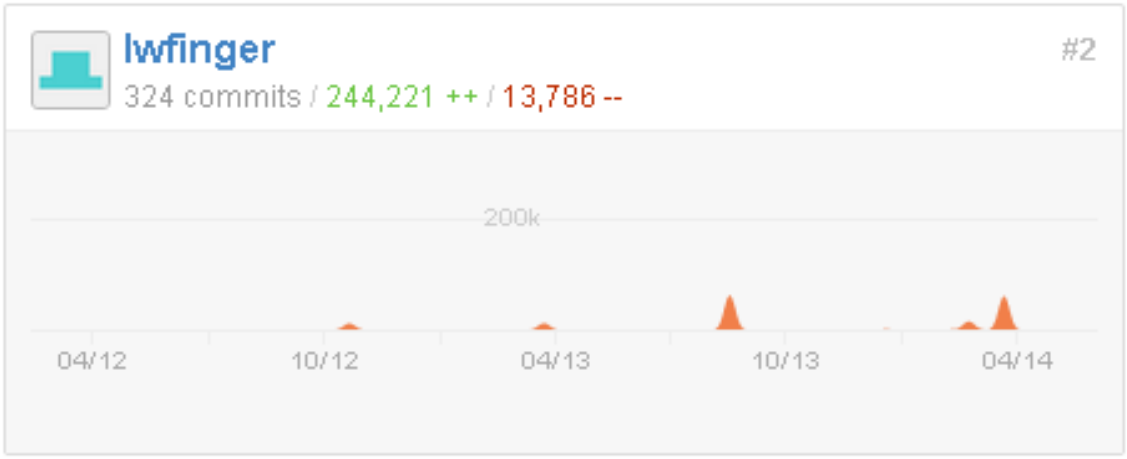

Figura 13 – Tiempo de vida GNU/Linux vs Cantidad de Eliminaciones desarrollador #2

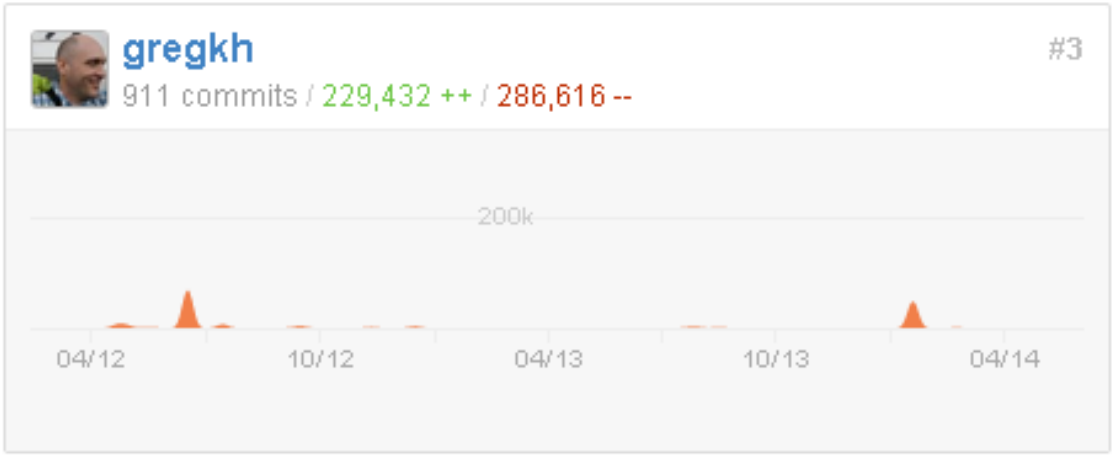

Figura 14 – Tiempo de vida GNU/Linux vs Cantidad de Eliminaciones desarrollador #3

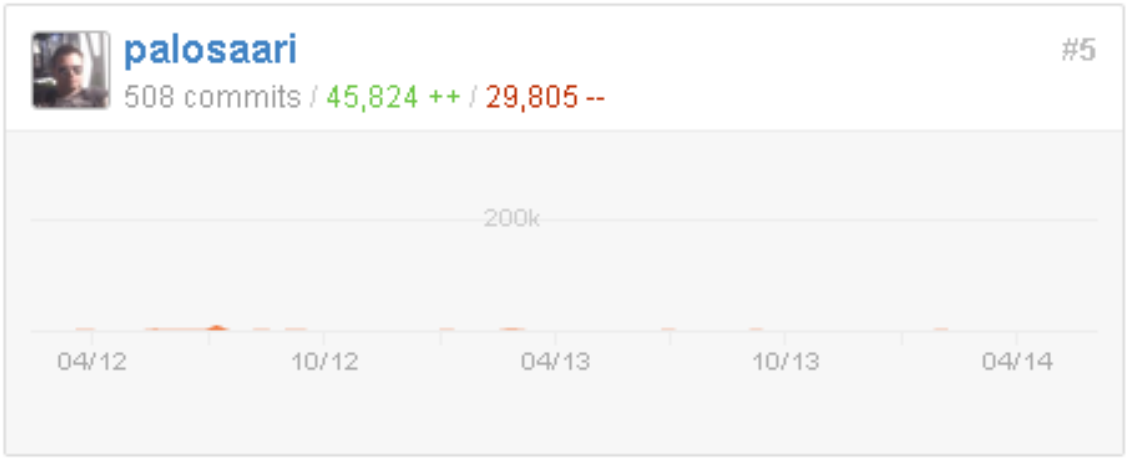

Figura 15 – Tiempo de vida GNU/Linux vs Cantidad de Eliminaciones desarrollador #5

Figura 16 – Tiempo de vida GNU/Linux vs Cantidad de Eliminaciones desarrollador #7

3.1.2 Algunos historiales de aportación en situación operación commit

Figura 17 – Tiempo de vida GNU/Linux vs Cantidad de commits

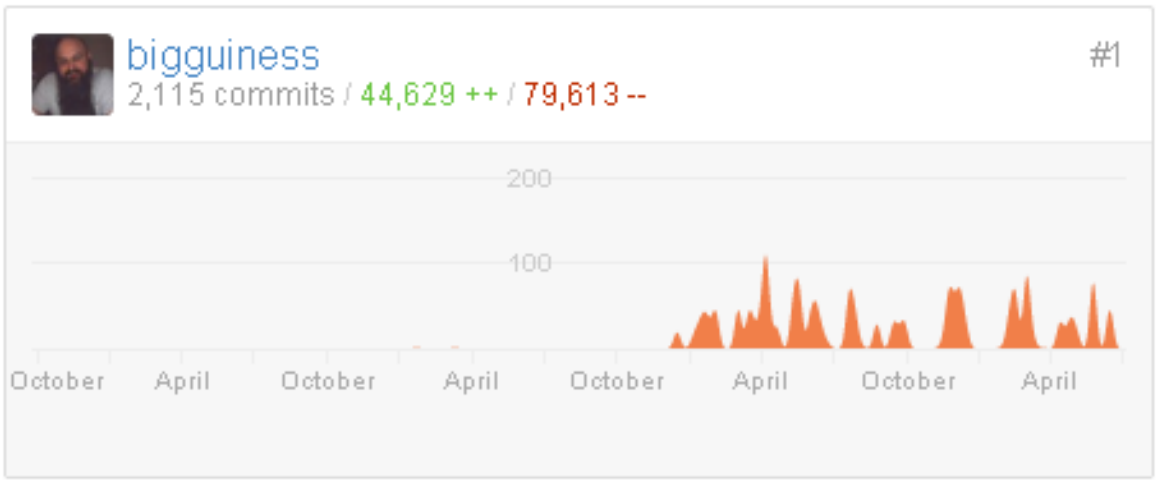

Figura 18 – Tiempo de vida GNU/Linux vs Cantidad de commits desarrollador #1

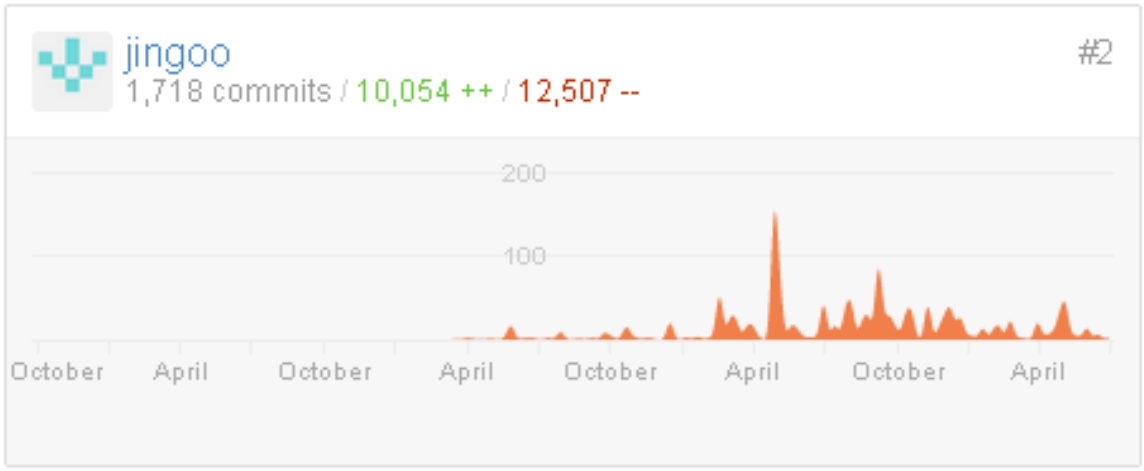

Figura 19 – Tiempo de vida GNU/Linux vs Cantidad de commits desarrollador #2

Figura 20 – Tiempo de vida GNU/Linux vs Cantidad de commits desarrollador #3

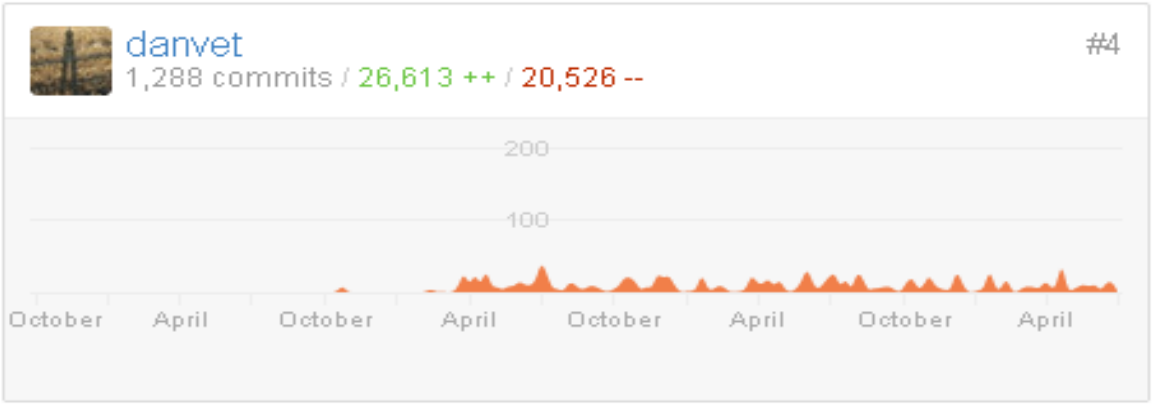

Figura 21 – Tiempo de vida GNU/Linux vs Cantidad de commits desarrollador #4

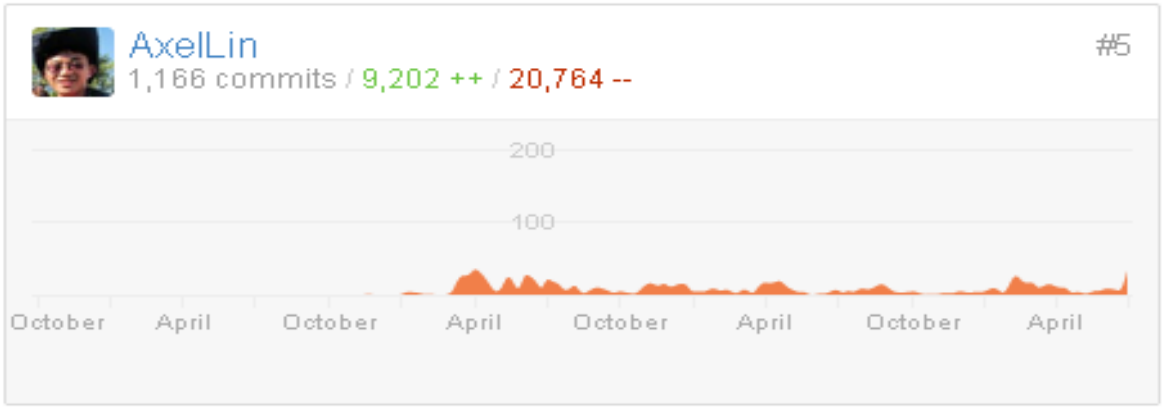

Figura 22 – Tiempo de vida GNU/Linux vs Cantidad de commits desarrollador #5

Observando las operaciones commits de los miembros con más aportaciones en el periodo 2013 y primer semestre de 2014 es observable que el ritmo de aportaciones es constante en los miembros, de forma diaria lo que hace el modo de producción muy constante y da la confianza necesaria para que nuevos miembros se unan y tengan el apoyo necesario por parte de la comunidad de desarrolladores. Esta constancia también sirve para dar por sentado la

previsibilidad de versiones, de prueba de las actuales e innovaciones implementadas en cada una de las nuevas.

3.1.3 Mecanismo de cooperación caso de estudio kernel GNU/Linux

Los mecanismos presente en este caso de estudio en el cual el número de aportaciones son del orden de más de 100 diarias y la cantidad de miembros activos del proyecto superan los 3.700 se pueden catalogar como dilemas sociales de gran escala.

Mecanismo de cooperación basado en confianza: Este mecanismo es utilizado en el caso de estudio del desarrollo GNU Linux ya que los participantes tienen como premisa fundamental el hecho de que aunque sus aportes individuales son normalmente pequeños para el gran tamaño del proyecto todo la comunidad está activa[28] como aportantes y el crecimiento del proyecto día a día evidenciado en las nuevas versiones y mejoras publicadas en la página oficial www.kernel.org .

La reputación del proyecto repercute directamente en la reputación de los miembros y de la comunidad de desarrollo, este hecho permite que el pertenecer al proyecto oriente los objetivos de la comunidad hacia el mantenimiento del proyecto en altos estándares de modernidad llevando esta situación a que nuevos miembros se quieran adherir al proyecto y a que los miembros antiguos obtengan experiencia valorada altamente por ser el proyecto de software libre con más

[28] Ver Jorge Andrick Parra Valencia, Constructo para la evaluación de la cooperación en dilemas sociales de gran escala , Pág. 20

compatibilidad[29], en la actualidad como desarrolladores cooperadores del proyecto[30].

3.2 Caso de estudio distribución FreeBSD

En el caso de estudio de la distribución FreeBSD encontramos varias situaciones de cooperación durante el desarrollo, se puede ver que en el periodo 2009 a 2010 se presenta una caída vertiginosa de producción de código fuente, lo que respectivamente significa que la mayoría de desarrolladores estuvieron en inactividad en este pequeño periodo, aunque la gráfica muestra esta caída como producto de la inactividad en la realidad se corresponde con fases de prueba y análisis a la versión 8 de FreeBSD para asimilar nuevas características indispensables que darían a la luz nuevos objetivos de desarrollo. En agosto de 2009 se inicia la revisión y en noviembre 25 del mismo año se da por terminada la primera versión de producción de la serie 8.

[29] Ver Robert Love 2010, Linux Kernel Development, Pag. 3

[30] Ver Jorge Andrick Parra Valencia, Constructo para la evaluación de la cooperación en dilemas sociales de gran escala , Pág. 21

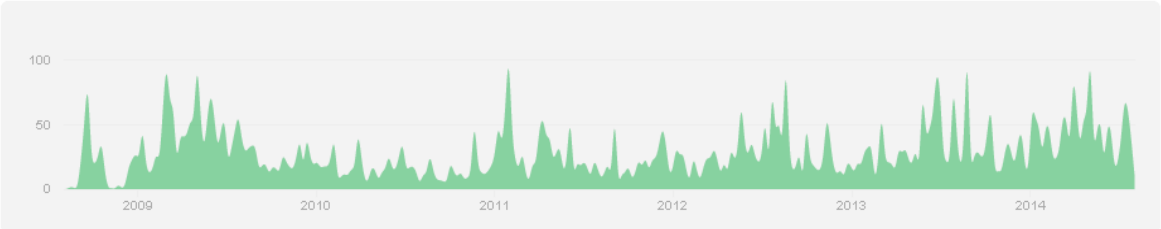

Figura 23 – Tiempo de vida FreeBSD vs Cantidad de commits

En este caso de estudio se puede apreciar en el periodo anterior al inicio de 2010 la gran cantidad de commits realizados durante mediados de 2009 y en notable decaída hasta el 2010, siendo este comportamiento la representación del preámbulo a la versión KDE 4.4 RC 1 del 6 de enero de 2010, lo que explica que la gran actividad se debe a la necesidad de tener a punto variadas características mayores del software. Durante el periodo 2010-2011 se nota una relativa constante y ningún pico suficientemente sobresaliente en este periodo, es notorio este comportamiento ya que en este periodo no hubo una carga de trabajo lo suficientemente grande como la que venía desde 2008 cuando la versión 4 fue reescrita y por lo tanto en el periodo 2009-2010 se realiza gran esfuerzo en mejoras funcionales, adiciones y corrección de errores; en 2010 se realiza la versión 4.5 pero como cambio mayor se presentan grandes mejoras en corrección de errores. A finales de 2010 se inicia la construcción de KDE SC 4.6 RC 1 , hacia el 22 de diciembre lo que muestra un pico mediano a final de 2010 y un gran pico de desarrollo en principio de 2011, la versión KDE SC 4.6 final fue lanzada el 25 de Enero de 2011.

3.2.1 Algunos historiales en caso adicionar código:

En el caso de FreeBSD es visible que alta participación de los miembros se evidencia no en la mayoría sino más bien en una minoría bastante pronunciada ya que en realidad los participantes en aportaciones no alcanzan a efectuar aportaciones como la minoría del proyecto, se puede observar en el comportamiento de adiciones comprendidos entre Marzo 20 de 2007 a Julio 26 de 2007 que la media de aportaciones estuvo algo más de una por miembro, así:

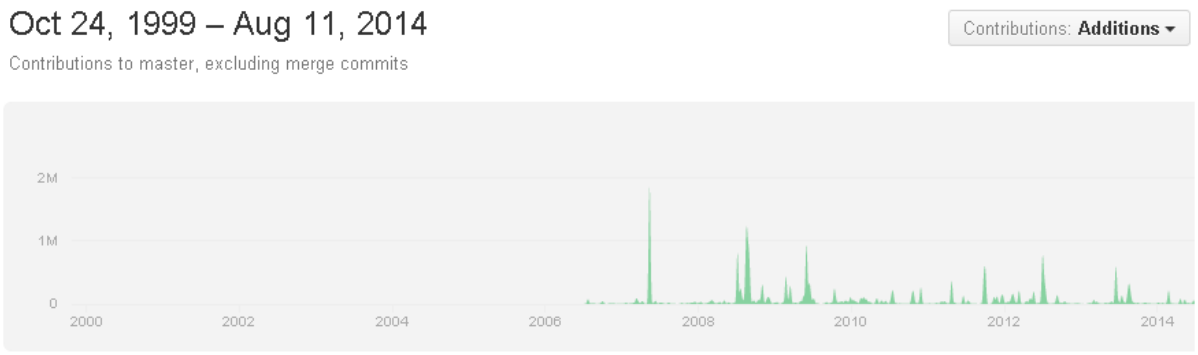

Figura 24 – Tiempo de vida FreeBSD vs Cantidad de adiciones

Las siguientes son las gráficas de los miembros con más aportaciones durante el periodo Marzo 20 de 2007 a Julio 26 de 2007

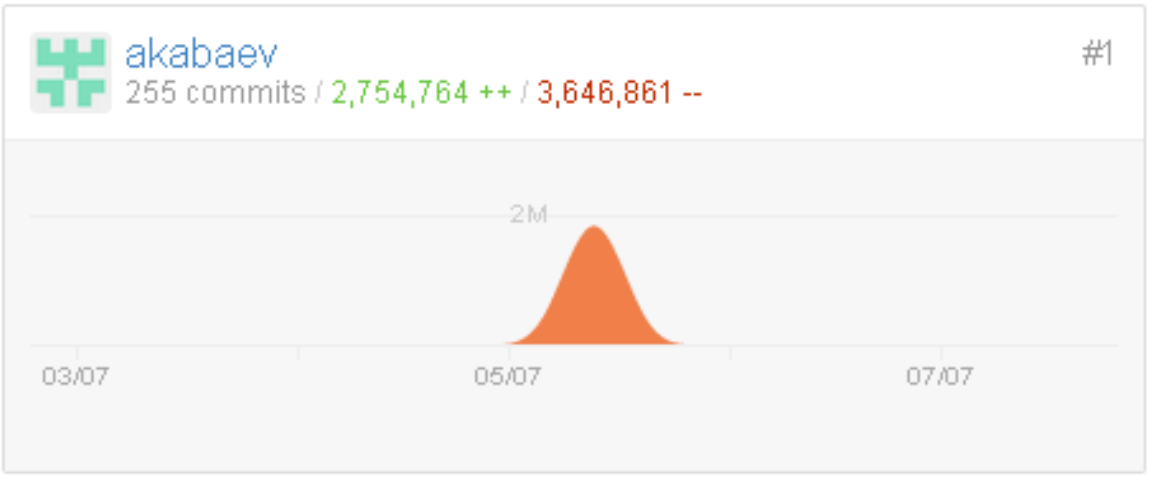

Figura 25 – Tiempo de vida FreeBSD vs Cantidad de adiciones desarrollador #1

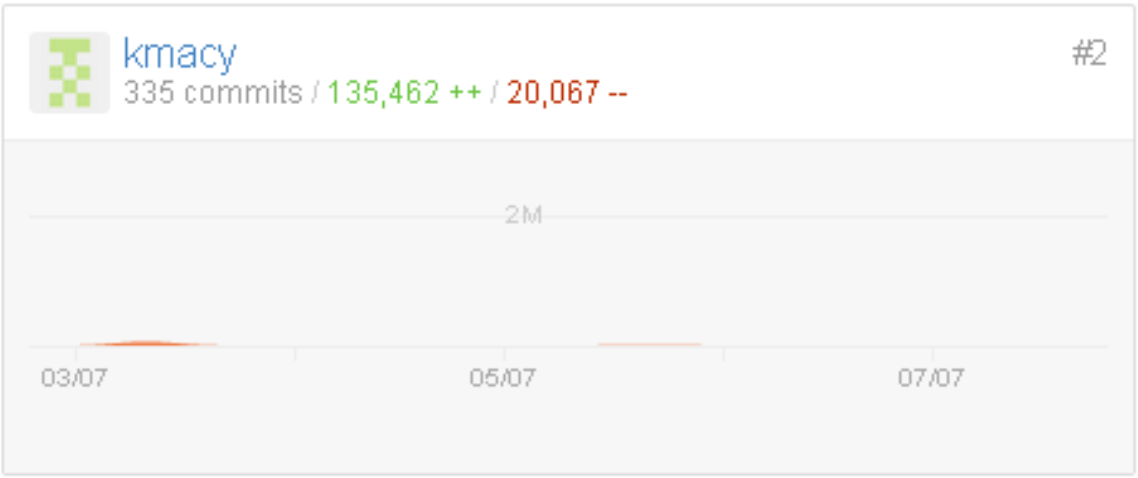

Figura 26 – Tiempo de vida FreeBSD vs Cantidad de adiciones desarrollador #2

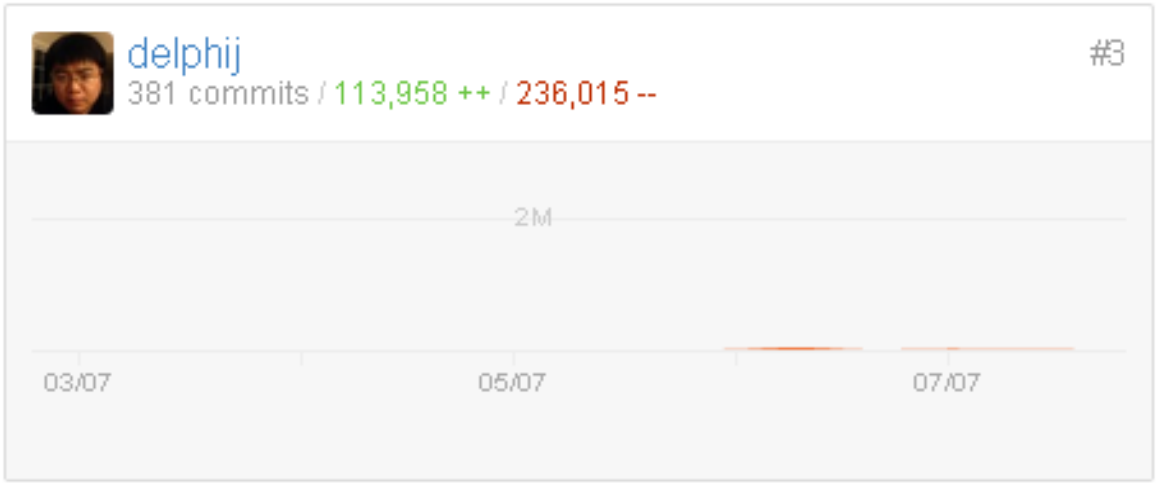

Figura 27 – Tiempo de vida FreeBSD vs Cantidad de adiciones desarrollador #3

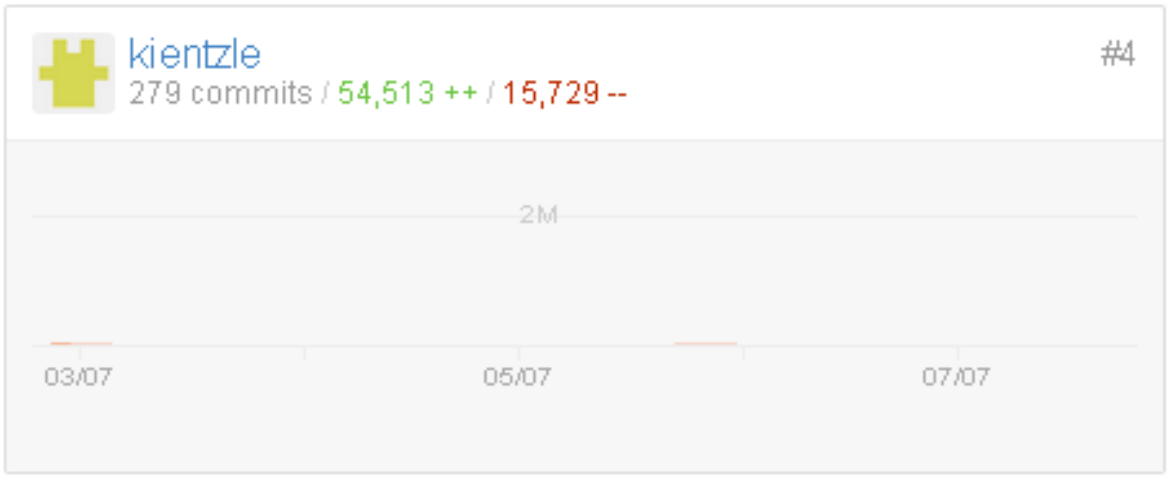

Figura 28 – Tiempo de vida FreeBSD vs Cantidad de adiciones desarrollador #4

Lo anterior deja de manifiesto que para el caso FreeBSD el estudio de caso debe ser con periodos más largos, porque en periodos cortos las aportaciones de los miembros tienden a ser inconstantes y poco frecuentes, ahora bien si el caso es

observado en un periodo de dos o tres años las aportaciones en periodos de tiempo largos (meses) tienden a mostrar un nivel constante de trabajo, así:

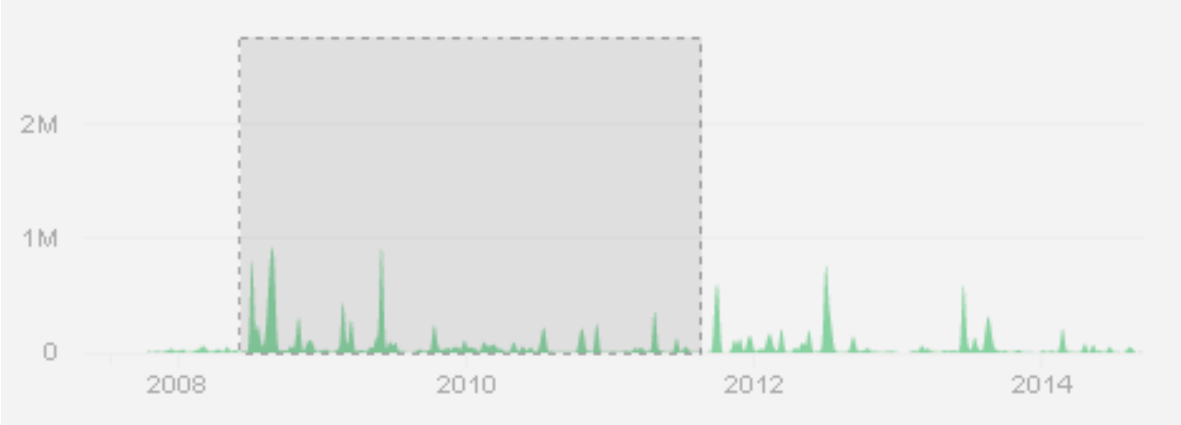

Figura 29 – Tiempo de vida FreeBSD vs Cantidad de adiciones desde Junio 3 de 2008 a Agosto 18 de 2011

Graficas de actividad de los miembros con más aportaciones durante el periodo:

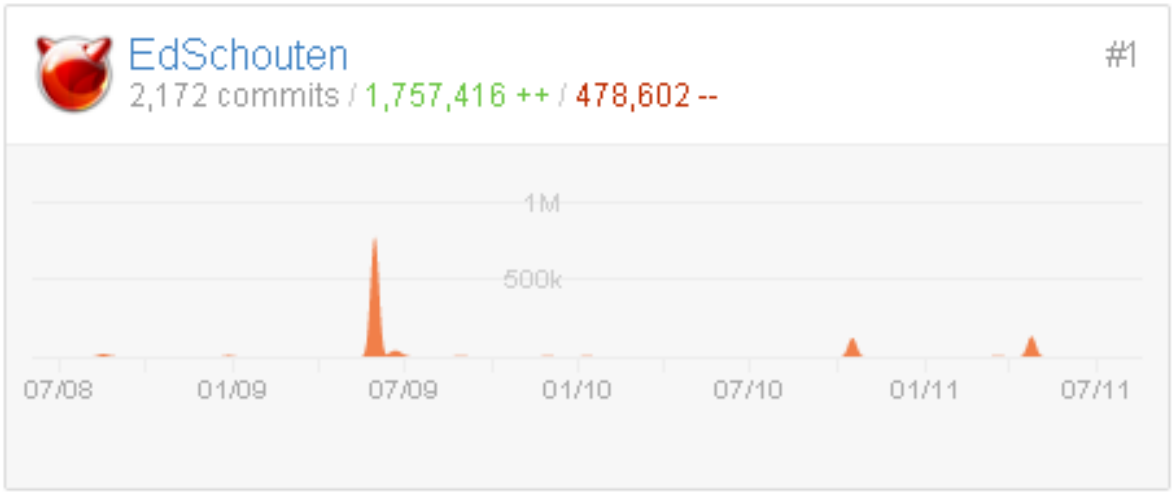

Figura 30 – Tiempo de vida FreeBSD vs Cantidad de adiciones desarrollador #1

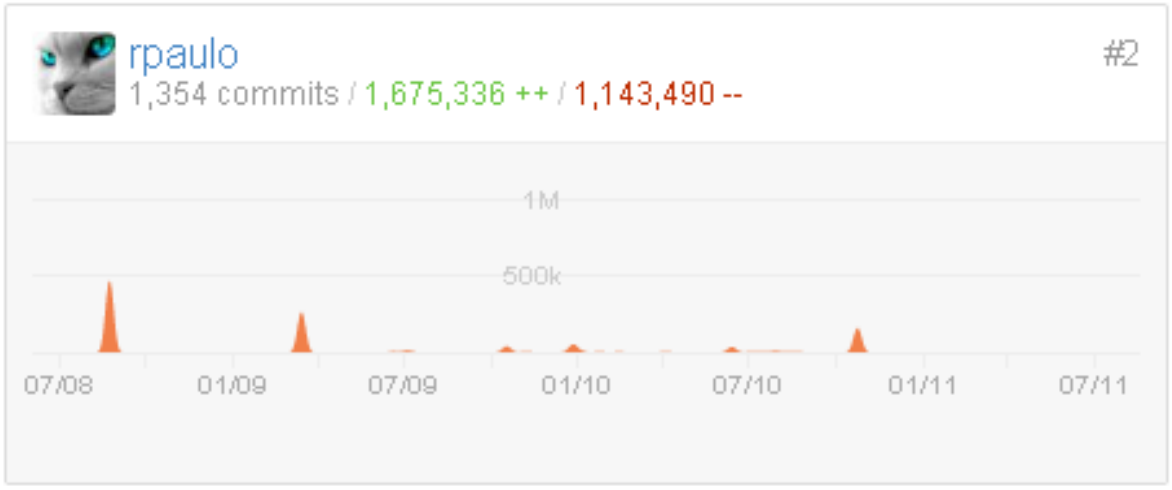

Figura 31 – Tiempo de vida FreeBSD vs Cantidad de adiciones desarrollador #2

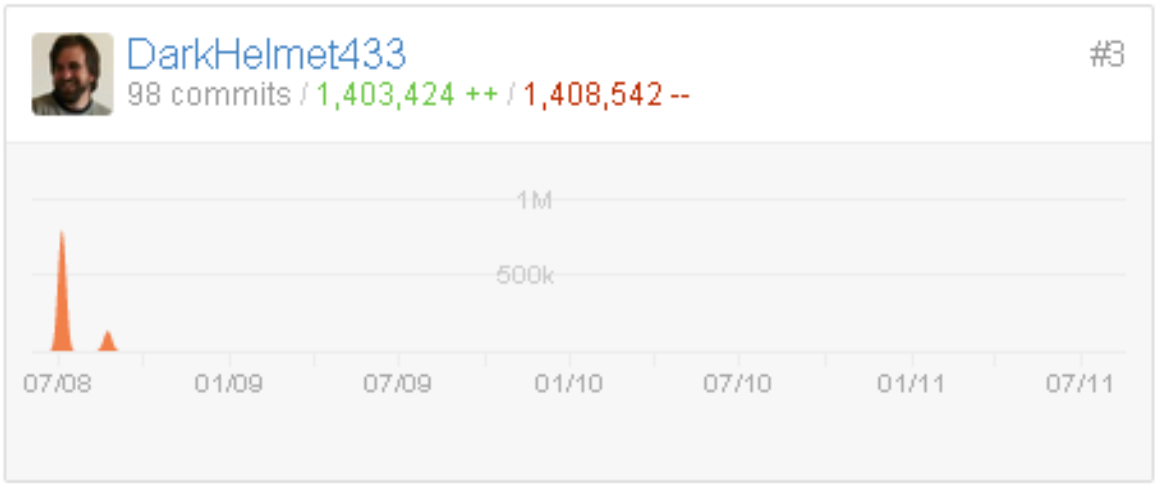

Figura 32 – Tiempo de vida FreeBSD vs Cantidad de adiciones desarrollador #3

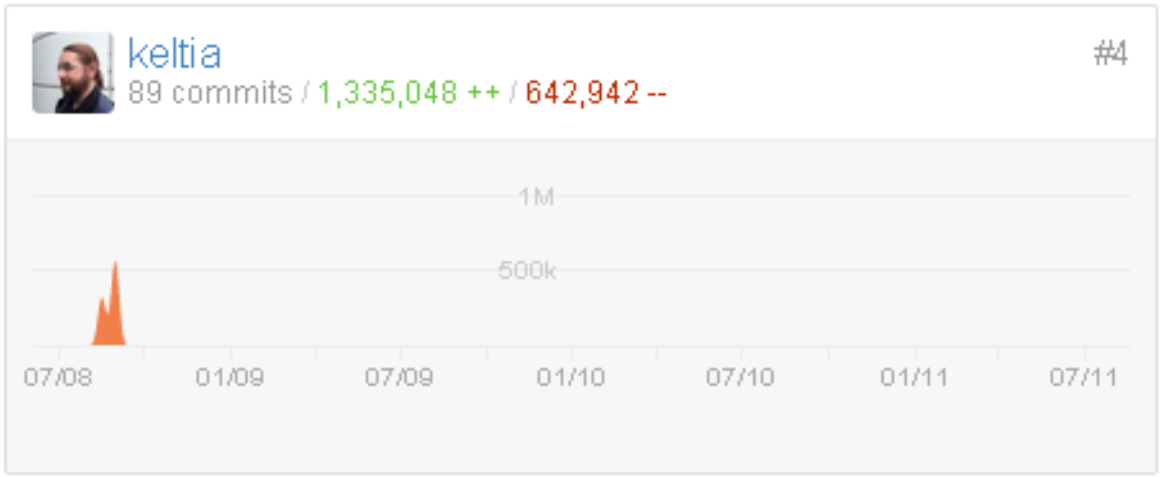

Figura 33 – Tiempo de vida FreeBSD vs Cantidad de adiciones desarrollador #4

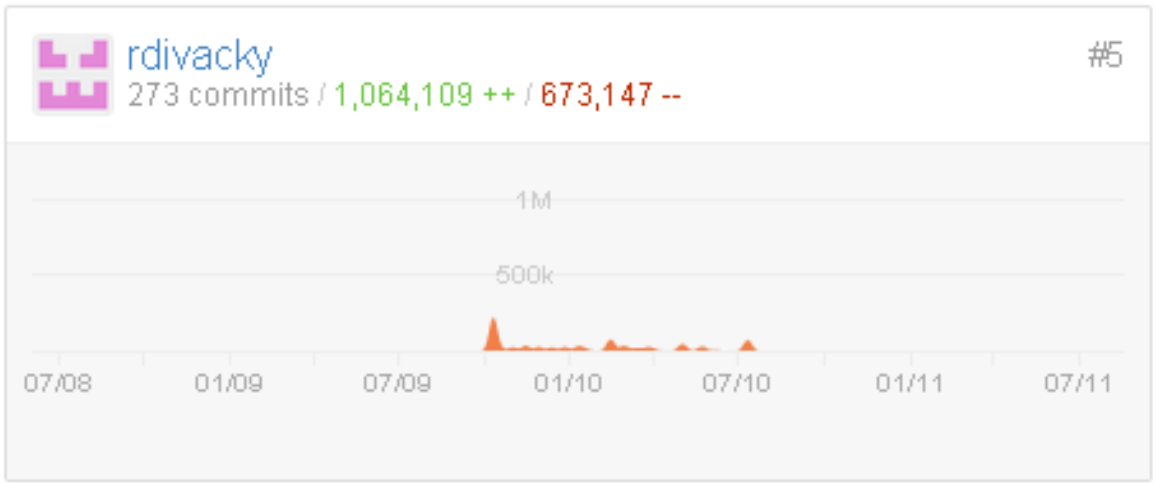

Figura 34 – Tiempo de vida FreeBSD vs Cantidad de adiciones desarrollador #5

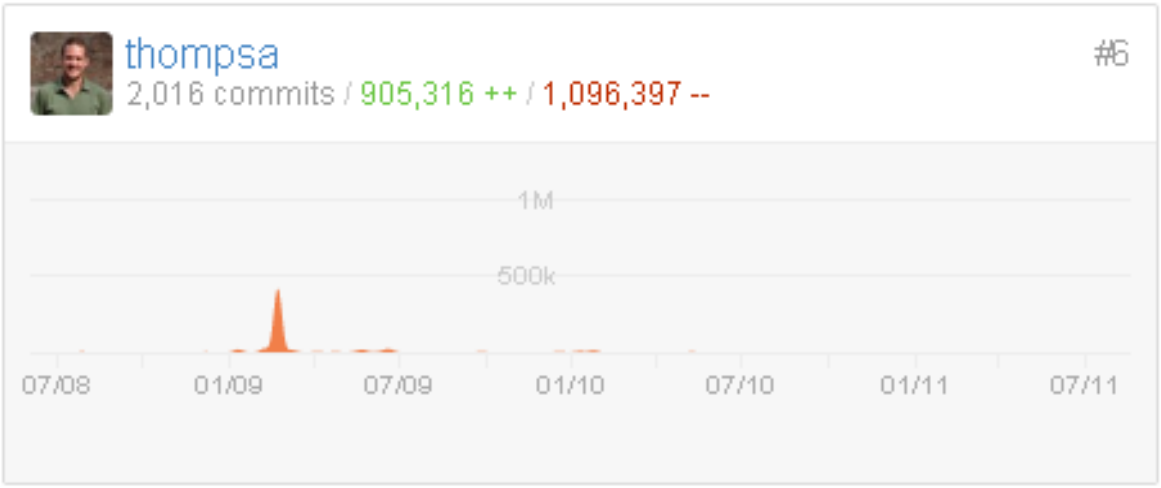

Figura 35 – Tiempo de vida FreeBSD vs Cantidad de adiciones desarrollador #6

Se puede apreciar en las gráficas que los miembros con más aportaciones en el proyecto FreeBSD presentan picos elevados de aportaciones nuevas de código, lo que puede significas actualizaciones importantes de la versión en la cual se trabaja o también adición de nuevas funcionalidades o características. Es de notar que en el caso en particular de FreeBSD los picos de trabajo de cada uno de los miembros con más aportaciones en el proyecto no se cruzan, lo que significa que de alguna forma sus trabajos son no relacionados, fueron secuenciales en el periodo estudiado o simplemente excluyente.

3.2.2 Algunas aportaciones de operaciones commit

Examinamos el caso FreeBSD para operaciones commit desde Febrero 3, 2006 a Septiembre 30, 2014, en el cual se observa una constante en operaciones commit, también examinando esta operación para los miembros con más aportaciones se

puede ver que existe un ritmo casi constante y con pocos cortes en ambos casos, tanto en la totalidad grupal del proyecto como en los miembros individualmente.

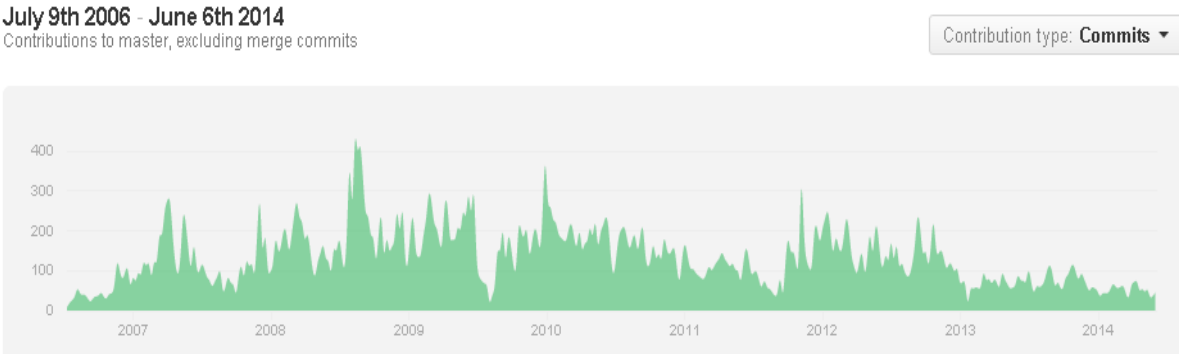

Figura 36 – Tiempo de vida FreeBSD vs Cantidad de commits

Analizando las colaboraciones de los desarrolladores que están primeros en el ranking de aportaciones se puede que estas personas no se dedican únicamente a una gran tarea de desarrollo continuado, y es lo que muestran sus graficas de interacción con el sistema de versiones, cuando se profundiza en el historial de comportamiento es observable que el número de aportaciones es tan grande no en razón de que el aporte sea de gran tamaño, sino más bien porque cada persona efectúa un gran número de pequeños aportes a muchas tareas, es decir que de esta forma el usuario programador hace muchos pequeños cambios al sistema pero de una forma gradual, lo que hace visible su trabajo y posiblemente retroalimentado desde la visión de la limitación de las tareas efectuadas.

Es notable que en el periodo 2009 a 2010 se corresponde en los gráficos de usuarios programadores como en el gráfico de histograma de **commits,** una

especie de receso, en los usuarios se puede apreciar que aunque no siendo concluyente que este lapso de quietud en el desarrollo de código afecte de forma continua el proyecto, si se puede afirmar que por alguna razón, algunos usuarios parecen demorarse un poco más en recobrar cierto ritmo de interacción. Es en este análisis que se puede decir que *los recesos prolongados pueden afectar el ritmo de trabajo en los proyectos de software libre.*

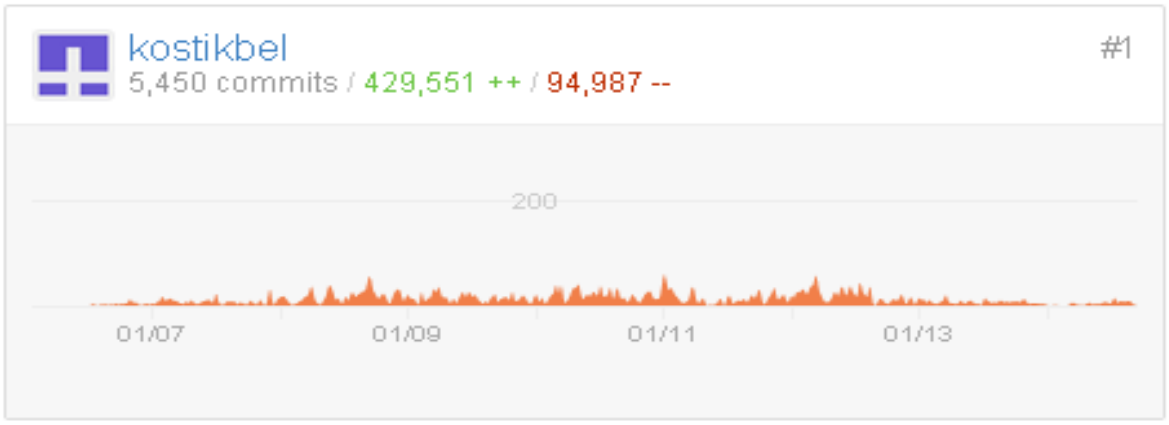

Figura 37 – Tiempo de vida FreeBSD vs Cantidad de commits desarrollador #1

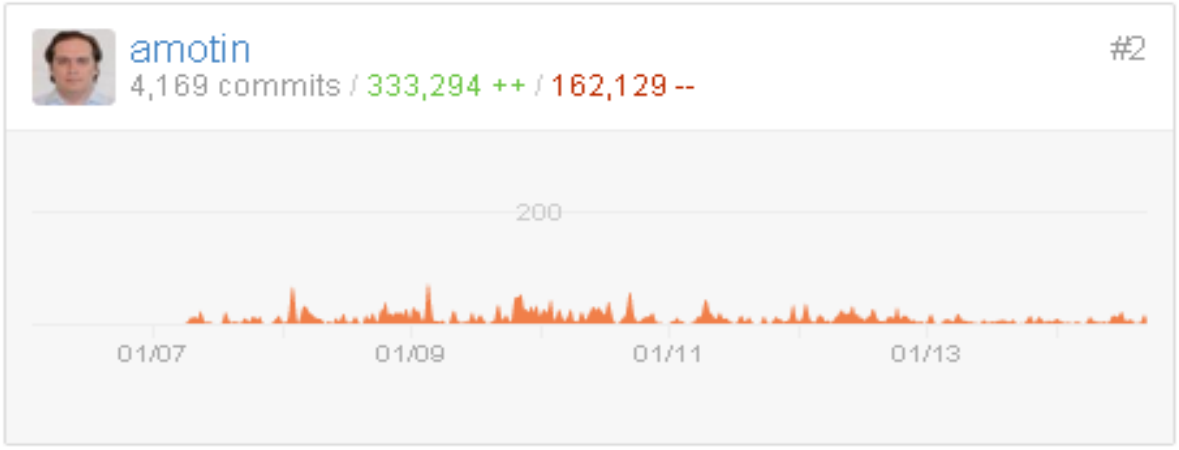

Figura 38 – Tiempo de vida FreeBSD vs Cantidad de commits desarrollador #2

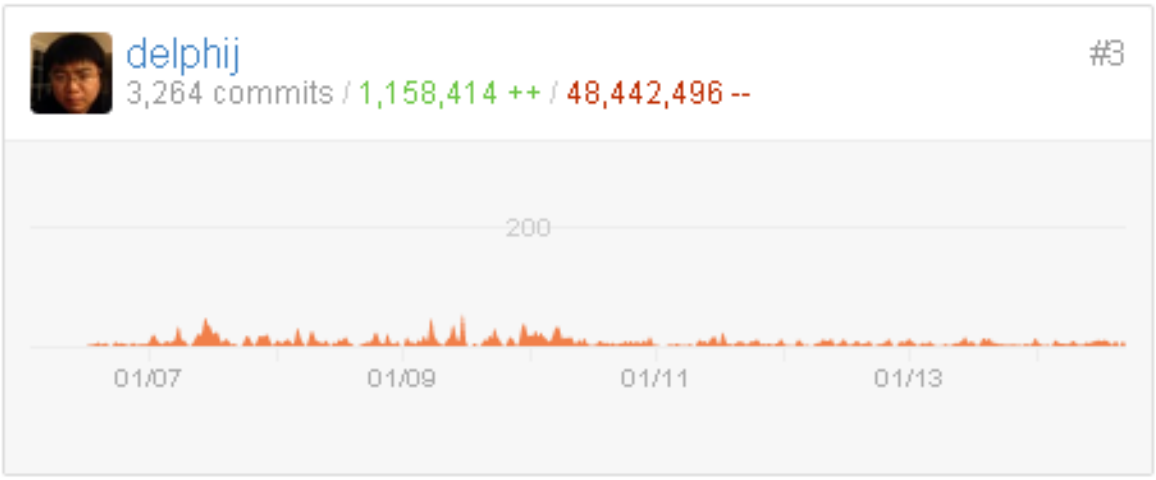

Figura 39 – Tiempo de vida FreeBSD vs Cantidad de commits desarrollador #3

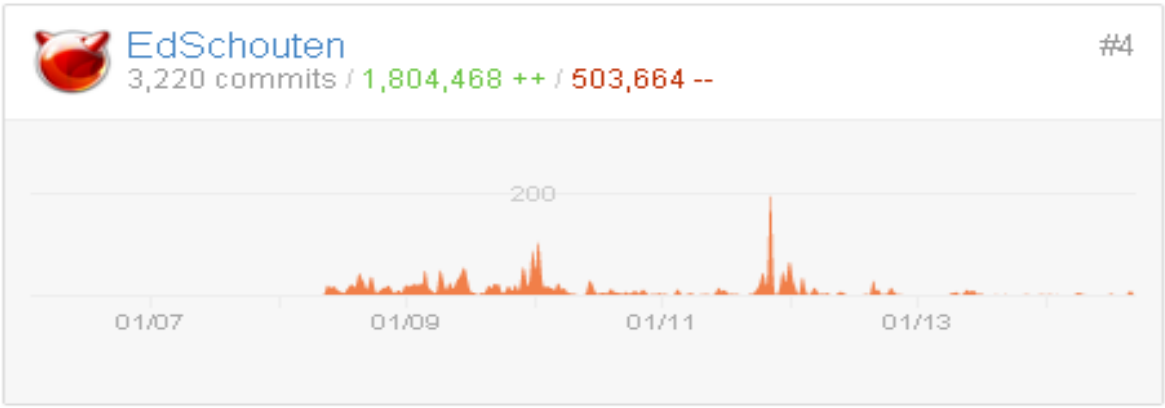

Figura 40 – Tiempo de vida FreeBSD vs Cantidad de commits desarrollador #4

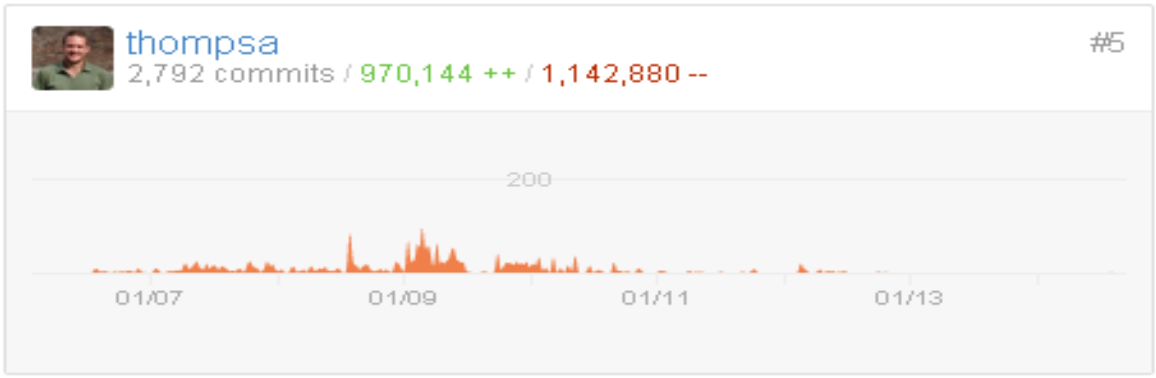

Figura 41 – Tiempo de vida FreeBSD vs Cantidad de commits desarrollador #5

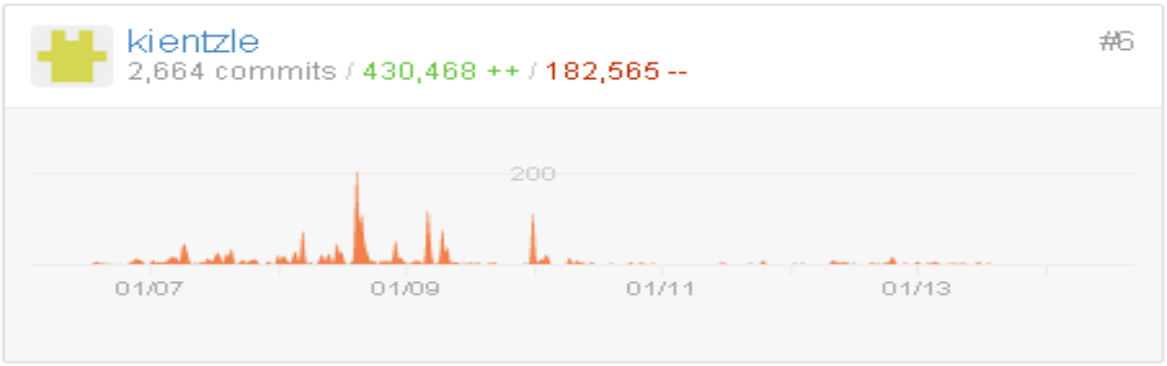

Figura 42 – Tiempo de vida FreeBSD vs Cantidad de commits desarrollador #6

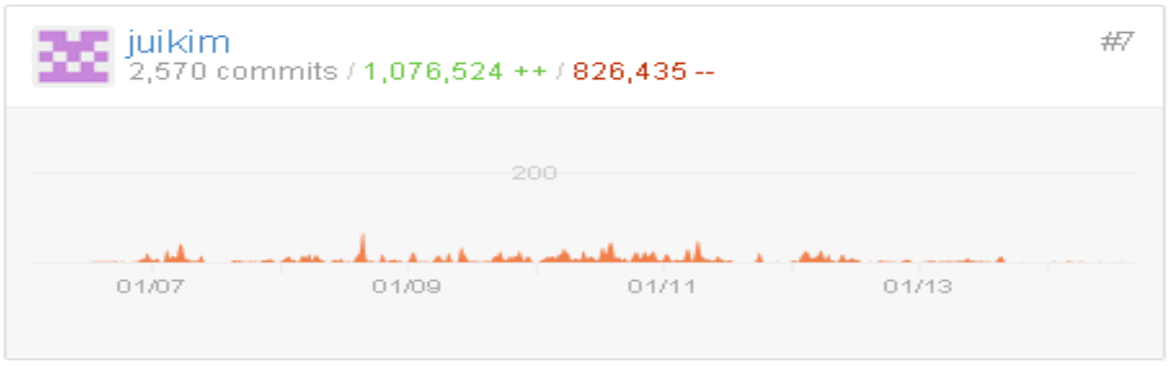

Figura 43 – Tiempo de vida FreeBSD vs Cantidad de commits desarrollador #7

La conclusión principal que se puede obtener de analizar las operaciones de interacción con el sistema de versiones que es que aunque en cortos periodos de tiempo las aportaciones son pocas por miembro y también por la totalidad del grupo del proyecto, es probable que examinando la cantidad de interacciones que los miembros tienen con el sistema de versiones en largos periodos de tiempo, presentan una cantidad considerable de aportaciones y visiblemente un ritmo de trabajo extendido en el tiempo (meses).

3.2.3 Mecanismo de cooperación en caso de estudio distribución FreeBSD

En este caso de estudio se pueden observar dos situaciones importantes la primera que muestra el poco volumen de aportaciones de los miembros pero sostenido en el tiempo y el segundo la caída vertiginosa en la actividad en el periodo de julio de 2009 que da origen a una lenta recuperación del volumen de la actividad, así los mecanismos identificados son:

Mecanismo de cooperación basado en confianza: En este caso existen retardos en la retoma del ritmo de aportaciones lo que significa que la confianza comienza a ser incremental, saliendo de la reducción del ritmo de aprendizaje, subiendo el volumen de desarrollo y mejorando la confianza entre los miembros de la comunidad[31].

La reciprocidad en este caso se presenta porque un gran número de participaciones acompaña de forma simultanea el desarrollo de tareas, lo que

[31] Ver Jorge Andrick Parra Valencia, Constructo para la evaluación de la cooperación en dilemas sociales de gran escala , Pág. 43

significa que las aportaciones de una parte de la comunidad va continuada de una acción de aportaciones en desarrollo cometidas por el resto de la comunidad[32].

[32] Ver Kollock 1998, SOCIAL DILEMMAS: The Anatomy of Cooperation, Pág. 199

3.3 Caso de estudio Gentoo/KDE

El proyecto Gentoo/KDE proporcionar soporte para todos los paquetes pertenecientes oficialmente al proyecto KDE. Soporta también un mayor número de paquetes subyacentes con el fin de disponer de un entorno KDE completamente funcional. Aunque el proyecto no soporta aplicaciones no oficiales, si se asegura que estas utilicen los eclasses de KDE adecuadamente, cuestión que se certifica con base a pruebas elaboradas de compatibilidad y funcionalidad de las aplicaciones oficiales

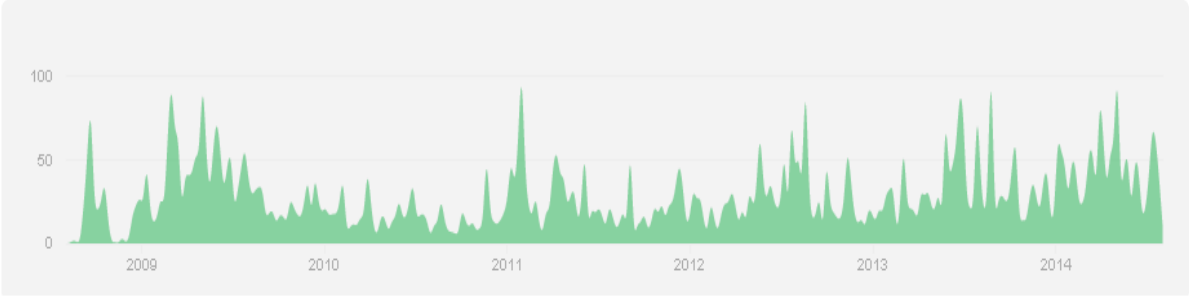

Figura 44 – Tiempo de vida Gentoo/KDE vs Cantidad de commits

Aunque en el periodo anterior a la versión KDE 4.8 lanzada en Enero 25 de 2012 no se observa un comportamiento de interacciones que fuera incrementándose lo que sí es destacable en el periodo 2012-2013 es un notable incremento total de participación de la comunidad en el proyecto. Situación que será visiblemente creciente para los periodos subsiguientes. Estos incrementos constantes pueden deberse principalmente a la popularización exorbitante que ha tenido el proyecto gracias a el soporte constante, la traducción de KDE y sus herramientas oficiales a más de 30 idiomas, y el desarrollo de herramientas de desarrollo para este

entorno han hecho que el proyecto sostenga este crecimiento constante año tras año.

3.3.1 Algunos historiales en operación adicionar código

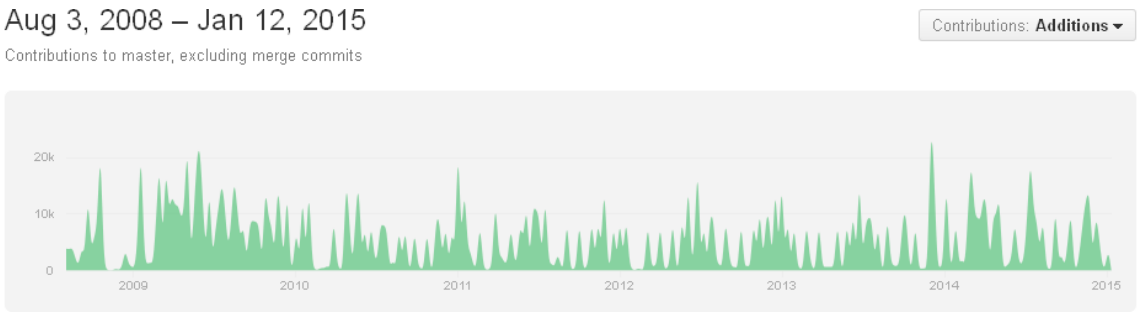

Figura 45 – Tiempo de vida Gentoo/KDE vs Cantidad de adiciones

Historial de aportaciones de algunos miembros más activos dentro del desarrollo del proyecto en el sistema de control de versiones presente en www.github.com

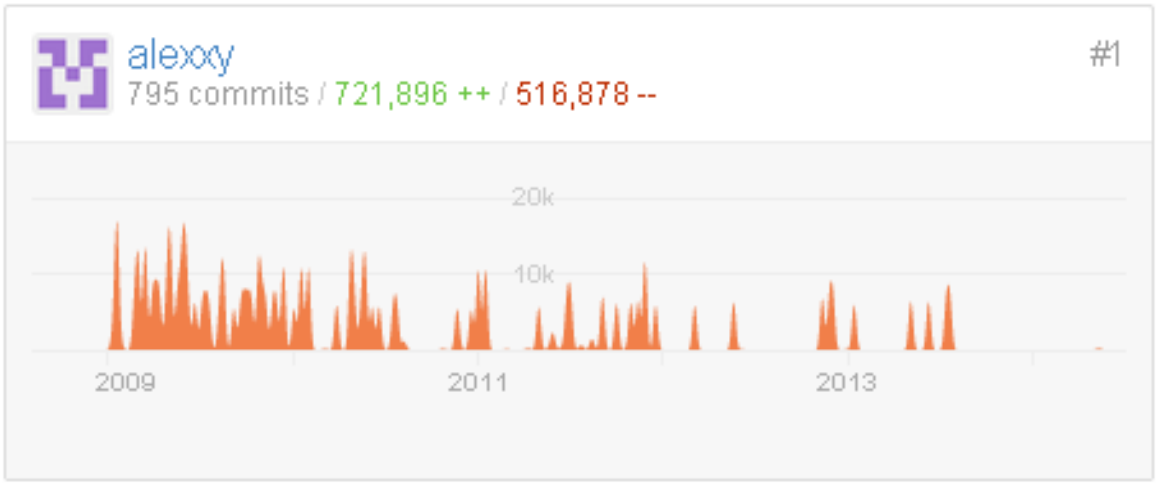

Figura 46 – Tiempo de vida Gentoo/KDE vs Cantidad de adiciones desarrollador #1

Figura 47 – Tiempo de vida Gentoo/KDE vs Cantidad de adiciones desarrollador #2

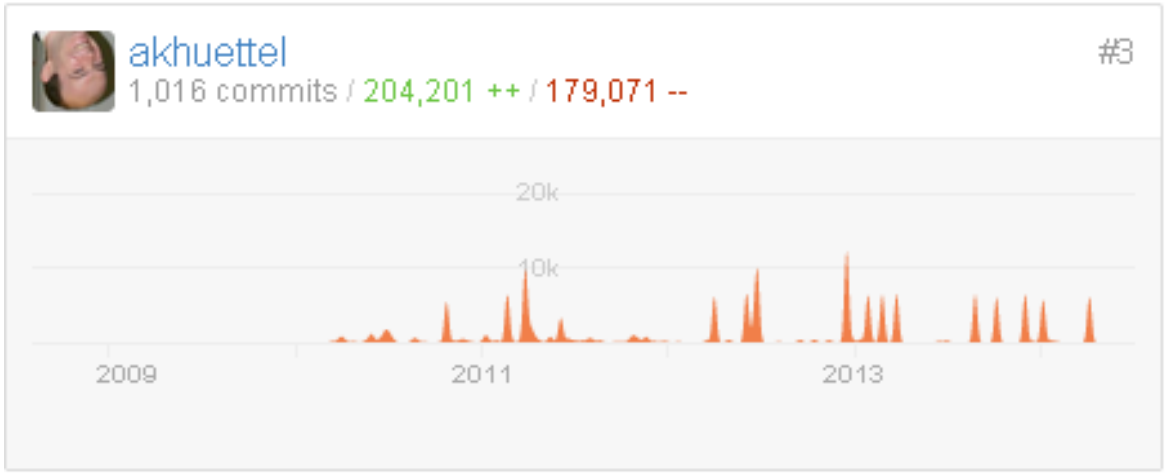

Figura 48 – Tiempo de vida Gentoo/KDE vs Cantidad de adiciones desarrollador #3

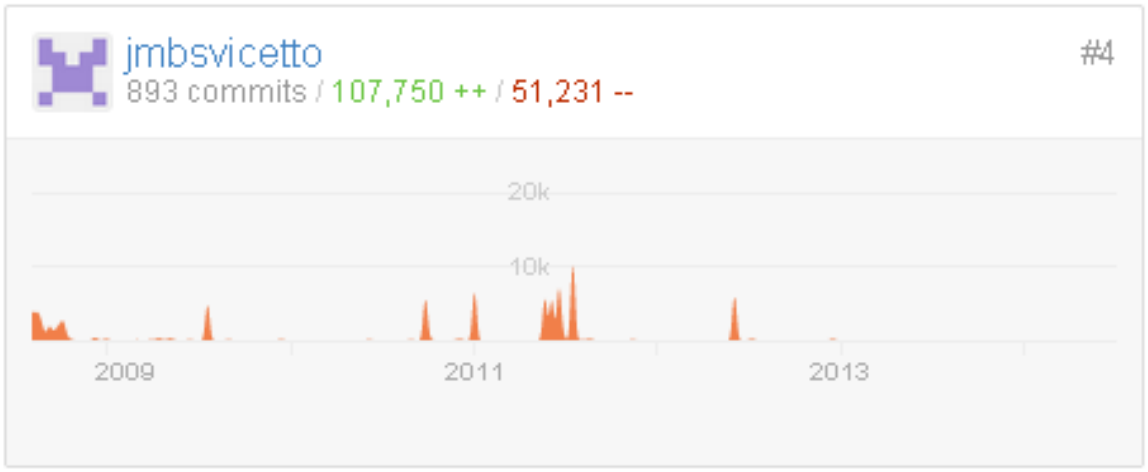

Figura 49 – Tiempo de vida Gentoo/KDE vs Cantidad de adiciones desarrollador #4

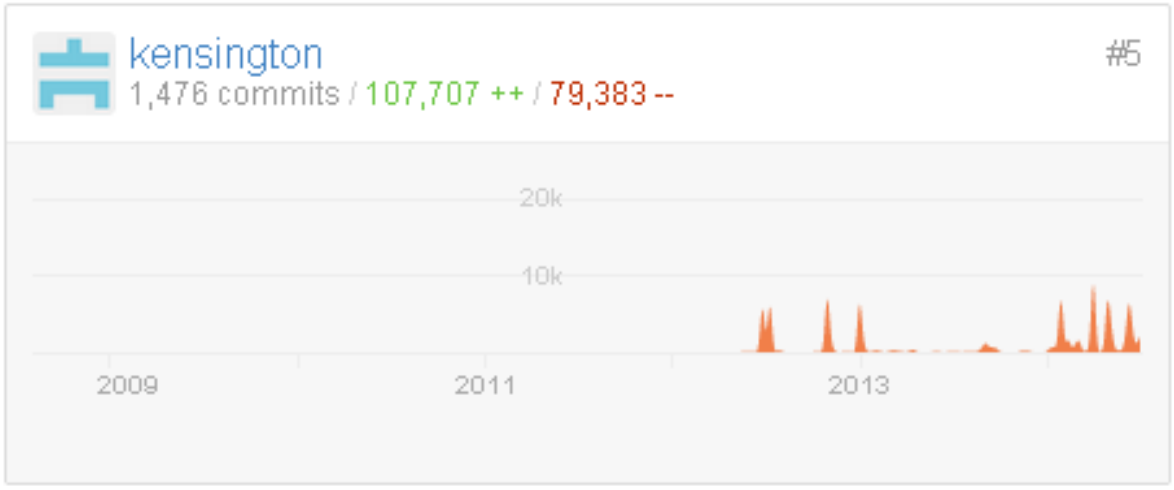

Figura 50 – Tiempo de vida Gentoo/KDE vs Cantidad de adiciones desarrollador #5

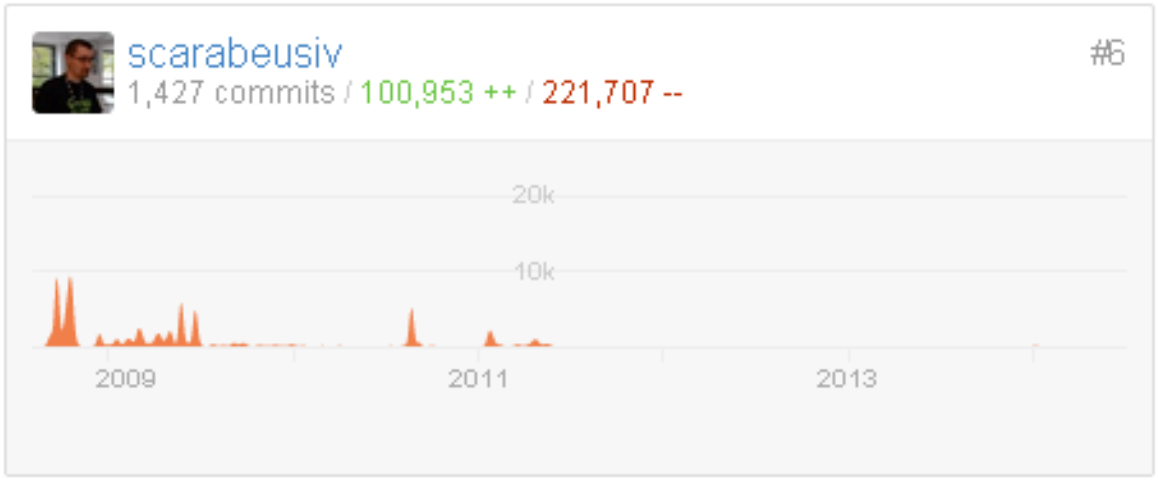

Figura 51 – Tiempo de vida Gentoo/KDE vs Cantidad de adiciones desarrollador #6

Aunque las interacciones de colaboración en sí, varían bastante de un individuo a otro, lo que sí es notable es que los dos integrantes que más han colaborado en número son dos integrantes que últimamente han estado activos lo que significa que aunque las colaboraciones responden a un interés por mejorar el proyecto, se

puede ver que es en este último periodo 2013-2014 que las contribuciones son más altas por parte de algunos miembros, normalmente miembros nuevos[33], lo que sugeriría un gran número de horas individuales invertidas al proyecto, muchas más horas que otros miembros que pueden haber estado más tiempo activos.

Se puede concluir en este caso de estudio que *el volumen de aportaciones ha crecido proporcionalmente a la popularidad del proyecto, el enriquecimiento de sus funcionalidades.*

3.3.2 Algunos historiales de aportación en situación operación commit

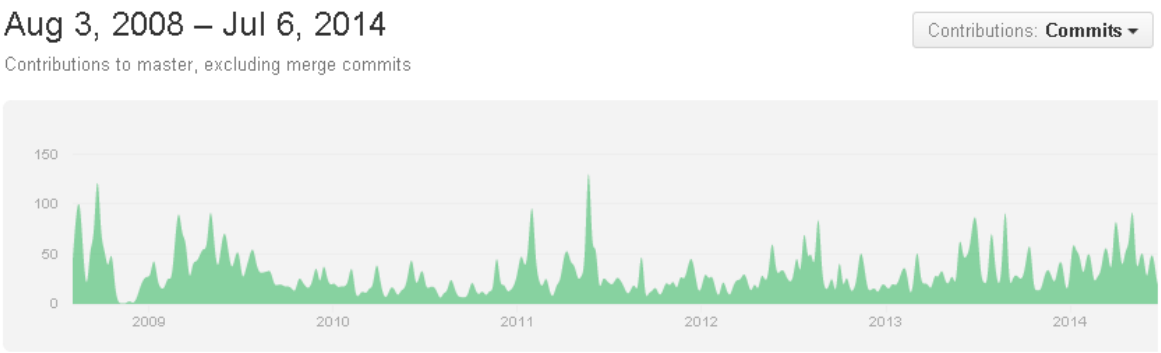

Figura 52 – Tiempo de vida Gentoo/KDE vs Cantidad de commits

El grafico anterior nos muestra la actividad constante que ha tenido el proyecto Gentoo/Kde también en el caso de la operación commit, lo que significa una alta rotación del proyecto hacia código nuevo y nuevas características, también puede

[33] Ver Anexo C-2 - Estadisticas commit proyecto Geentoo/Kde

deberse a la actualización del proyecto hacia nuevas distribuciones Linux o más seguramente a ambos factores.

Los siguientes gráficos muestran la actividad de los miembros en la aplicación de la operación commit.

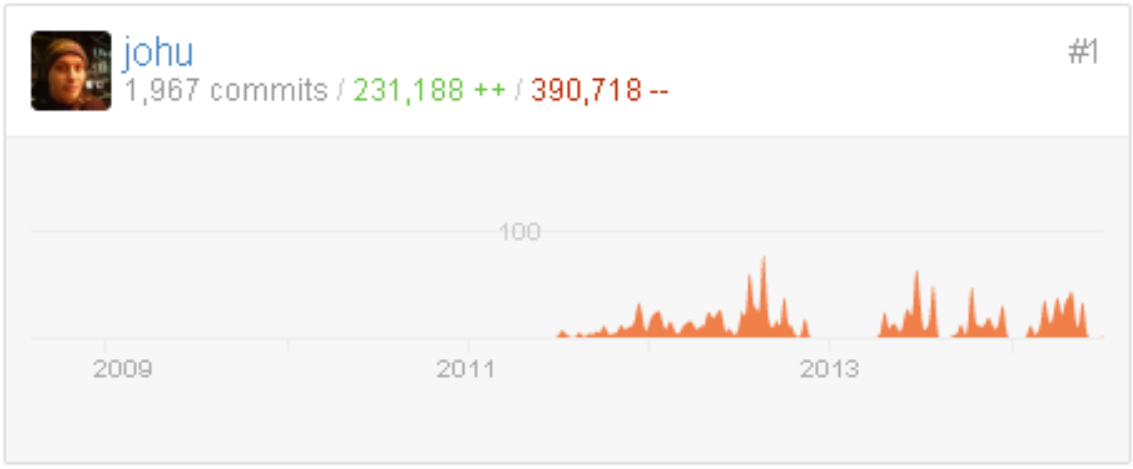

Figura 53 – Tiempo de vida Gentoo/KDE vs Cantidad de commits desarrollador #1

Figura 54 – Tiempo de vida Gentoo/KDE vs Cantidad de commits desarrollador #2

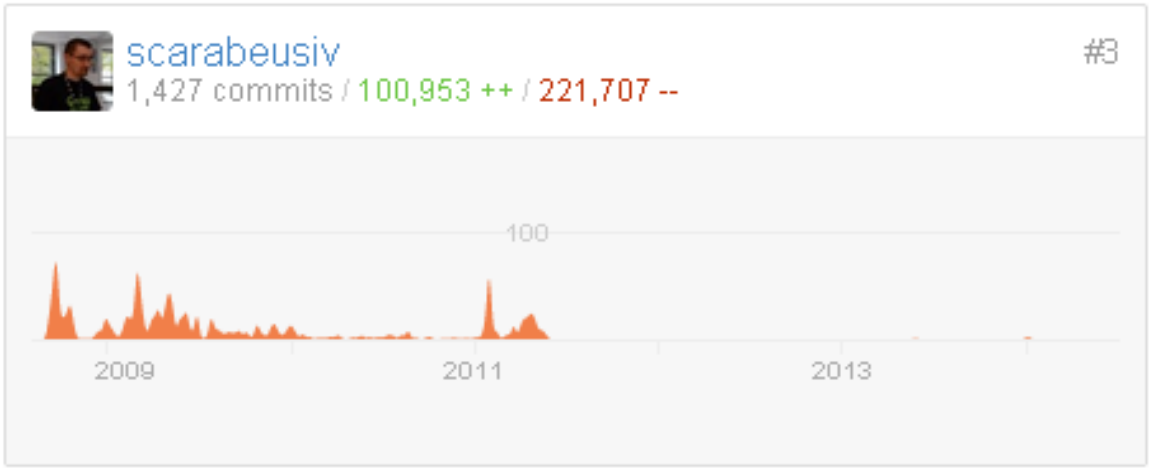

Figura 55 – Tiempo de vida Gentoo/KDE vs Cantidad de commits desarrollador #3

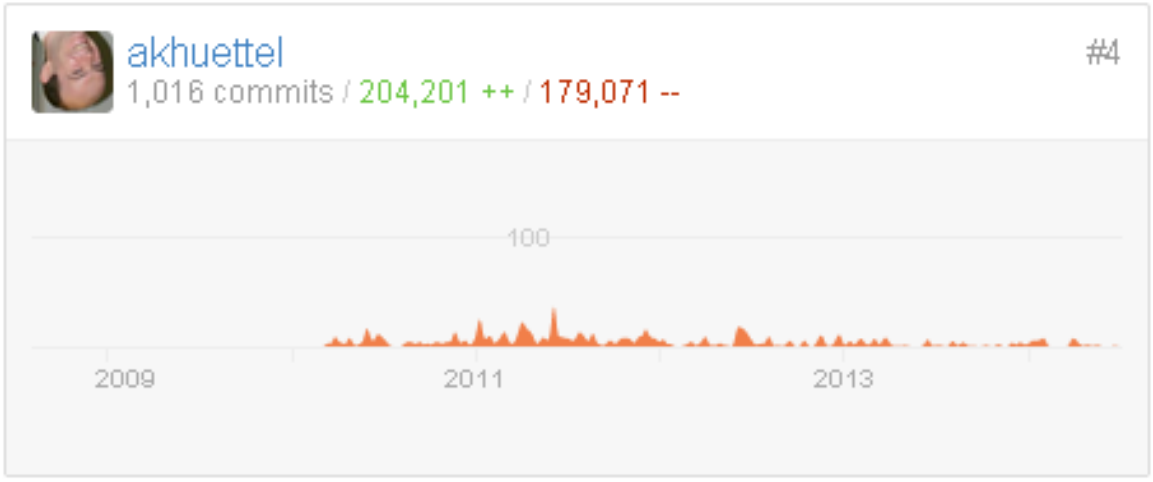

Figura 56 – Tiempo de vida Gentoo/KDE vs Cantidad de commits desarrollador #4

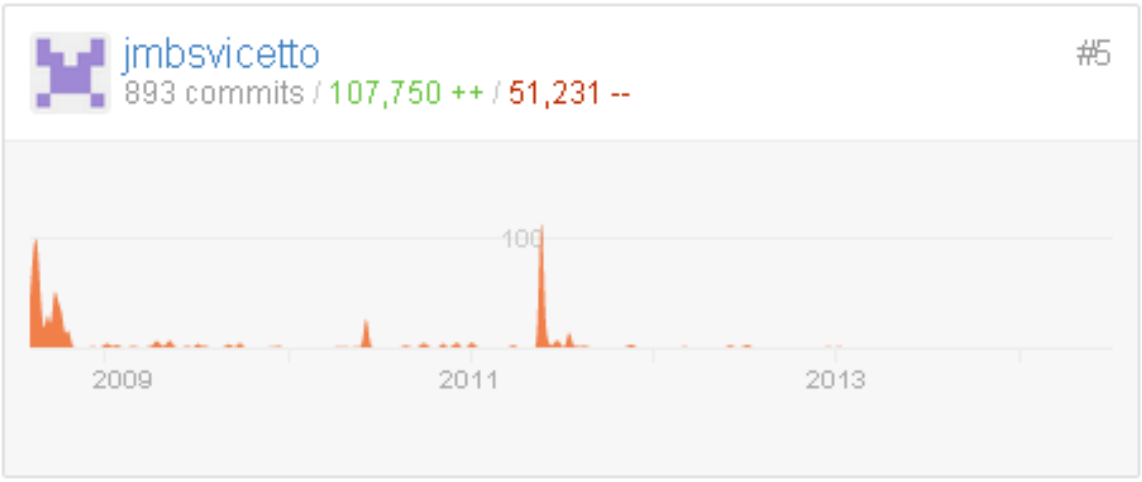

Figura 57 – Tiempo de vida Gentoo/KDE vs Cantidad de commits desarrollador #5

Figura 58 – Tiempo de vida Gentoo/KDE vs Cantidad de commits desarrollador #6

Teniendo como herramientas las comparaciones visuales entre las operaciones **commit** y **adicionar** código de los usuarios alexxy, johu y akhuettel podemos ver que estas dos operaciones se corresponden de forma proporcional, lo que puede significar que en el trasegar de colaboración en el proyecto su colaboración ha sido constante y el periodo de prueba es relativamente corto, o para el objeto principal de este proyecto es inmediato lo que expone claramente las gráficas que

el tiempo entre las actividades de adicion y commit es casi 0, ya que este proyecto en particular no se ocupa de hacer grandes aportaciones de funcionalidad en nuevos desarrollos sino en probar y mejorar compatibilidades cuestión que en la práctica raras veces requiere de una gran cantidad de escritura de código.

3.3.3 Mecanismo de cooperación en Caso de estudio Gentoo/KDE

Ya que en este proyecto la rotación y llegada de miembros es notable en el conjunto de los más aportantes y también, que el tiempo de pruebas y modificación en las tareas de los miembros es casi 0, inmediato, se pueden citar como principales mecanismos de colaboración:

Mecanismo de cooperación por aprendizaje social: en la práctica cuando un miembro llega al proyecto tiene la intención de probar el software de escritorio KDE sobre la distribución Getoo, la persona que ingresa en esta actividad posee capacidades para la compilación y pruebas de funcionalidades, siendo la operación commit y las correcciones al código un proceso normalmente individual o individual secuencial[34], así el comportamiento social rápidamente aprendido es la prueba y aseguramiento (commits) de cambios del código fuente; diferencia marcada en comparación a demás proyectos de desarrollo donde el usuario/programador se ocupa en primera instancia de implementar nuevas funcionalidades que por lo general son tan extensas que requieren la interacción de múltiples personas[35].

---

[34] Ver Anexo C - Estadisticas commit proyecto Geentoo/Kde

[35] Kollock 1998, SOCIAL DILEMMAS: The Anatomy of Cooperation, Pág. 199

Mecanismo de cooperación por estructura de pagos: en este caso se puede observar que la colaboración se mantiene y aun se incrementa en base a las aportaciones al proyecto gracias a la racionalidad que existe cuando el miembro ve reflejadas sus aportaciones en el proyecto de desarrollo del cual puede hacer un uso individual[36] y del cual tal vez pueda demostrar tangiblemente la mejora efectuada.

3.4 Caso de estudio Joomla-CMS

En el grafico se puede observar la actividad de operaciones commits en el proyecto Joomla-CMS que ofrece un promedio de participaciones estable, en el orden de los 30 a 35 promedio durante el periodo de mediados de 2005 a inicios de 2008, y un 20 a 25 promedio durante el periodo finales de 2008 a septiembre de 2014, ahora bien se muestra un notable receso durante el periodo Enero 20 de 2008 a Septiembre de 2008; este periodo de receso se tratara de explicar buscando una correspondencia causal con las versiones finales publicadas durante la época o por el balance de participantes activos durante el periodo en cuestión. Observando en la tabla de versiones del proyecto se puede concluir que las versiones más próximas a este periodo son la versión 1.5 (LTS) con soporte desde enero 22 de 2008 hasta diciembre 01 de 2012, lo que explica claramente el poco desarrollo durante el periodo 2008, orientado a la corrección de fallas y refinamiento de la versión 1.5"[37].

---

[36] Kollock 1998, SOCIAL DILEMMAS: The Anatomy of Cooperation, Pág. 200

[37] Ver Jennifer Marriott 2010, Elin Waring, The Official Joomla! Book, Pag. 13

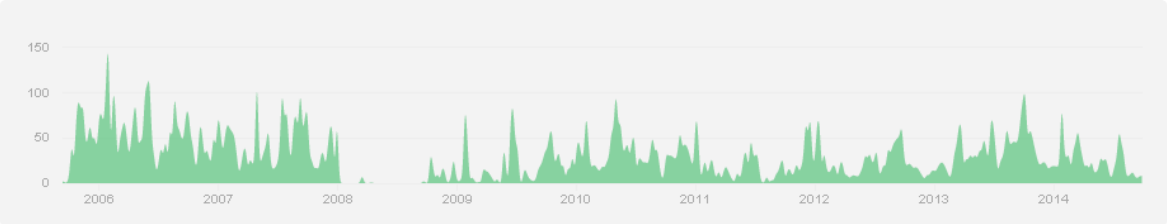

Figura 59 – Tiempo de vida Joomla-CMS vs Cantidad de commits

3.4.1 Algunos historiales en operación adicionar código

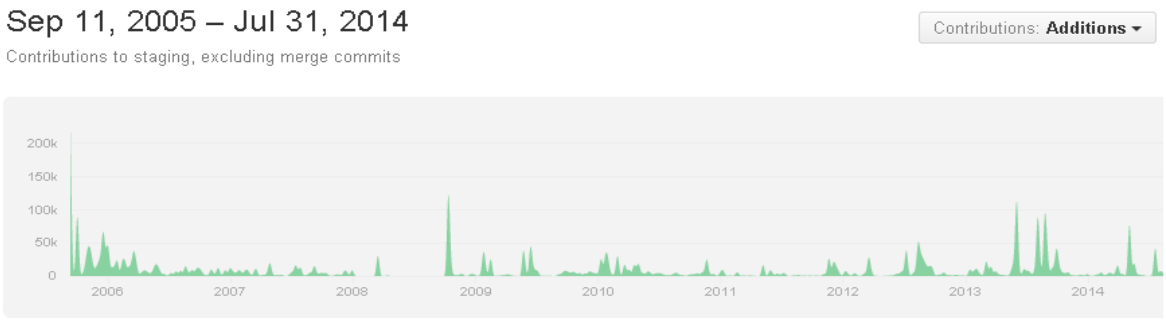

Figura 60 – Tiempo de vida Joomla-CMS vs Cantidad de adiciones

Figura 61 – Tiempo de vida Joomla-CMS vs Cantidad de adiciones desarrollador #1

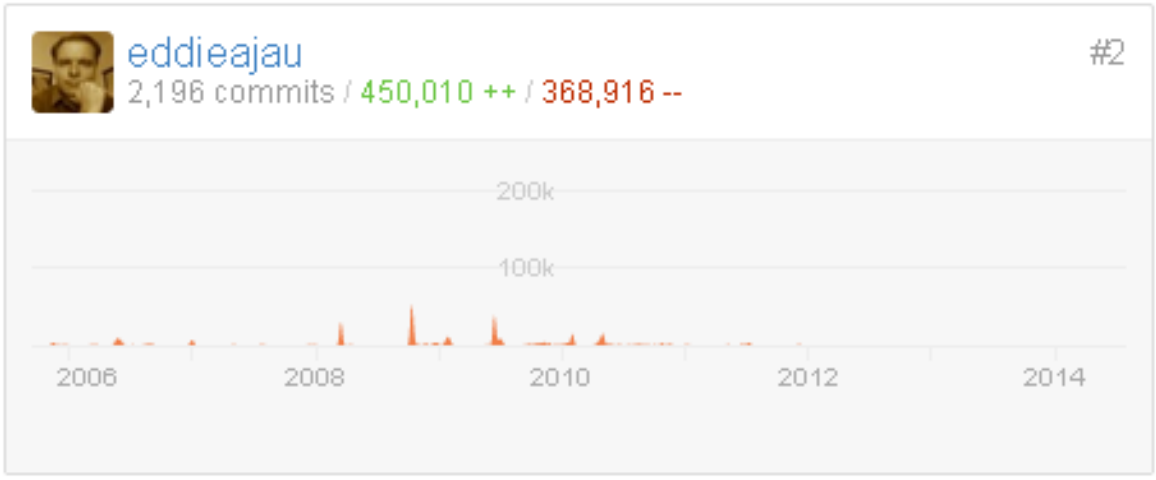

Figura 62 – Tiempo de vida Joomla-CMS vs Cantidad de adiciones desarrollador #2

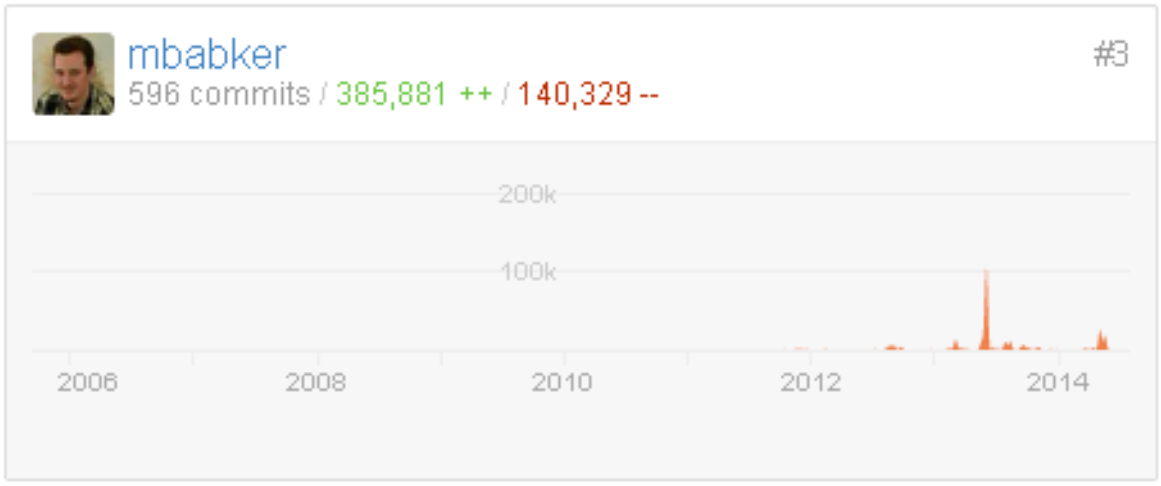

Figura 63 – Tiempo de vida Joomla-CMS vs Cantidad de adiciones desarrollador #3

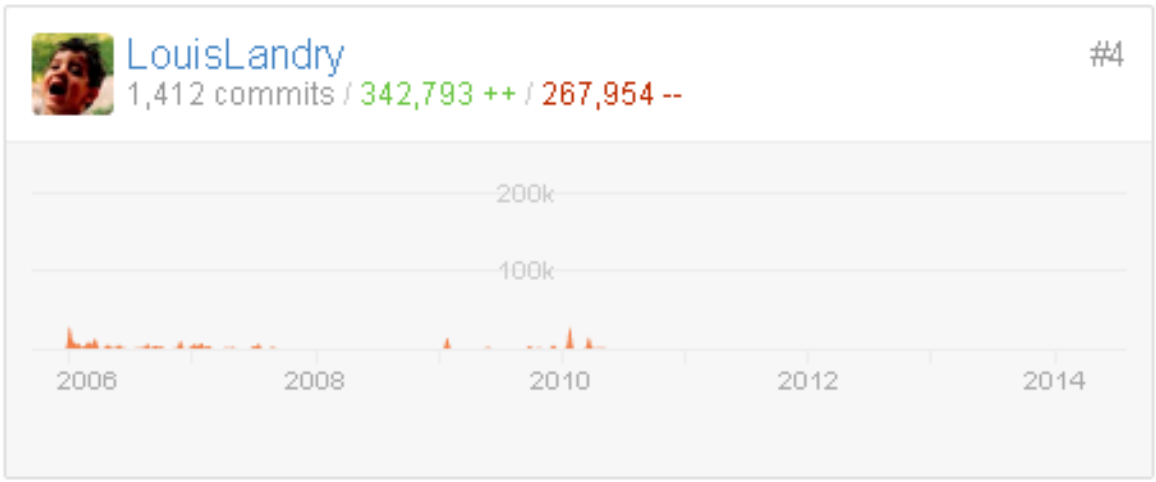

Figura 64 – Tiempo de vida Joomla-CMS vs Cantidad de adiciones desarrollador #4

Figura 65 – Tiempo de vida Joomla-CMS vs Cantidad de adiciones desarrollador #5

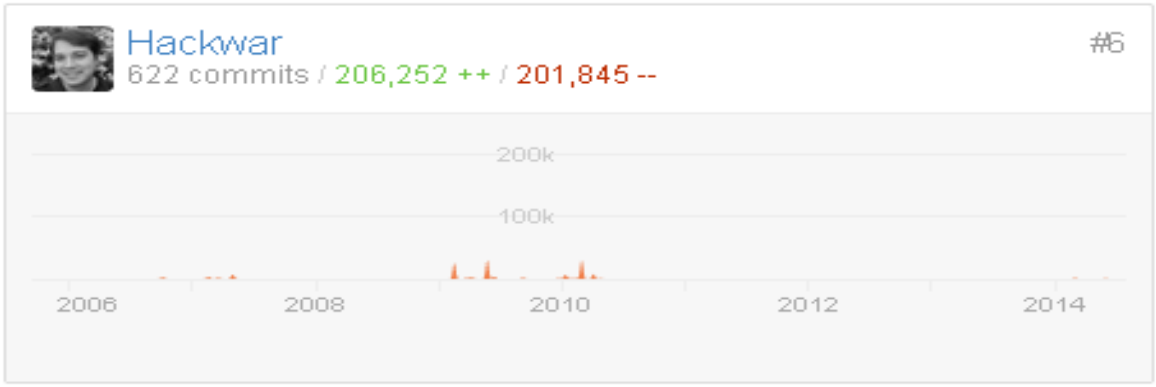

Figura 66 – Tiempo de vida Joomla-CMS vs Cantidad de adiciones desarrollador #6

Las situación de aportaciones en el proyecto por los miembros más destacados es correspondiente al grafico principal durante el mismo periodo, con algunas diferencias y picos de desarrollo, pero su principal característica general es la visible inactividad presentada durante el periodo 2008, situación ya estudiada anteriormente.

3.4.2 Algunos historiales de aportación en operación commit

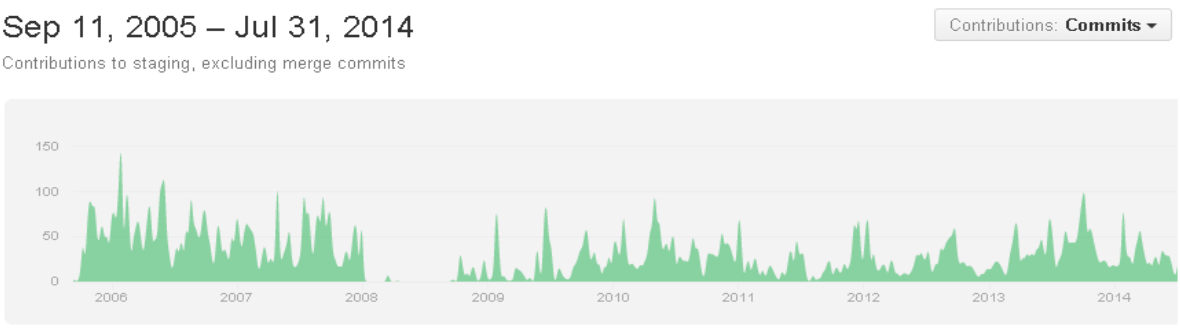

Figura 67 – Tiempo de vida Joomla-CMS vs Cantidad de commits desarrollador

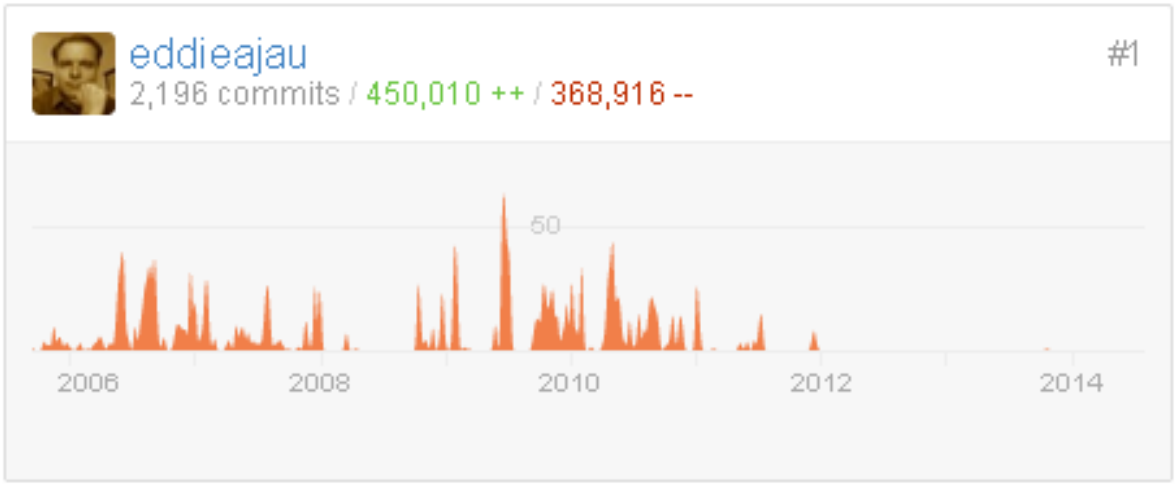

Figura 68 – Tiempo de vida Joomla-CMS vs Cantidad de commits desarrollador #1

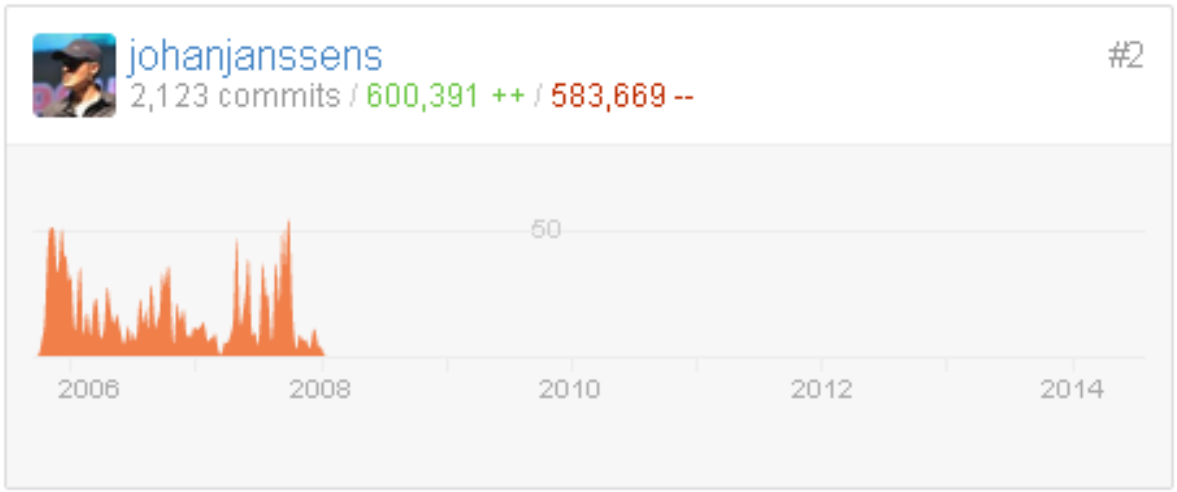

Figura 69 – Tiempo de vida Joomla-CMS vs Cantidad de commits desarrollador #2

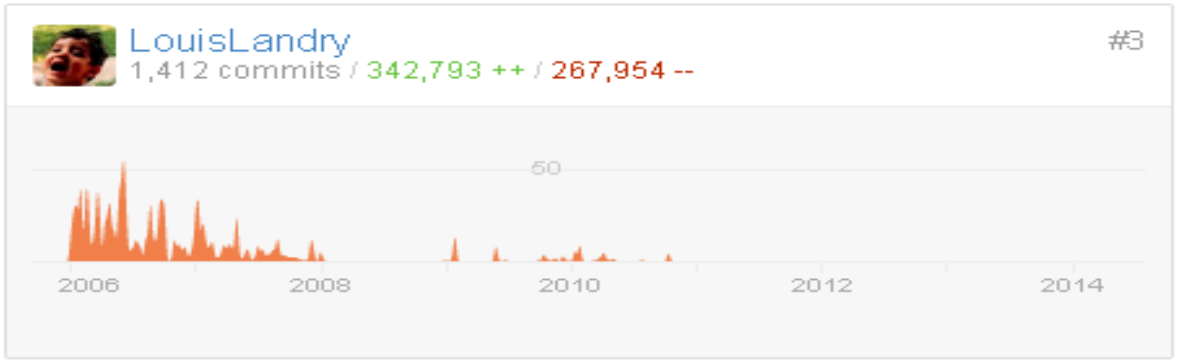

Figura 70 – Tiempo de vida Joomla-CMS vs Cantidad de commits desarrollador #3

Figura 71 – Tiempo de vida Joomla-CMS vs Cantidad de commits desarrollador #4

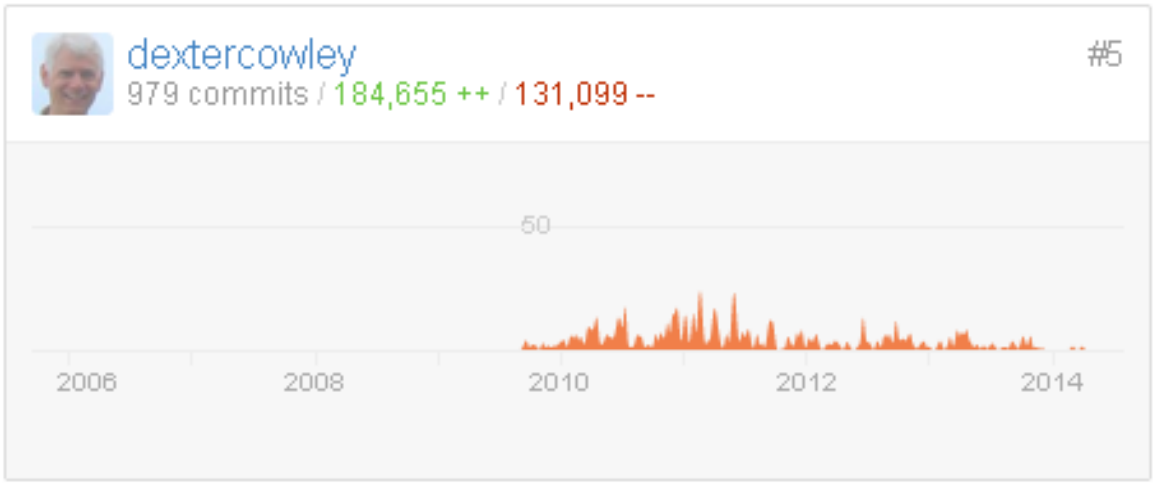

Figura 72 – Tiempo de vida Joomla-CMS vs Cantidad de commits desarrollador #5

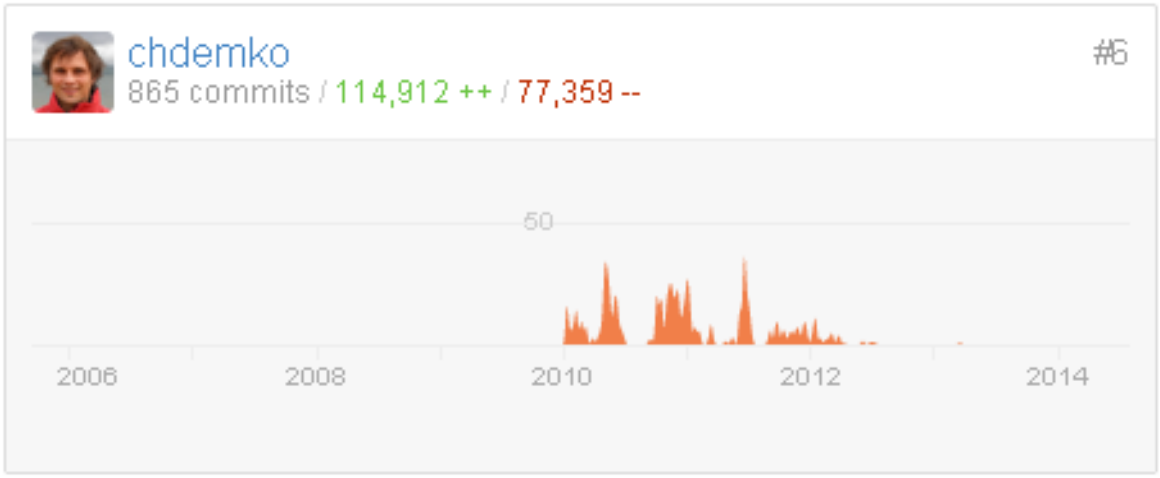

Figura 73 – Tiempo de vida Joomla-CMS vs Cantidad de commits desarrollador #6

En la operación commit se puede observar que las aportaciones se mantienen en un volumen con picos hasta de más de 50 aportaciones cuestión que contrasta con el desarrollo de las otras versiones de Joomla como la 1.x y la 1.6[38]

3.4.3 Mecanismo de cooperación en caso de estudio Joomla-CMS

Mecanismo de iteración e identificabilidad: En vista de que en el periodo 2008 la comunidad de desarrollo de Joomla CMS atraviesa por el mantenimiento de un proyecto con dos ramas principales (1.x y la 1.5LTS) y el inicio de una rama secundaria (1.6), se hubiese podido esperar que durante los periodos subsiguientes la actividad se tornara baja o apenas reconocible, hecho que en la realidad fue totalmente contrario, la comunidad inicio participaciones activas en las

[38]Ver Jennifer Marriott 2010, Elin Waring, The Official Joomla! Book, Pag. 13

tres ramas de forma igualmente constante en el periodo 2009, este mecanismo habla de la mejora en la disponibilidad de información, de la duración de los encuentros entre los cooperadores y de la identificabilidad de los participantes[39], cuestiones ampliamente reflejadas durante el periodo 2009, en el cual la versión 1.6 tuvo los mayores avances unificando y tomando características de las otras ramas de desarrollo cuestión que se logró gracias al extendido numero en el tiempo de interacciones con el sistema de control de versiones github.com sustentable también por el motivo de que entre los 100 miembros con más índice de aportaciones en el proyecto, 15 miembros han estado activos desde antes del año 2008 y 7 de ellos siguieron activos contribuyendo después del periodo 2008[40].

[39] Kollock 1998, SOCIAL DILEMMAS: The Anatomy of Cooperation, Pág. 199

[40] Ver Anexo C - Estadisticas commit proyecto Joomla-CMS

3.5 Caso de estudio PhpMyAdmin

El programa PhpMyAdmin está desarrollado en php, html y javascript, la finalidad principal es la de administrar un servidor MySQL entero o una base de datos sencilla. Es muy popular en desarrollo y manejo de bases de datos Mysql, se puede obtener como un paquete individual para instalación en servidor con motores de base de datos MySql pero también es distribuido ampliamente como parte de paquetes integrales de servidor orientados al manejo de base de datos MySQL, aunque también puede manejar a través de configuraciones especiales bases de datos de motores de bases de datos tales como PostgreSQL, SQLite entre otras. En la actualidad este programa se encuentra licenciado en GPL v2[41].

"MySQL es sobre todo compatible con el estándar SQL 2003, es un sistema de base de datos bien conocido por su velocidad, robustez y poca sobrecarga de conexión. Esto es importante en el contexto de página web en la que deben ser servida la información lo más rápidamente posible".[42]

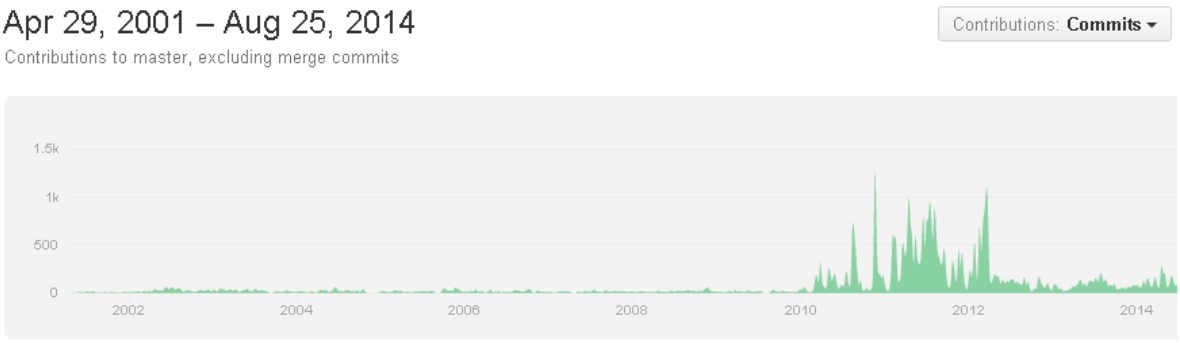

Figura 74 – Tiempo de vida PhpMyAdmin vs Cantidad de commits

[41] Ver 1998-2000 Tobias Ratschiller, https://phpmyadmin-spanish.readthedocs.org/en/latest/copyright.html , Derechos de autor

[42] Ver 2012 Marc Delisle, Mastering phpMyAdmin 3.4 for Effective MySQL Management, Pag. 7

En el grafico se puede apreciar que el proyecto ha estado activo desde 2002 en la plataforma de manejo de versiones github.com con promedio de aportaciones de alrededor de 100, desde el año 2002 hasta el inicio del periodo 2010. En el periodo 2010 las aportaciones aumentan de forma enorme comparada con los periodos anteriores, lo que muestra el aumento de popularidad en el desarrollo gracias al desarrollo y mejoras en el lanzamiento de la versión 3.3 desde el 31 de diciembre de 2009[43].

3.5.1 Algunos historiales en operación adicionar código

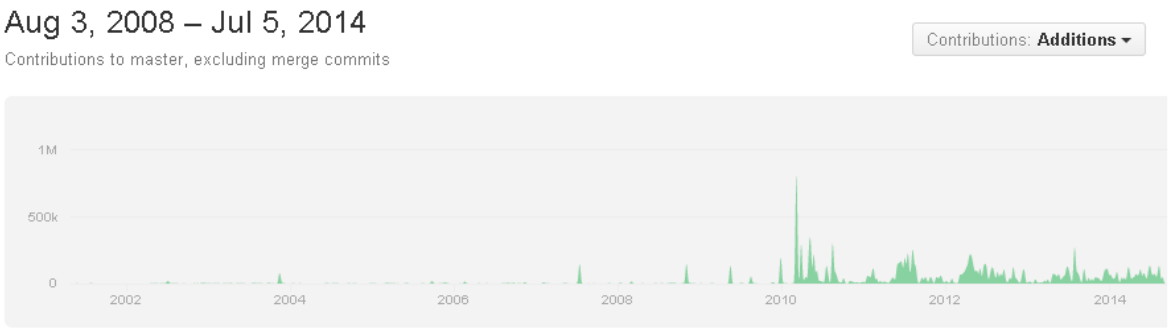

Figura 75 – Tiempo de vida PhpMyAdmin vs Cantidad de adiciones

[43] Ver SourceForge.net 2009, http://sourceforge.net/p/phpmyadmin/news/?page=17 , News PhpMyAdmin,

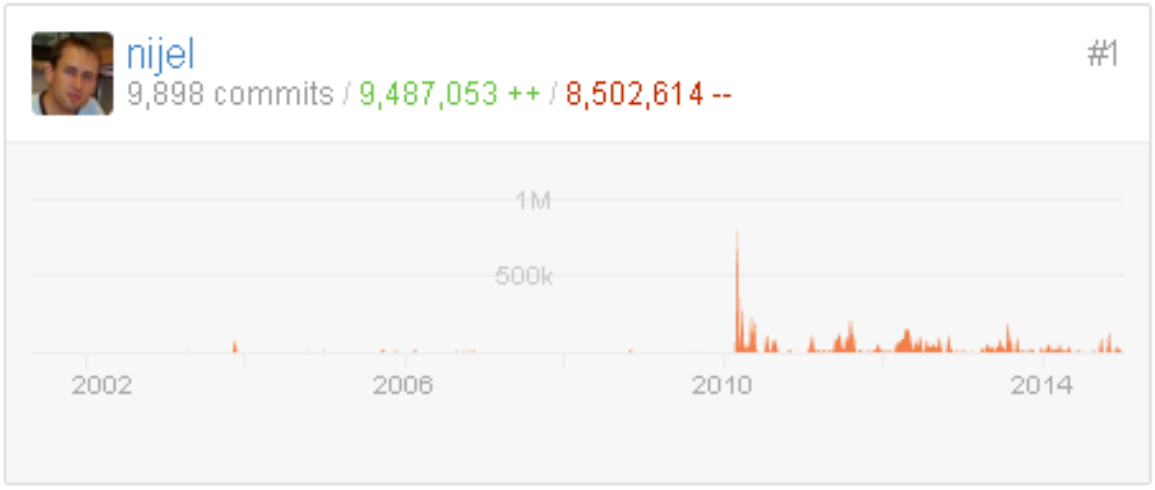

Figura 76 – Tiempo de vida PhpMyAdmin vs Cantidad de adiciones desarrollador #1

Figura 77 – Tiempo de vida PhpMyAdmin vs Cantidad de adiciones desarrollador #2

Figura 78 – Tiempo de vida PhpMyAdmin vs Cantidad de adiciones desarrollador #3

Figura 79 – Tiempo de vida PhpMyAdmin vs Cantidad de adiciones desarrollador #4

Figura 80 – Tiempo de vida PhpMyAdmin vs Cantidad de adiciones desarrollador #5

Figura 81 – Tiempo de vida PhpMyAdmin vs Cantidad de adiciones desarrollador #6

Entre los participantes más activos del proyecto PhpMyAdmin se observa que efectivamente las aportaciones van en incremento en el periodo 2010, lo que demuestra que gracias al inicio del desarrollo de la versión 3.3 el número de aportaciones por miembro se incrementa también.

Es de anotar que el volumen de aportaciones en la operación adicionar es bastante grande en la comunidad, como se observa en el primer cuadro unos 500

o 1.000 aportaciones diarias en la comunidad hasta llegar a picos de 100.000 y un importante pico de 800.000 en marzo de 2010.

3.5.2 Algunos historiales de aportación en situación operación commit

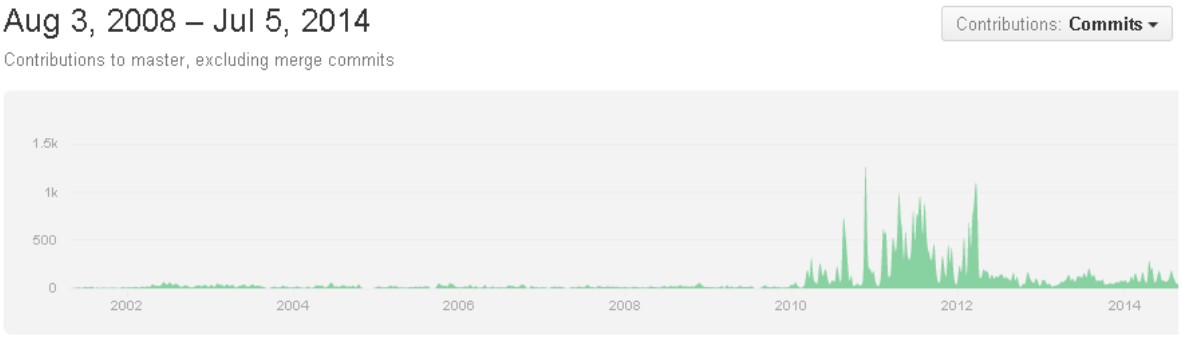

Figura 82 – Tiempo de vida PhpMyAdmin vs Cantidad de commits

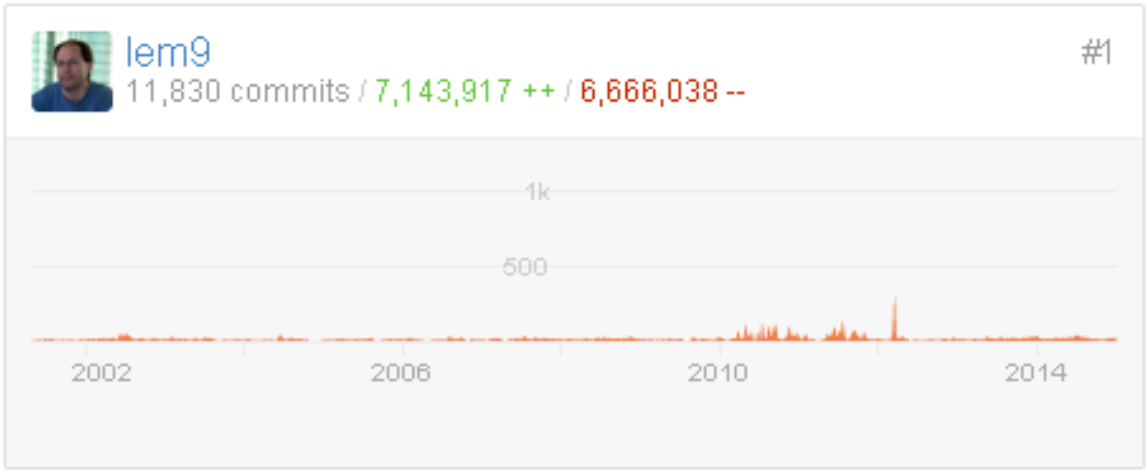

Figura 83 – Tiempo de vida PhpMyAdmin vs Cantidad de commits desarrollador #1

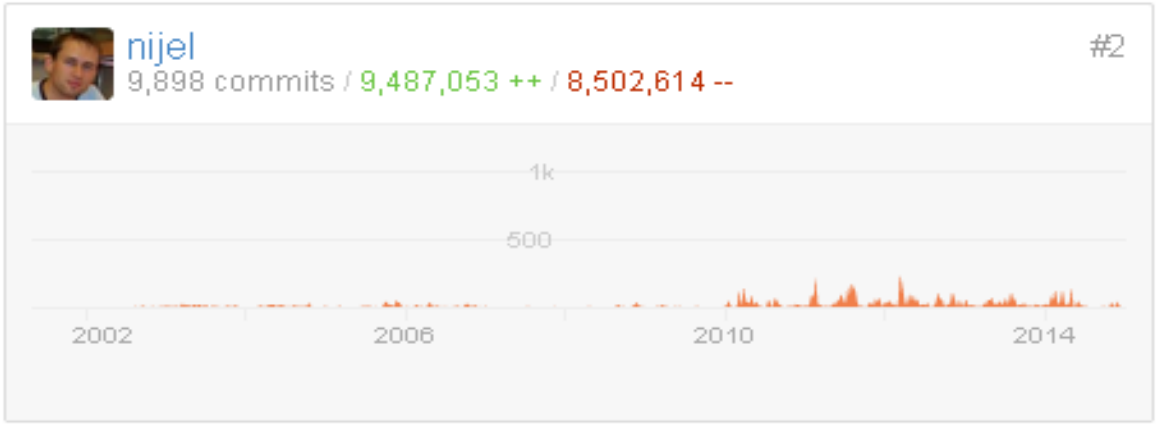

Figura 84 – Tiempo de vida PhpMyAdmin vs Cantidad de commits desarrollador #2

Figura 85 – Tiempo de vida PhpMyAdmin vs Cantidad de commits desarrollador #3

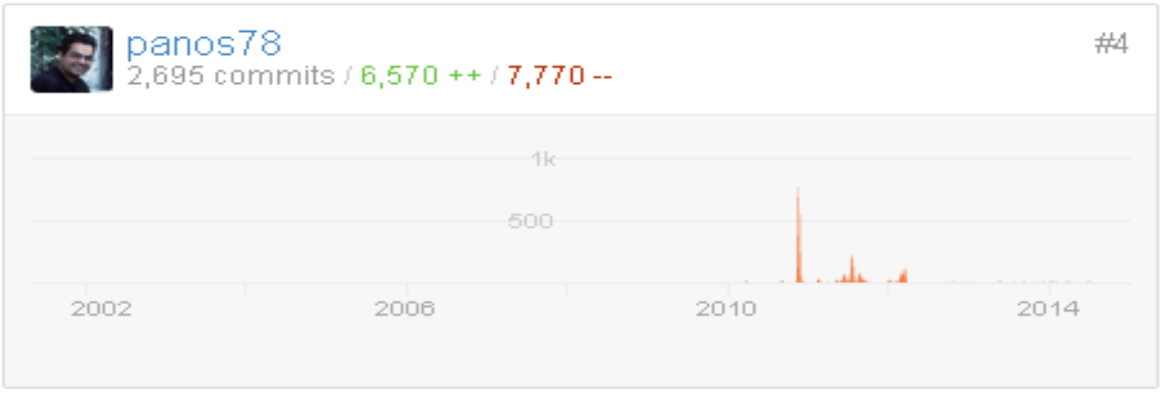

Figura 86 – Tiempo de vida PhpMyAdmin vs Cantidad de commits desarrollador #4

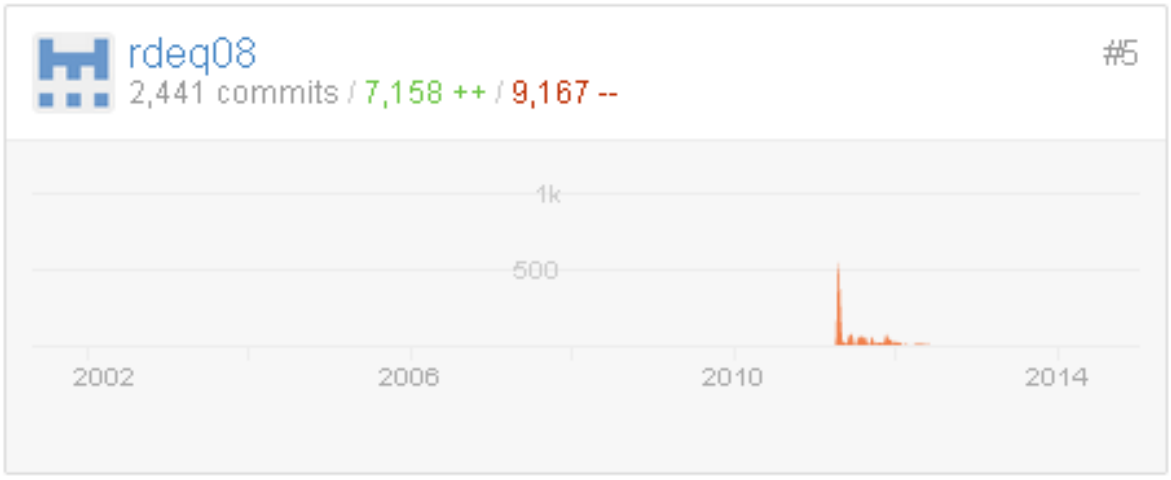

Figura 87 – Tiempo de vida PhpMyAdmin vs Cantidad de commits desarrollador #5

Figura 88 – Tiempo de vida PhpMyAdmin vs Cantidad de commits desarrollador #6

3.5.3 Mecanismo de cooperación en caso de estudio PhpMyAdmin

Mecanismo de cooperación basado en confianza: Este caso en particular de mecanismo de cooperación se puede enmarcar basado en la confianza primeramente, con una apreciación especial desde el periodo 2010 en cual cual inicia un crecimiento de aportaciones bastante notable llegando a picos de más de 1.000 aportaciones en algunos momentos, diferenciándose primariamente del Mecanismo de iteración e identificabilidad, que el proyecto de desarrollo PhpMyAdmin hasta el periodo 2010 no ha tenido grandes volúmenes de aportaciones y que hasta ese periodo el ritmo de aportaciones se mantuvo casi constante sin pausas largas que hubiesen bajado la apropiación de conocimientos a niveles. Ahora es importante notar que si se estudia el caso para el periodo 2010 entonces se pueden tener unas condiciones iniciales de confianza, reputación y reciprocidad que van creciendo a partir de ese momento.

Mecanismo de cooperación por aprendizaje social: Examinando este caso de estudio es posible hacer una similitud para el periodo 2010 y el periodo 2001, porque en el 2001 el proyecto inicia formalmente registrado en sourceforge.net con un numero de aportaciones que va creciendo de forma cada vez más visible teniendo en cuenta que el proyecto iniciaba como continuación al proyecto de Peter Kuppelwieser y su MySQL-Webadmin, y para el periodo de 2010 el numero de aportaciones se incremente de forma más notable siendo clara también la idea de que en ese periodo ya el proyecto sería continuación de la versión 3.x que se venía desarrollando. Analizando las estadísticas de 100 principales cooperadores en el periodo 2010 hasta final de 2014[44] es posible observar que 62 miembros no presentan interacciones largas en el periodo 2010 lo que significa que son

[44] Anexo E - Estadisticas commit proyecto PhpMyAdmin

miembros nuevos que vinieron después de este periodo a entran en la comunidad de desarrollo[45], caso práctico que demuestra el uso de mecanismo de aprendizaje social durante estos dos grandes momentos de incremento de desarrollo en el proyecto.

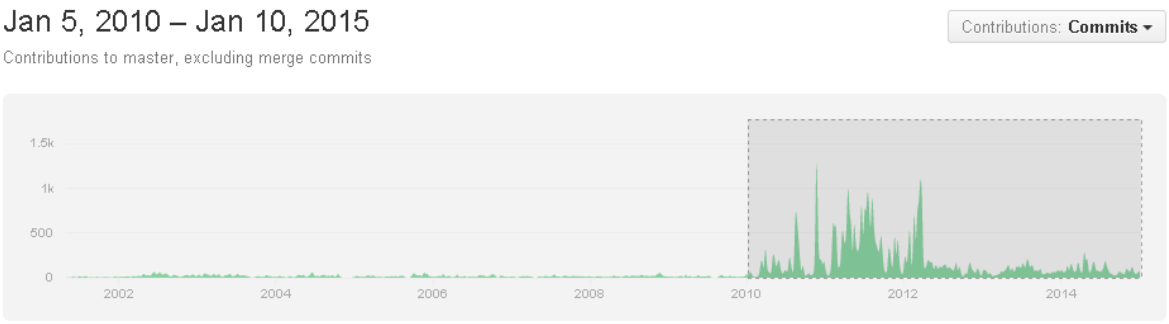

Figura 89 – Tiempo de vida Ruby on Rails vs Cantidad de commits

3.6 Caso de estudio Ruby on Rails

Ruby on Rails, es una framework que simplifica la programación para utilizarlo en el modelo de vista controlador, haciendo que la interacción con las bases de datos, los objetos del servidor y del cliente sea rápida de programar gracias a la abstracción del modelo que hace que este framework genere de forma automática vistas, operaciones de consulta entre otras. El soporte de RoR es amplio en el mercado de servidores web tales como Nginx, Mongrel, Apache, Lighttpd con

[45] Ver Jorge Andrick Parra Valencia, Constructo para la evaluación de la cooperación en dilemas sociales de gran escala , Pág. 85

FastCGI, y en las bases de datos también pasando por SGBDRs, MySQL, PostgreSQL, SQLite, IBM DB2 hasta llegar a Oracle.

Ruby on Rails hace la escritura de programas más legible al programador ya que sus lemas del inglés “Don’t Repeat Yourself (DRY), Convention over Confi guration, and You Ain’t Gonna Need It (YAGNI)”[46] son especificaciones para esta plataforma que hace que se evite repetición de definiciones (DRY), clases o funcionalidades que se usaran múltiples veces durante el programa, la plataforma asume definiciones por defecto para uso del programa (COC) y evitar hacer implementaciones de escalabilidad por posibles futuras funcionalidades o compatibilidades (YAGNI).

En la siguiente tabla se muestra la actividad de desarrollo del proyecto Ruby on Rails

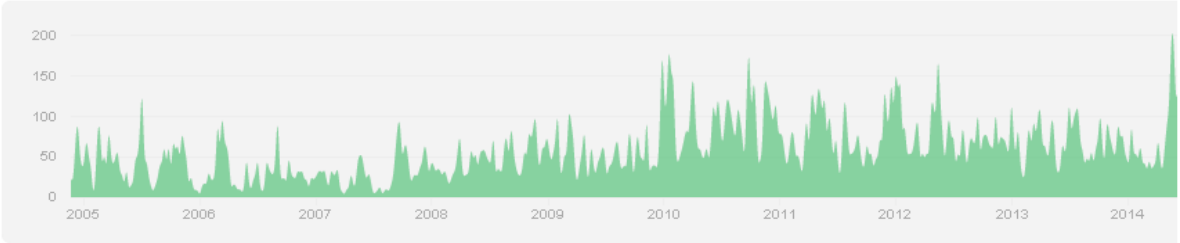

Figura 90 – Tiempo de vida Ruby on Rails vs Cantidad de adiciones

[46] Ver Antonio Cangiano 2009, Ruby on Rails for Microsoft Developers , Pag 44

3.6.1 Algunos historiales de aportación en situación adicionar código

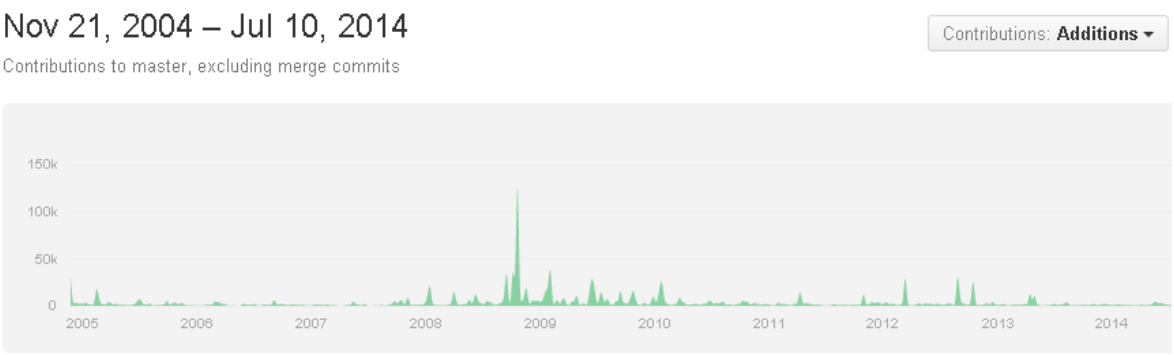

Figura 91 – Tiempo de vida Ruby on Rails vs Cantidad de adiciones

Se puede apreciar que el ritmo de operación adiciones en el proyecto es bastante constante y que en finales del año 2008 se presentó un pico de aportaciones por el orden de los 120.000 coincidente con la salida al mercado de la versión 2.2

Figura 92 – Tiempo de vida Ruby on Rails vs Cantidad de adiciones desarrollador #1

Figura 93 – Tiempo de vida Ruby on Rails vs Cantidad de adiciones desarrollador #2

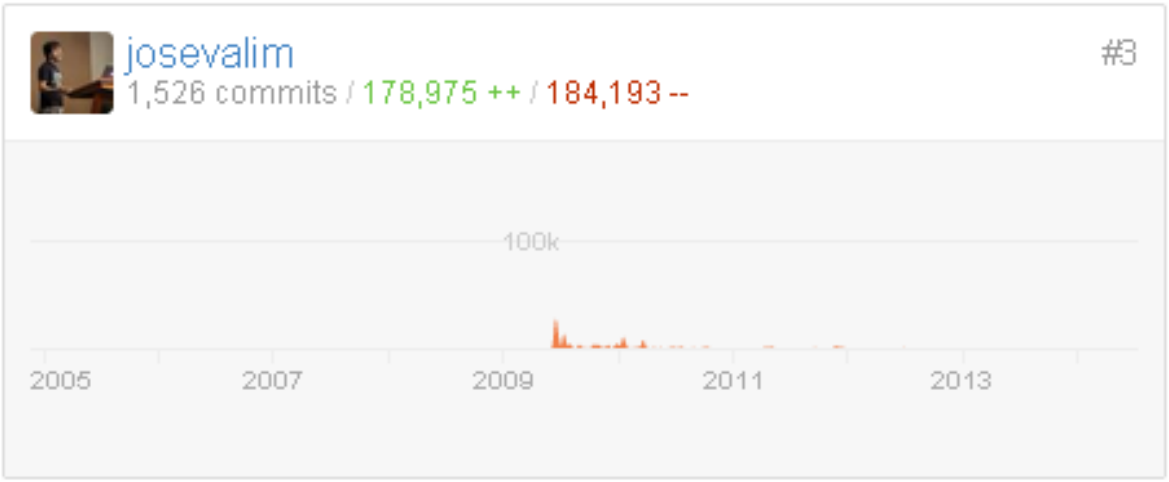

Figura 94 – Tiempo de vida Ruby on Rails vs Cantidad de adiciones desarrollador #3

Figura 95– Tiempo de vida Ruby on Rails vs Cantidad de adiciones desarrollador #4

Figura 96– Tiempo de vida Ruby on Rails vs Cantidad de adiciones desarrollador #5

Figura 97– Tiempo de vida Ruby on Rails vs Cantidad de adiciones desarrollador #6

Comparando las gráficas de los principales miembros se aprecia que son pocos los momentos en que existen un gran número de aportaciones por parte de los miembros que se hayan presentado de forma simultánea en el desarrollo del proyecto siendo producto de situaciones de desarrollo entonces es deducible que las actividades requieren secuencialidad en las tareas o desarrollo de tareas excluyentes entre sí para estos miembros en particular.

3.6.2 Algunos historiales de aportación en situación operación commit

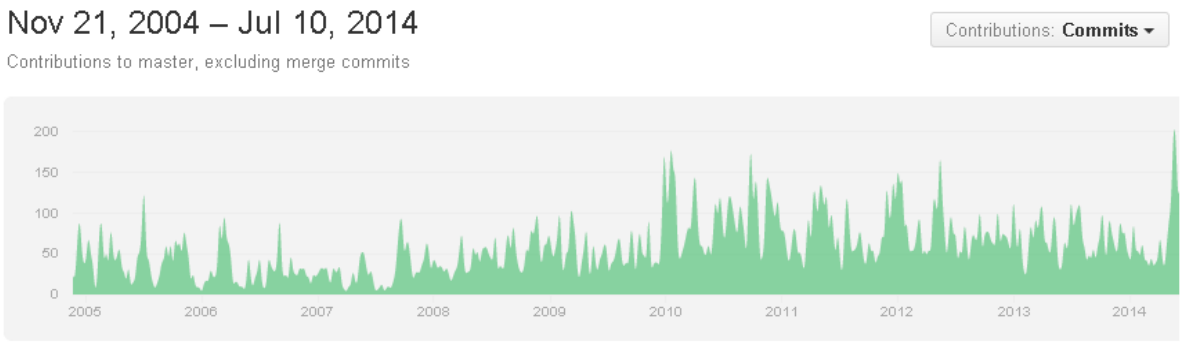

Figura 98– Tiempo de vida Ruby on Rails vs Cantidad de commits

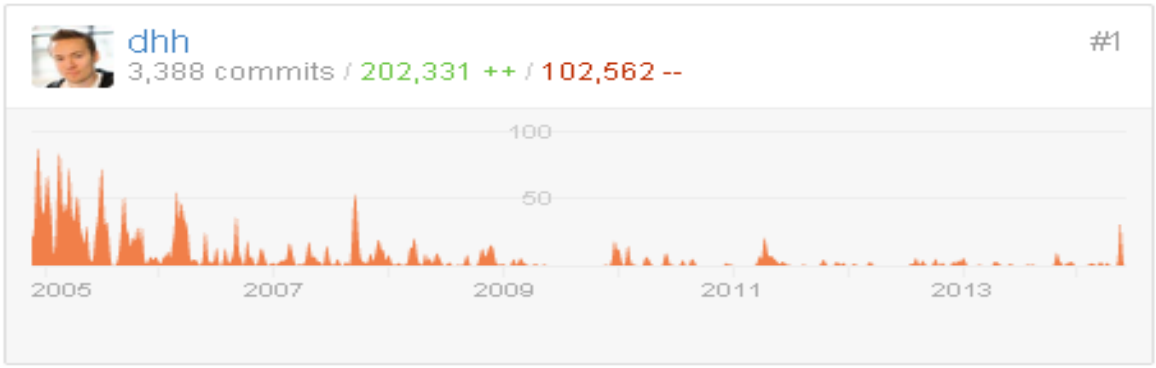

Figura 99– Tiempo de vida Ruby on Rails vs Cantidad de commits desarrollador #1

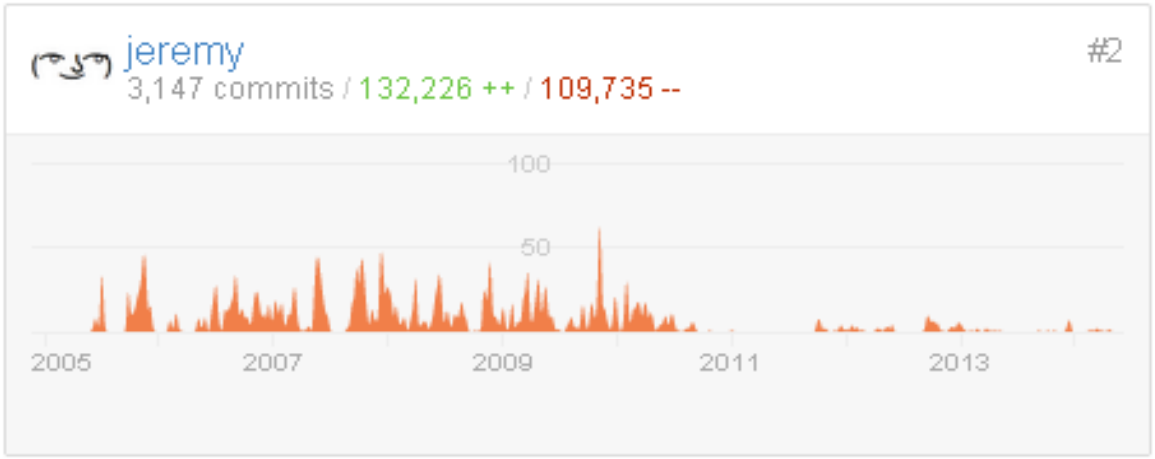

Figura 100– Tiempo de vida Ruby on Rails vs Cantidad de commits desarrollador #2

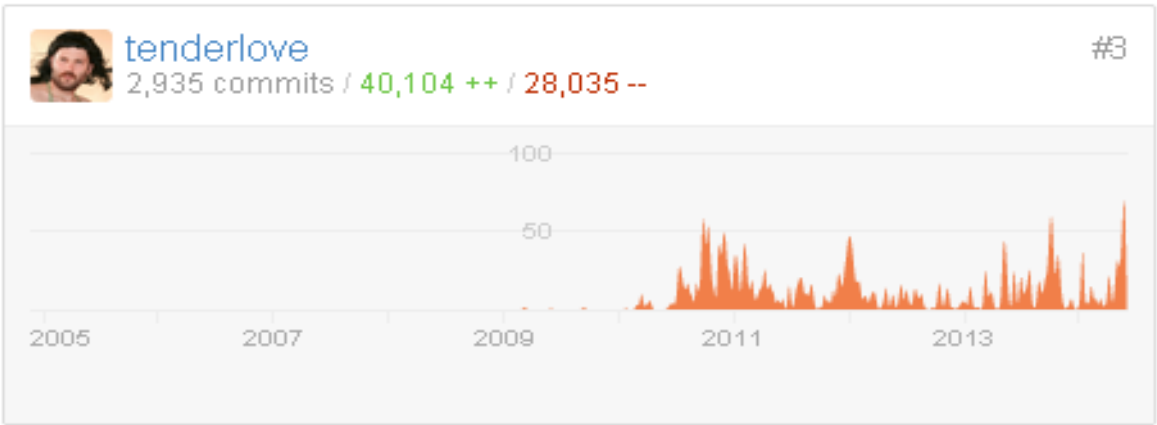

Figura 101– Tiempo de vida Ruby on Rails vs Cantidad de commits desarrollador #3

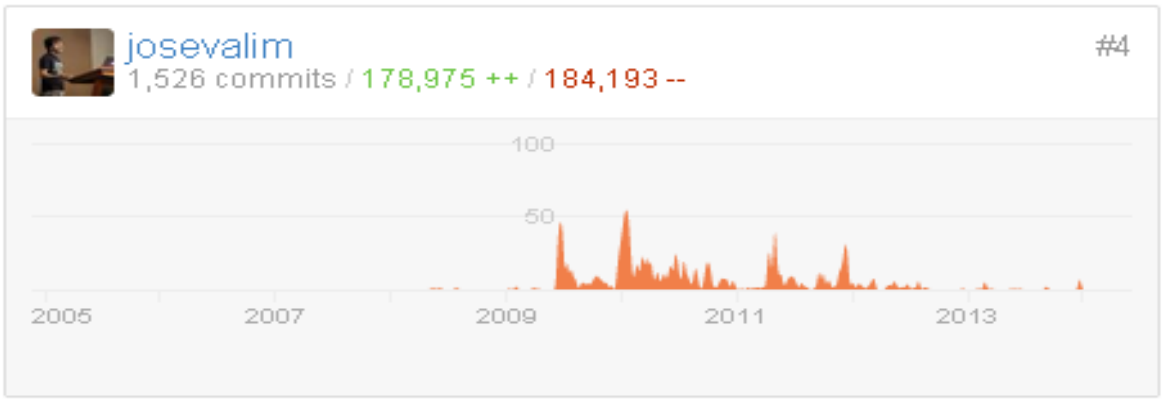

Figura 102– Tiempo de vida Ruby on Rails vs Cantidad de commits desarrollador #4

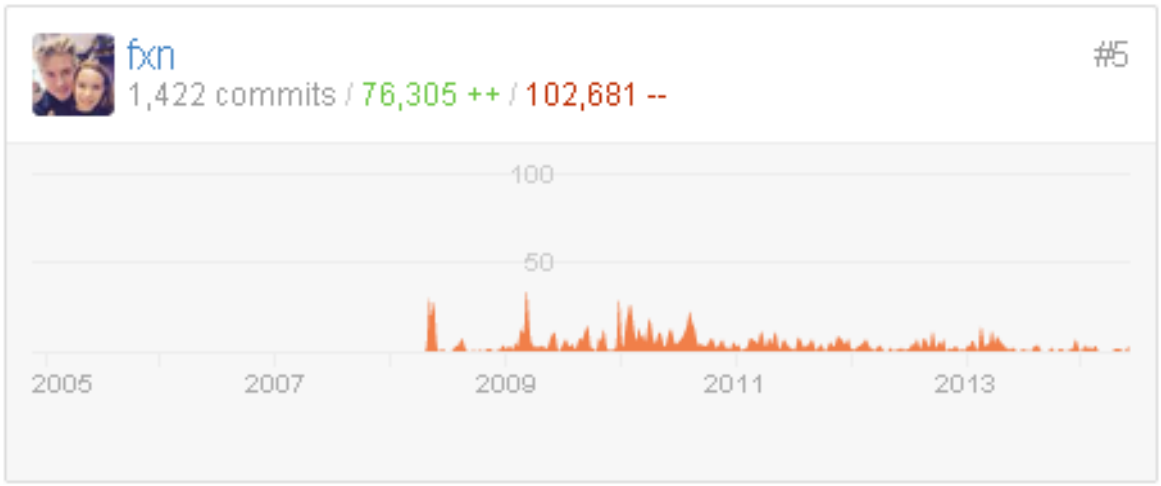

Figura 103– Tiempo de vida Ruby on Rails vs Cantidad de commits desarrollador #5

Figura 104– Tiempo de vida Ruby on Rails vs Cantidad de commits desarrollador #6

En el caso de operación commit el número de aportaciones de los principales miembros pueden llegar a tener una cantidad de aportaciones del orden de 50 o 70 en ciertos periodos de tiempo, y a diferencia de la operación adicionar código al proyecto en la operación commit son varios los momentos de amplio movimiento simultaneo entre los miembros.

3.6.3 Mecanismo de cooperación en caso de estudio Ruby on Rails

En este caso de estudio tenemos dos mecanismo para expresar la forma de cooperar, donde el mecanismo de cooperación basado en confianza, mientras el mecanismo de cooperación por reciprocidad.

Mecanismo de cooperación basada en confianza: este mecanismo se aplica en la situación de la operación adicionar en vista de que las actividades se

desarrollan de forma asíncrona entre los miembros y en el orden de los miles de aportaciones en cada periodo, situación que hace de la confianza un mecanismo efectivo para que los miembros cooperen teniendo la expectativa que el resto de la comunidad cooperara también en aportaciones al proyecto de tal sus mejoras puedan ser observadas por los miembros en periodos de tiempo no inmediatos[47].

Mecanismo de cooperación basada en reciprocidad: es aplicado visiblemente en el caso de operación commit en el cual las aportaciones alcanzan hasta un volumen de 50 y se presentan variados casos de simultaneidad de interacciones en el proyecto de desarrollo, en donde por la misma naturaleza encontrada de simultaneidad en buena cantidad de operaciones[48], es deducible que los miembros desarrollan la operación commit manteniendo un ritmo de trabajo grupal para efectuar pruebas que conduzcan a confirmaciones de código positivas para el proyecto, y a diferencia de las operaciones adicionar evitar códigos de programación que funcionen en la inmediatez pero que afecten otras características indirectamente[49].

[47] Ver Jorge Andrick Parra Valencia, Constructo para la evaluación de la cooperación en dilemas sociales de gran escala , Pág. 84

[48] Ver Kollock 1998, SOCIAL DILEMMAS: The Anatomy of Cooperation, Pág. 196

[49] Ver Jesús M. González Barahona - Joaquín Seoane Pascual - Gregorio Robles, noviembre 2003, UOC Introducción al software libre Pag. 135

3.7 Resumen y conclusiones

Este capítulo ha tomado proyectos representativos del software libre en tres populares géneros como los son los sistemas operativos, sistema de escritorio y sistemas web, para estudiar las interacciones más notables a través de las operaciones adicionar y commit de cada proyecto de forma que sea posible deducir a través del histórico de cada proyecto mecanismos de cooperación por medio de los cuales los miembros cooperantes han contribuido con el proyecto de forma directa, son múltiples y diferenciados en cada caso el comportamiento expresado, dependiendo de variables que son citadas en estos análisis para prestar claridad a la hora de tener en cuenta el comportamiento que representa un mecanismo en particular, algunas de estas son:

- Periodo de vida del proyecto.
- Genero del proyecto de desarrollo.
- Versiones del proyecto de software.

Aunque el número de participantes es en sí misma una variable lo suficientemente determinante, en vista de que este número es bastante alto, en el orden de decenas o miles, para estos casos de estudio no se propone una dependencia de la cooperación en base al número de participantes, se enfoca más bien en sustentar los mecanismos de cooperación más utilizados en cada proyecto de desarrollo según el comportamiento presentado.

En consecuencia del género de software para el cual se hizo este estudio es tener en cuenta que las orientaciones de cada proyecto se deben interpretar con sus objetivos inmediatos, el tipo de usuario-desarrollador al cual va dirigido así:

Proyecto	Objetivo
kernel LINUX	Producción, Mantenimiento y pruebas del software kernel GNU/Linux, orientado al uso de desarrolladores avanzados en sistema GNU/Linux o similares
Berkeley Software Distribution (BSD)	Producción, mantenimiento y pruebas de kernel BSD y sus módulos relacionados, orientado a desarrolladores avanzados en sistemas BSD o similares.
K Desktop Environment (GENTOO / KDE)	Pruebas de paquetes oficiales para el sistema operativo Gentoo sobre el escritorio KDE; estas pruebas normalmente son de compatibilidad, escalabilidad y funcionamiento de los paquetes oficiales del SO Gentoo. Este proyecto está orientado a garantizar las funcionalidades y compatibilidades de software sin distingo de nivel de manejo de sistema operativos o de escritorio.
Joomla Content Management	Desarrollo del sistema de administración de contenidos Joomla, pruebas y soporte, incluyendo el desarrollo del

System	framework propio de esta plataforma web para desarrollo de terceros. Este proyecto de desarrollo de software se orienta en mantener un fácil uso grafico por medio de navegador web para el usuario final, técnico o profesional de sistemas, y de mantener un robusto framework de para usuarios programadores.
PhpMyAdmin	Desarrollo, pruebas y soporte del administrador de base de datos, orientado principalmente hacia el usuario final.
Ruby on Rails	El desarrollo de este proyecto propone funcionalidades bastante avanzadas para el programador promedio, hecho que hace que el desarrollo en sí de este proyecto sea por expertos en el lenguaje Ruby pero también en el manejo del framework y conjunto de funcionalidades ofrecidas por Ruby on Rails, para poder hacer pruebas, implementaciones y correcciones que conduzcan a mejoras en el proyecto.

Tabla 3 – Contexto de los casos de estudio

Estos proyectos tienen por función en caso de Joomla y PhpMyAdmin de servir de facilitadores de funciones para el usuario final de un modo gráfico; los proyectos de kernel FreeBSD y GNU/Linux se enfocan más en la apropiación de nuevas características que puedan prestar mejor aprovechamiento de máquina y permanecer a la vanguardia de los sistemas operativos de tipo servidor; el proyecto Gentoo/KDE es una forma de certificar el buen funcionamiento de los programas utilizados en este entorno, mientras el Ruby On Rails es la

implementación de un lenguaje y ampliación de funcionalidades tambien para desarrollo web.

Se puede comprobar que el crecimiento en número de archivo y líneas es un proceso lento que requiere un esfuerzo mantenido en el tiempo por la comunidad de desarrolladores que cooperan en cada proyecto, los anexos dejan por sentado el hecho de que la diferencia entre la cantidad de operaciones adicionar y eliminar por miembro en cortos periodos y aun en largos periodos es despreciable, no asi las operaciones commit para las cuales se puede analizar una independencia con las operaciones adicionar y eliminar, pero si una dependencia directa de la expectativa de producción de alguna versión de software en particular, en especial si la versión implica un cambio muy valorable en la vida del proyecto de software.

4. Propuesta de lineamientos

4.1 Objetivo de las propuestas de mecanismos de cooperación

Presentar lineamientos estratégicos que permitan a los proyectos de software libre mitigar la no cooperación que acontecen en la vida de desarrollo de los mismos, algunas de las situaciones comunes que obstaculizan el crecimiento o normal ejercicio cooperativo en los proyectos de software libre que se pueden encontrar comúnmente son:

- Alta deserción
- Bajo número de aportaciones
- Extenso periodo de inactividad

En la búsqueda de plantear lineamientos que puedan ser efectivos se busca principalmente que la cooperación sea el mecanismo que impulse el crecimiento de los proyectos de software libre y de esta forma se puedan mantener en el mercado presentando cualidades iguales o mejores a las ofrecidas por software propietario[50], siendo aparte de una opción de bajo coste, también una opción atractiva por otras características como robustez, escalabilidad, compatibilidad y seguridad.

En esta investigación se buscó esquematizar algunos mecanismo de cooperación presentes en los casos de estudio tomando como base el comportamiento observado en las comunidades de desarrollo en grandes periodos de tiempo (años) o aun durante todo el tiempo de vida que ha tenido el proyecto de

[50] Dirk Vermeir 2005, How Open is the Future?, Pag 107

desarrollo en sí, teniendo en cuenta que son múltiples las actividades que se llevan a cabo en un proyecto de desarrollo, siendo que en algunos casos estas tareas son totalmente concurrentes[51] en un número determinado de miembros de los cuales no se pudo hacer medición, tareas desarrolladas de forma síncrona en un proyecto, asíncronas en otros casos y/o secuenciales; estas cuatro situaciones se asumieron por simplicidad y limitación en la definición de datos adquirida como de desarrollo cooperativo síncrono de tareas y como desarrollo cooperativo asíncrono, agrupando correspondientemente las cuatro anteriores.

Para este investigación se tuvieron en cuenta casos representativos de desarrollo de software libre para observar mecanismos de cooperación que estuvieran en ejercicio en la vida del proyecto o en largos periodos de tiempo (años), los proyectos estudiados poseían ya datación y estadísticas de comportamiento presentes como parte de la vida del proyecto, en ningún caso de estudio se presenta la situación de inicio y final de vida de desarrollo de un caso de estudio en particular cuestión que limita notablemente las condiciones de aplicación de estos lineamientos porque no se asume en ningún caso condiciones iniciales de confianza cero o similares; ya que las observaciones demuestran el ingreso de nuevos miembros a los proyectos de desarrollo y su permanencia dentro de los mismos por demostrar actividad de aportaciones dentro del proyecto se asume el mejoramiento en la competitividad[52], el compromiso en cooperar dentro del proyecto y la satisfacción con las labores de desarrollo.

Estas propuestas de lineamientos están fundamentadas en los estudios de comportamiento que presentan los casos de estudio, siendo estas interacciones con el sistema de manejo de versiones y la actividad demostrada por los miembros a través del tiempo, los datos necesarios para fundamentar

---

[51] Ver Karl Fogel, Producing Open Source Software, How to Run a Successful Free Software Project, Pag vi

[52] Geert Hofstede Gert Jan Hofstede Michael Minkov 2010, Cultures and Organizations, Pág. 358

mecanismos de cooperación dentro de los proyectos de desarrollo de software libre, que puedan ser sustentados y estudiados con base a la teoría de cooperación.

4.2 Caracterización de la cooperación en los casos de estudio

En los casos de estudio se tienen características sistémicas en la cooperación que deben ser nombradas para entender el contexto de los comportamientos comunitarios e individuales para los cuales se efectúa esta propuesta de lineamientos basados en la teoría de la cooperación.

En los siguientes puntos se ofrecen varias apreciaciones a una serie de características del dilema social de cooperación en el desarrollo de proyectos de software libre, para demostrar una clara relación con dilemas sociales de pequeña escala y más directamente con dilemas sociales de recurso agotable de gran escala. Teniendo en cuenta las versiones de software como un recurso agotable de gran escala, porque si el número de aportaciones y desarrolladores cae vertiginosamente, una extensión en el tiempo deja de manera palpable una determinada versión de software obsoleta para los usuarios finales que hacen utilización, y de forma similar para un número reducido de desarrolladores el mantenimiento y desarrollo de una versión de software sería tan asfixiante que se verían en la obligación de migrar a otra proyecto de desarrollo de software libre, o dejar por completo de cooperar y adoptar una solución de software propietario.

Contexto cooperativo: El contexto que ofrece el caso de estudio es puramente practico y de campo, se hace observación sobre el sistema gestor de versiones que es el encargado de registrar un histórico del proyecto de operación adicionar, eliminar y commit; también el número de aportaciones producidas por los miembros con más volumen de aportaciones en el tiempo total de vida del proyecto registrado por el manejador de versiones, la distribución geográfica de los miembros no es tenida en cuenta por el gran número de estos en cada proyecto, cuestión que los hace difícilmente localizables.

Tamaño grupo: el número de desarrolladores que colaboran en cada proyecto objeto de los casos de estudio, varía desde las decenas de integrantes de forma directa hasta los miles, cuestión que hace de este estudio una primera mirada de estas situaciones de cooperación en el software libre.

Características grupo: en los casos de estudio se presenta la dualidad de tener grandes subgrupos de miembros cooperantes con homogéneas capacidades para la generación de aportaciones en cada proyecto tales como programación, traducción, probadores, usuarios finales; como también heterogéneas características de estos mismos aun dentro de los subgrupos mencionados, están características van desde ocupación, nacionalidad, idioma hasta el grado de estudios.

Magnitud retardo: En los proyectos de software libre objeto de este estudio el retardo en la representación de aportaciones normalmente difiere según el casos de estudio, teniendo en cuenta la periodicidad de salida al mercado de las versiones de software en las cuales se trabaja se pueden deducir los siguientes tiempo de retardo para cada proyecto de desarrollo de software libre asi:

- kernel GNU/LINUX → nuevas versiones semanales, incluye cambios de versión, secuencias de versión, actualización o parches.
- Berkeley Software Distribution (BSD) → nuevas versiones mensuales, secuencias de versión, actualización y parches.
- K Desktop Environment (GENTOO / KDE) → Diarias, este grupo de desarrollo se ocupa de soporte y pruebas de compatibilidad hecho que se puede ver reflejado inmediatamente en cualquier sistema operativo Gentoo/KDE con la opción de actualización y corrección automática.
- Joomla Content Management System: con un periodo quincenal, nuevas versiones, secuencias de versión, mejoras de compatibilidad, actualización y parches.
- PhpMyAdmin: con un periodo de dos días, nuevas versiones, corrección de errores de versión, secuencias de versión, mejoras de compatibilidad, actualización y parches.
- Ruby on Rails: con un periodo de una semana, nuevas versiones, corrección de errores de versión, secuencias de versión, mejoras de compatibilidad, actualización y parches.

Calidad realimentación: en los proyectos de desarrollo de software libre esta cualidad es realmente baja, en los momentos de desarrollo para la realización de una versión final o salto de versión, y en las actividades de corrección de errores es demasiado costoso realizar pruebas de caja blanca y caja negra completas para el programa o fragmento del mismo, se toma como positiva la superación de ciertas condiciones de funcionalidad, entrada y salida. Es así como en periodos largos de tiempo se presentan comportamientos erróneos del software, cuando este es sometido a variadas situaciones de operación.

Modelo racionalidad: para los casos de estudio analizados en esta investigación se deduce un modelo de racionalidad acotada sustentado en el hecho de que los desarrolladores que cooperan en un proyecto de software libre pueden observar

de forma retardada un comportamiento en proyecto de software según las aportaciones grupales de la comunidad de desarrollo, este comportamiento observable se puede discriminar en número de bajadas del software (popularidad), cantidad de versiones recientes, mejoras significativas de funcionamiento y llegar también a la comparación en todos los sentidos con otros proyectos de software libre o propietario.

Encuentros: el número de encuentros es infinito y continuo en los periodos de tiempo que dure el miembro cooperante en el proyecto de desarrollo.

Comunicación: Aunque la comunicación no es físicamente frente a frente, las interacciones y el tiempo de respuesta, prueba que casi lo es, ya que las respuestas en foros, listas de soporte y otros sistemas de comunicaciones poseen un tiempo relativamente bajo (del orden de horas o días) comparado con los periodos de desarrollo de una versión completa o el tiempo de vida de un proyecto de software libre (meses o años).

En la siguiente tabla se listan los mecanismos de cooperación encontrados para cada caso de estudio analizado en el capítulo anterior, en búsqueda de construir una síntesis de los mecanismos de cooperación más comunes utilizados en el desarrollo de proyectos de software libre.

	Mecanismo encontrado
kernel LINUX	*Mecanismo de cooperación basado en confianza:* El crecimiento del número miembros, el largo tiempo de vida[53] y la adaptación de nuevas características; Estos y otros factores dan confianza a el desarrollador cooperador del proyecto, manteniendo su reciprocidad y obteniendo reputación por el proyecto mismo[54].

[53] Ver Robert Love 2010, Linux Kernel Development, Pag. 3

[54] Ver Jorge Andrick Parra Valencia, Constructo para la evaluación de la cooperación en dilemas sociales de gran escala , Pág. 21

Berkeley Software Distribution (BSD)	*Mecanismo de cooperación basado en confianza:* se presentan retardos en la retoma del ritmo de desarrollo después de pausas visibles, implicando que la confianza es incremental, saliendo de la reducción del ritmo de aprendizaje, subiendo el volumen de desarrollo[55] y mejorando la confianza y reciprocidad entre los miembros de la comunidad[56].
K Desktop Environment (GENTOO / KDE)	*Mecanismo de cooperación por aprendizaje social:* el aprendizaje en este proyecto está orientado a aseguramientos (commits) de cambios del código fuente[57], sin que la interacción con otros miembros como parte del desarrollo sea esencial. *Mecanismo de cooperación por estructura de pagos:* la colaboración se incrementa porque el miembro puede demostrar y usar tangiblemente la mejora efectuada[58].
Joomla Content Management System	*Mecanismo de iteración e identificabilidad:* En el periodo 2009, la versión 1.6 tuvo los mayores avances tomando características de las otras ramas de desarrollo[59]. Gracias a las múltiples interacciones con el sistema de control de versiones github.com sustentable también por que 22 de los miembros más activos pasaron por el periodo 2008[60].

[55] Ver Jorge Andrick Parra Valencia, Constructo para la evaluación de la cooperación en dilemas sociales de gran escala , Pág. 43

[56] Ver Kollock 1998, SOCIAL DILEMMAS: The Anatomy of Cooperation, Pág. 199

[57] Kollock 1998, SOCIAL DILEMMAS: The Anatomy of Cooperation, Pág. 199

[58] Kollock 1998, SOCIAL DILEMMAS: The Anatomy of Cooperation, Pág. 200

[59] Kollock 1998, SOCIAL DILEMMAS: The Anatomy of Cooperation, Pág. 199

[60] Ver Anexo C - Estadisticas commit proyecto Joomla-CMS

PhpMyAdmin	*Mecanismo de cooperación basado en confianza:* El proyecto de desarrollo hasta el periodo 2010 no ha tenido grandes volúmenes de aportaciones y se mantuvo casi constante sin pausas largas que hubiesen bajado la apropiación de conocimientos a bajos niveles. *Mecanismo de cooperación por aprendizaje social:* En los periodos 2001 y 2010 se observa entrada de nuevos desarrolladores[61], y seguidamente aumento en aportaciones con estabilidad de miembros[62].
Ruby on Rails	*Mecanismo de cooperación basada en confianza*: las aportaciones de los demás miembros son normalmente desarrolladas y probadas de forma asíncrona[63]. *Mecanismo de cooperación basada en reciprocidad:* presenta simultaneidad en la aplicación de operaciones commit[64], producen largas pruebas para los commit evitando funcionalidad en la inmediatez y futuros errores[65].

Tabla 4 – Hallazgos de mecanismo de cooperación

4.3 Efectividad de los mecanismos de cooperación estudiados

[61] Anexo E - Estadisticas commit proyecto PhpMyAdmin

[62] Ver Jorge Andrick Parra Valencia, Constructo para la evaluación de la cooperación en dilemas sociales de gran escala , Pág. 85

[63] Ver Jorge Andrick Parra Valencia, Constructo para la evaluación de la cooperación en dilemas sociales de gran escala , Pág. 84

[64] Ver Kollock 1998, SOCIAL DILEMMAS: The Anatomy of Cooperation, Pág. 196

[65] Ver Jesús M. González Barahona - Joaquín Seoane Pascual - Gregorio Robles, noviembre 2003, UOC Introducción al software libre Pag. 135

En la construcción de lineamientos de cooperación que puedan ser lo suficientemente benéficos en los proyectos de desarrollo de software libre es necesario el análisis de factores que puedan fundamentar la credibilidad de los lineamientos, de forma que estos puedan ser inferidos válidamente en otros proyectos de desarrollo de software libre en los cuales las condiciones situacionales o de propósito sean similares a los casos de estudio estudiados en esta investigación.

Los siguientes factores se estudian para cada caso de estudio de tal modo se exponga clara idea de las condiciones históricas de cada caso de estudio que demuestren la factibilidad de los mecanismos de cooperación para el crecimiento y mantenimiento de los proyectos de desarrollo de software libre.

4.3.1 Superación de condiciones de desconfianza

Se describe a continuación si las condiciones de desconfianza afectan los mecanismos encontrados en los diferentes casos de estudio a tal punto de producir situaciones de baja cooperación al interior del proyecto de desarrollo.

	Mecanismo	**Supera condiciones iniciales de desconfianza**
kernel LINUX	*Mecanismo de cooperación basado en confianza*	Sí: se puede probar esto por los bajos valles de aportaciones, que se remontan después de un corto periodo de tiempo
Berkeley Software Distribution (BSD)	*Mecanismo de cooperación basado en confianza*	Sí: se puede probar esto por los bajos valles de aportaciones, que se remontan después de un corto periodo de tiempo
K Desktop Environment (GENTOO /	*Mecanismo de cooperación por aprendizaje social*	Sí: los periodos de pausas se ven superados por miembros nuevos que ven atractivo el aprendizaje y la practica en este proyecto.

KDE)	*Mecanismo de cooperación por estructura de pagos*	Sí: el hecho de poder aprovechar el software producido en desarrollos propios hace que exista una confianza en recibir esto como retribución por la cooperación ejercida.
Joomla Content Management System	*Mecanismo de iteración e identificabilidad*	Sí: La confianza se basa en la gran cantidad de soporte que hace que cooperar en este proyecto sea garantía de cambios y adaptaciones del mismo
PhpMyAdmin	*Mecanismo de cooperación basado en confianza*	Sí: se puede probar esto por los bajos valles de aportaciones, que se remontan después de un corto periodo de tiempo
	Mecanismo de cooperación por aprendizaje social	Sí: los periodos de pausas se ven superados por miembros nuevos que ven atractivo el aprendizaje y la practica en este proyecto.
Ruby on Rails	*Mecanismo de cooperación basada en confianza*	Sí: se puede probar esto por los bajos valles de aportaciones, que se remontan después de un corto periodo de tiempo
	Mecanismo de cooperación basada en reciprocidad	No se puede probar en operación commit ya que se presentan varios trabajos simultáneos, no hay pausas prolongadas en esta operación.

Tabla 5 – Pertinencia de los mecanismos

4.3.2 Sostenibilidad de la cooperación

Se muestran a continuación los casos de sostenibilidad de los mecanismos de cooperación encontrados, examinando si en la vida del proyecto de desarrollo existen periodos de dejamiento total que hayan puesto el proyecto en posible situación de finalización o descontinuación definitiva.

	Mecanismo	Sostenibilidad de la cooperación
kernel LINUX	*Mecanismo de cooperación basado en confianza*	Sí: el proyecto no presenta cortes de aportaciones o versiones abandonadas de soporte.
Berkeley Software Distribution (BSD)	*Mecanismo de cooperación basado en confianza*	No es probable, el proyecto presenta bajas significativas en los siguientes periodos: Agosto a Octubre de 2007 Julio a Agosto de 2009 Julio a Septiembre de 2011
K Desktop Environment (GENTOO / KDE)	*Mecanismo de cooperación por aprendizaje social*	Si: se puede apreciar un amplia cantidad de operaciones commit durante la vida del proyecto, también el hecho de que los miembros más aportantes rodean una antigüedad de 2 años.
	Mecanismo de cooperación por estructura de pagos	Sí: se puede deducir el incentivo de lograr que cierto programa funcione en esta plataforma.
Joomla Content Management System	*Mecanismo de iteración e identificabilidad*	Sí: aunque existen múltiples bajas de cooperación en la rama principal del proyecto se ha demostrado que estas bajas se corresponden con alto volumen de aportaciones en las ramas secundarias como parte del desarrollo de otras versiones
PhpMyAdmin	*Mecanismo de cooperación basado en confianza*	Sí: el proyecto no presenta cortes de aportaciones o versiones abandonadas de soporte.
	Mecanismo de cooperación por aprendizaje social	No: el proyecto presenta varios picos importantes en el periodo 2010 a abril de 2012, sin embargo después de este periodo el promedio no se mantuvo y el número de miembros aportantes ha decrecido a un nivel casi igual a periodos anteriores al 2010.
Ruby on Rails	*Mecanismo de cooperación basada en confianza*	Sí: el proyecto no presenta cortes de aportaciones o versiones abandonadas de soporte.
	Mecanismo de cooperación basada en reciprocidad	Sí: se presentan multitud de casos de sincronía en aportaciones, independientemente del volumen total en el proyecto.

Tabla 6 - Sostenibilidad de la cooperación

4.3.3 Realimentación de información

En los casos de estudio analizados en el presente trabajo de investigación existe disponibilidad de la información para las diferentes actividades de desarrollo algunas de estas situaciones de disponibilidad son:

Foros de discusión: En estos foros se discuten características necesarias en las implementaciones del software.

Foros de soporte: Estos foros asignan responsabilidad en la solución de problemas a desarrolladores o expertos cooperadores del proyecto.

Versiones publicadas: En el software libre se puede acceder fácilmente a versiones de software de prueba (alfa, beta…) como a versiones estables (finales, largo soporte…), para cualquier usuario; las versiones de software producidas y publicadas son el factor de realimentación más notable en los proyectos de software libre, de esta situación el desarrollador puede apreciar la aceptación y calidad de su trabajo.

Listas de comunicación: Normalmente los proyectos de software libre poseen listas de correos electrónico según las distintas categorías en que se establezca el equipo, por ejemplo gentoo-user@lists.gentoo.org para soporte general al usuario y discusiones, gentoo-announce@lists.gentoo.org para anuncios generales de Gentoo gentoo-dev@lists.gentoo.org para discusiones generales entre desarrolladores Gentoo.

4.3.4 Cooperación como norma

En los proyectos de desarrollo de software libre es necesaria la sostenibilidad de la cooperación, esta se debe general como norma en cada comunidad de desarrollo para que se manifieste en un conjunto de procesos que puedan enfrentar una crisis pasajera como una serie de obstáculos a través de la vida del proyecto mismo. En este cuadro se mencionan los procesos que hacen de la cooperación una prioridad en los proyectos de software libre

	Mecanismo	**Existencia de la cooperación como norma**
kernel LINUX	*Mecanismo de cooperación basado en confianza*	Si: los miembros tienen presente procesos tanto integrales del proyecto como afectaciones globales que se pudiesen presentar por sus aportaciones. También poseen páginas para promover capacitación, el alistamiento, las pruebas de desarrollo, el reporte de errores entre otros.
Berkeley Software Distribution (BSD)	*Mecanismo de cooperación basado en confianza*	Si: los miembros tienen presente procesos tanto integrales del proyecto como afectaciones globales que se pudiesen presentar por sus aportaciones. También poseen páginas para promover capacitación, el alistamiento, las pruebas de desarrollo, el reporte de errores entre otros.
K Desktop Environment (GENTOO / KDE)	*Mecanismo de cooperación por aprendizaje social*	Si: Aunque comprende los mismos procesos para promover la cooperación, en relación a los otros proyecto de desarrollo se *distingue por hacer correcciones al propio software del proyecto y también a paquetes externos o librerías* adicionales, tareas para las cuales no existen procedimientos formalizados por la gran cantidad de paquetes y versiones.
	Mecanismo de cooperación por estructura de pagos	No es probable: En vista de que el desarrollador tiene en la versión de software publicada una retribución y muestra de su aportación al proyecto, existen desarrolladores pasajeros que llegan motivados por mejorar la compatibilidad de un paquete en particular y ven poca retribución cuando han solucionado el problema del paquete de su interés.
Joomla Content Management System	*Mecanismo de iteración e identificabilidad*	Si: En las situaciones abordadas se pudo comprobar que cuando existen bajas de cooperación por subdividir el proyecto en varias ramas de desarrollo, las cooperaciones se mantienen a tal punto de incrementar las aportaciones para una próxima versión con enriquecimiento de características.

PhpMyAdmin	*Mecanismo de cooperación basado en confianza*	Si: los miembros tienen presente procesos tanto integrales del proyecto como afectaciones globales que se pudiesen presentar por sus aportaciones. También poseen páginas para promover capacitación, el alistamiento, las pruebas de desarrollo, el reporte de errores entre otros.
	Mecanismo de cooperación por aprendizaje social	Si: los desarrolladores implementan mejoras relacionadas a versiones en particular del servidor web, del motor de base de datos y de errores reportados, cuestiones no demasiado extensas y que dan pie a tener poca rotación de membresía casi constante volumen de aportaciones.
Ruby on Rails	*Mecanismo de cooperación basada en confianza*	Si: los miembros tienen presente procesos tanto integrales del proyecto como afectaciones globales que se pudiesen presentar por sus aportaciones. También poseen páginas para promover capacitación, el alistamiento, las pruebas de desarrollo, el reporte de errores entre otros.
	Mecanismo de cooperación basada en reciprocidad	Si: los sistemas manejadores de versiones son una parte del proceso de cooperación que se ocupa no solo de hacer ordenada la sincronía de labores, también de que la concurrencia produzca errores.

Tabla 7 - Cooperación como norma

4.3.5 Enfrentar deserción

En los proyectos de desarrollo de software libre con gran número de desarrolladores es casi despreciable el caso de que pocos desarrolladores abandonen la comunidad de desarrollo, no es el caso para proyecto de desarrollo en los cuales el número de desarrolladores es bastante bajo (aproximado a 10 miembros); en los casos de estudio que se analizan en este trabajo de investigación es una situación normal el abandono de la comunidad por parte de los miembros, teniendo en cuenta que el flujo diferencial de personas que ingresan

y desertan de los proyectos de desarrollo en los casos de estudio citados se mantiene casi constante con tendencia al alza en todos los casos.

Se puede notar en los casos de estudio y en las comunidades de software libre en general que los principales incentivos para pertenecer y no desertar de un proyecto de desarrollo son[66]:

Hacer parte activa de una comunidad de desarrollo

Obtener experiencia

Mejorar habilidades

Mejorar características de un software

Sin que estos sean los únicos motivantes para permanecer en un proyecto de software libre, se puede entender la deserción en los proyectos de software libre como la finalización de un miembro desarrollador de participar en el proyecto por haber satisfecho su objetivos motivantes para cooperar en el desarrollo.

4.3.6 Percepción de daño

En los proyecto de software libre por la naturaleza de la realimentación en las actividades efectuadas por la comunidad de desarrollo para una versión en particular la percepción de daño es una característica que es comprobable por los miembros de forma práctica por que varios factores pueden ser la prueba principal

[66] Ver Jesús M. González Barahona - Joaquín Seoane Pascual - Gregorio Robles, noviembre 2003, UOC Introducción al software libre Cap 4.

de mejoramiento o deterioro del proyecto, algunos de estos factores de calidad[67] según norma ISO 9126 son:

- Funcionalidad
- Confiabilidad
- Usabilidad
- Eficiencia.
- Factibilidad de mantenimiento
- Portabilidad.

Para propósitos de esta investigación probar bajas súbitas de aportaciones de desarrollo (cercanas a cero) en algunas periodos como resultado de decremento en la aceptabilidad de los factores de calidad antes citados es improbable por cuestiones que atañen de cierta forma a la naturaleza de desarrollo del software libre; Estas dificultades inmediatas se explican a continuación:

Medición de factores de calidad en el software libre: Los factores de calidad en el software libre no son medidos continuamente, la apreciación de mejoras en estas características están primariamente indicadas por la aceptación de mejoras en las versiones y el conjunto de operaciones que esto comprende como actualizaciones, instalación de parches, compatibilidad con paquetes de software entre otros no están sincronizados de forma permanente, así una versión de software es usada en un periodo de tiempo, pero también en otros periodos de tiempo implicando esto demasiada complejidad a la hora de determinar las consecuencias de aportaciones por parte de un desarrollador. Sin embargo en todos los casos de estudio de esta investigación es notable la subida no regular de colaboradores, de líneas del proyecto de desarrollo.

---

[67] Roger S. Pressman 5a Ed, Ingeniería del Software un Enfoque Práctico, Pág. 326

Aportaciones cercanas a cero: No existen periodos de tiempo en que las aportaciones sean cercanas a cero dentro de un proyecto de desarrollo, en los casos de estudio analizados, cuestión que ha hecho de la cooperación en los proyectos anteriores un recurso siempre disponible; al igual que el software producido en los casos analizados permanece con algún tipo de licencia GPL.

Escases del recurso: El cambio de licenciamiento en los casos de estudio puede significar en un momento de vida del proyecto la eliminación de libertades que los desarrolladores mantengan como motivantes principales en su actividad, el uso de licencias GPL y GPL v2 en los proyectos de desarrollo que se analizan en este estudio producen como consecuencia fundamental en la continuidad de vida de los proyectos de desarrollo de software libre que estos paquetes de versiones subsiguientes se mantengan como software libre[68]; en particular para el caso de estudio del sistema operativo FreeBSD no se han registrado periodos en los cuales las aportaciones o la disponibilidad de software sea cercana a cero y con menos posibilidad como consecuencia por relación a cambios en el licenciamiento.

4.4 Mejorar la cooperación

Los lineamientos que propone este estudio están fundamentados en la identificación de procesos, mecanismos y naturaleza del desarrollo del software libre para los que se puede demostrar su incidencia positiva en los proyectos de software libre. Por otro lado también se proponen lineamientos en los que su

[68] Karl Fogel, Producing Open Source Software, How to Run a Successful Free Software Project, Pag 21

prueba práctica es la necesidad de algún nivel de mejoramiento en aspectos ya vistos durante los análisis de los casos de estudio.

Existen gran cantidad de variables que pudiesen afectar en alguna medida el desarrollo de la cooperación en los proyectos de software libre, como calidad de estos análisis para sustentar la propuesta que se realiza se tienen en cuenta condiciones visibles fácilmente de los casos de estudio como lo es la historia de aportaciones en operaciones adicionar y en operaciones commit, el historial de versiones y los periodos de vida de cada proyecto, esto con el fin de lograr deducir de forma clara en cada caso de estudio los mecanismos utilizados para hacer frente a problemas como la falta de confianza, la tentación de desertar y la escases de recurso en el proyecto de desarrollo.

Proponer lineamientos como mejoras en el dilema de la cooperación en el software libre no resulta ser una mera apreciación profunda sobre el comportamiento de un proceso de desarrollo de software libre, ni la forma lineal de ver como suceden los acontecimientos de cohesión de esta comunidad mundial a través de la historia reciente de la informática. Una gran parte de la descripción de este dilema social en la cual se ve enfrentado el modelo de propiedad y producción privada con la gestión e interés comunitario debe de probarse desde el punto de vista de quienes producen el software libre, siendo estos los principales actores y en la mayoría de casos los principales consumidores. No siempre como podría pensarse a priori son personas dedicadas al mundo de la programación o de los sistemas operativos, se debe de acoger en la concepción de “cooperación en el software libre” la comunidad como el conjunto de personas de diferentes características, pero también las empresas con intereses explícitos e implícitos en la explotación de este recurso comunitario, entendiéndose empresas, como del sector privado y también del sector público[69].

[69] Ver Jesús M. González B., Joaquín S. P., Gregorio R. 2000, Software libre 2ª Ed. Febrero 2008, Pág. 70

La relación confianza – reciprocidad – reputación es de dinámica cíclica, proporcional y en los casos de estudio analizados el proceso enmarcado evoluciona a través del tiempo, las variables a su vez presentan incrementos proporcionales al conjunto, y siendo que en el software libre el objetivo es sacar a la luz pública como producto terminado un programa de computador con las cualidades y funcionalidades que se tenían previstas, el proceso de maduración del software libre es una fase colaborativa que tiende a estabilizar de forma casi constante las interacciones colaborativas que responden al mejoramiento y resolución de fallas del mismo.

4.5 Elementos de la propuesta

En los siguientes puntos se proponen lineamientos que buscan mejorar la cooperación en los proyectos de desarrollo y corregir posibles errores o vacíos observados durante el análisis.

4.5.1 Promover la llegada de nuevos miembros

Aunque los proyectos de software libre se nutren continuamente de personas nuevas para el proyecto de desarrollo, la promoción en sí, no es un asunto del todo efectivo, pues por lo general los nuevos miembros buscan pertenecer a la comunidad de desarrollo por alguna razón sea personal o no. Aunque una buena parte de la promoción que efectúan los proyectos de software libre ofrece una

serie de facilidades para aprender sobre el contexto, la promoción puede ser entendida también como el ofrecimiento de capacidades y beneficios a los nuevos miembros, así que:

> *Proveer al nuevo miembro en un proyecto de software libre, de múltiples beneficios personales en la cooperación para el desarrollo del proyecto.*

Algunos proyecto de desarrollo implementan procesos en los que hacen uso de este lineamiento:

Joomla CMS: en el cual se pueden ver varios tipos de documentos que inicia al nuevo desarrollador en mejoras de funcionamiento en la sección *Developers (https://docs.joomla.org/Portal:Developers),* sin embargo no se hacen alusiones a los beneficios personales que puede obtener un desarrollador según los conocimientos particulares que posea, los cuales pueden ir desde el aspecto económico, laboral entre otros.

4.5.2 Aprovechar las bifurcaciones

En los proyectos que han tenido ramas de desarrollo secundarias con suficientes características novedosas o diferentes de la rama principal es predecible un miedo a la independización de esa rama como un nuevo proyecto de software libre, atrayendo miembros de la rama principal y minando la cooperación en el proyecto (*por ej: la distribución Debian es una con más cantidad de forks).* No siempre es el caso, con algunas proyectos de software libre vistos en los casos de estudio anteriores como FreeBSD y PhpMyAdmin se analizaron bajas de producción de aportaciones en algunos periodos pero observando la historia de versiones se pudo comprobar que existían bifurcaciones de desarrollo del árbol principal del

proyecto, cuestión que producía en la vida del proyecto un salto de versión con nuevas características más refinadas que las versiones anteriores. Entonces el aprovechamiento de esta comportamiento en la naturaleza de desarrollo se puede potenciar si

> *Unificar, compatibilizar o eliminar características de un proyecto de software libre en una versión acumulativa fortifica la cooperación en la comunidad.*

El hecho de poder asociar a la rama principal de desarrollo factores de calidad ya implementados por las ramas secundarias que den origen a mejoras de funcionalidad, confiabilidad y usabilidad; No solamente hacen de la rama principal del proyecto una mejor versión que haga obligatoria la incorporación de los miembros alejados, también hace un proyecto con más integridad, más rico y menos diferenciado de lo que podría ser un nuevo proyecto nacido de una rama secundaria de desarrollo.

4.5.3 Implementar estructura de reconocimiento

Si bien los proyectos de desarrollo de software libre se distinguen por la gran cantidad de aportaciones anónimas de las cuales se componen, poder potenciar el reconocimiento tal vez no sea un lineamiento que conduzca a hacer que el proyecto en sí, crezca enormemente ya que este reconocimiento estará ligado al interior del proyecto, en el contexto de desarrollo de la comunidad; Este lineamiento fomenta el sentido de pertenencia, una forma positiva de remuneración de esfuerzo que hace a la comunidad más real y humana, estableciendo relaciones sociales que tengan un efecto de estabilidad y

permanencia en la cooperación de los miembros desarrolladores[70]. Es decir que la cooperación en un proyecto de software libre:

> *Con una estructura que permita el reconocimiento de esfuerzos se hace más personal y estable la permanencia en la comunidad.*

Si se observan los cuadros estadísticos de los casos de estudio se puede ver que no es una regla general que los miembros más antiguos sean los que más aportaciones efectúan en el proyecto de desarrollo, pero el hecho de poder reconocer a otros desarrolladores en la estructura del proyecto hace que la comunidad tenga perfiles que sirvan de ejemplificación al resto de miembros que pertenecen a esta.

4.5.4 Cambiar no Desertar

Abandonar un proyecto de software libre implica mucho más que no volver a desarrollar código fuente, la deserción en el campo de la codificación puede significar la adopción de otro rol de usuario no orientado a la producción de software pero si a otros aspectos directamente relacionados con el proyecto de software libre, examinando la situación de un desarrollador se ha dicho que llega normalmente por algún tipo de motivación personal, lo más común es que el software para el cual hace sus aportaciones de desarrollo tenga un uso con el cual el desarrollador este desenvolviéndose, entonces el conocimiento obtenido por esta persona será bastante importante en la cobertura de funciones que no necesariamente son simultaneas a la creación de código fuente y para las cuales con los sistemas de reportes automáticos actuales no es necesaria la intervención directa, tales como la prueba de funcionalidades, reporte de errores, mejoras de

[70] Karl Fogel, Producing Open Source Software, How to Run a Successful Free Software Project, Pag 87

compatibilidad entre otros. Lo que se puede concluir para el miembro que abandona la fase de desarrollo de código fuente es que:

> *Un miembro puede cooperar en variados aspectos de un proyecto de software libre, estableciendo como base de la comunidad la rotación de actividades .*

La anterior parece una regla que aplique una comunidad para mantener al desarrollador involucrado de alguna manera en el proyecto de desarrollo, aunque no es el formalismo en los proyectos de desarrollo de software libre, en algunas comunidades por algún factor desconocido los miembros pasan de una labor a otra (caso GNU/Linux) tal vez se deba a gran tamaño de algunas comunidades de desarrolladores que pueda dar pie a un cambio casi total de contexto, por ejemplo en caso de estar desarrollando código fuente el miembro puede dejar las labores asociadas y hacer parte del equipo de traducción o de pruebas de errores entre otros.

4.5.5 Seguimiento de la calidad

En una cosa de locos sacar un producto al mercado y no hacer seguimiento de a la calidad implementada, en la practica la producción de software libre implementa un control de calidad basado en reporte de errores, reporte de vulnerabilidades, publicación de parches, actualizaciones automáticas entre otros; existe un gran vacío al seguimiento de factores de calidad que puramente se evalúan en la práctica y aunque la modificación del código fuente y del programa mismo sea una ventaja del software libre, el seguimiento de los factores comunes de calidad por parte de los desarrolladores es bastante lento y se basa más en el reporte de fallas que en la misma aprobación de los usuarios creando una clara consecuencia de amplia demora en refinamiento de estos factores, el día de hoy

algunos programas invitan al usuario común a contribuir de forma anónima a un “programa de mejoramiento” (LibreOffice v4.x en el momento de instalación) no es una norma, en los casos de estudio analizados solo FreeBSD dispone de uno como tal, por lo tanto se puede afirmar que en los proyectos de software libre:

> *Es necesario disponer de procedimientos de seguimiento a la calidad de cada versión.*

De tal forma que las características mejorables de una versión de software en particular puedan ser mejoradas para poder competir con superioridad contra otros programas de software aprovechando alta frecuencia con que los proyectos de software libre sacan a la luz actualizaciones y nuevas versiones.

4.6 Consideraciones especiales de los lineamientos propuestos

Los lineamientos propuestos en este documento son derivados en la observación de los casos de estudio analizados, como se ha aclarado estos casos de estudio dependen de condiciones temporales (año, mes, día) como de la versión de desarrollo, tamaño de la comunidad y objetivos prácticos del proyecto de desarrollo, también de otras variables como calidades de miembros, ocupaciones, grado de estudios entre otras que no fueron citadas en el presente por carecer de estos datos individuales.

Para este apartado se presentan algunas valorables distinciones entre los casos de estudio que vale la pena tener en cuenta para consideraciones futuras, según

los lineamientos propuestos anteriormente se pueden tener algunas aclaraciones o acotaciones a su acción, así:

> *Proveer al nuevo miembro en un proyecto de software libre, de múltiples beneficios personales en la cooperación para el desarrollo del proyecto.*

Para el miembro que quiere iniciarse en desarrollo de programas informáticos no es del todo sencillo emprender esta actividad, garantizar que cualquier persona puede encontrar un desempeño favorable en cualquier proyecto de software libre de tal modo que este pueda repercutir en algún beneficio personal no es un camino seguro para todos los nuevos miembros, porque dependerá de sus condiciones personales y muchas veces también del grado de complejidad y curva de aprendizaje en el proyecto de desarrollo.

> *Unificar, compatibilizar o eliminar características de un proyecto de software libre en una versión acumulativa fortifica la cooperación en la comunidad.*

Las comunidades de software libre no siempre son estables en su estructura de organización interna, esto conduce de que en determinadas ocasiones las bifurcaciones de software sean positivas para una comunidad en el sentido de fortalecer diferencias que puedan representar un factor diferenciador para la escogencia del usuario, como es el caso de las primeras distribuciones basadas en BSD tales como OpenBSD, NetBSD y FreeBSD; también la continuación de una rama de desarrollo basada una versión antigua como PhpMyAdmin que fue comenzado a desarrollar basado en MySQL-Webadmin de Peter Kuppelwieser en 1998.

> *Con una estructura que permita el reconocimiento de esfuerzos se hace más personal y estable la permanencia en la comunidad.*

En algunos casos el reconocimiento puede llegar a ser excesivo y contraproducente, haciendo que la comunidad se vea identificada con uno o varios miembros en particular, que puedan afectar de forma negativa la imagen atractiva del proyecto, y consecuentemente una motivación de los miembros a cooperar con el proyecto en cuestión.

Un miembro puede cooperar en variados aspectos de un proyecto de software libre, estableciendo como base de la comunidad la rotación de actividades.

Este lineamiento pretende ser una estrategia para tratar de retener al miembro en actividades que puedan percibir para el proyecto algún tipo de colaboración que conduzca a mejorar algún aspecto del proyecto de desarrollo, sin embargo en proyectos de prueba de compatibilidades como Gentoo/KDE u otros similares no poseen una amplios campos de desenvolvimiento como posibles traducciones o diseños gráficos, para posibles colaboradores, entonces la deserción de un desarrollador y probador solo sería mejorable con otro desarrollador que pudiese desarrollar la actividad abandonada.

Es necesario disponer de procedimientos de seguimiento a la calidad de cada versión.

Tal vez el seguimiento integral de los factores de calidad de una aplicación sean demasiado engorrosos como para poder hacer uso de estos posibles datos recogidos, en caso por ejemplo de un computador con instalación de kernel GNU/Linux y en el cual se maneje el sistema sin gestor de ventanas y sin salida directa de internet ya de por si es algo engorroso poder identificar en que parte del kernel GNU/Linux se produce el error y si es un error nuevo tratar de evitar el cuelgue para identificarlo y poder reportarlo en la lista de soporte de internet.

5. Conclusiones y Discusion

5.1 Sumario hallazgos

Los principales hallazgos de esta investigación están compuestos por conclusiones deducidas de los comportamientos que han presentado los proyectos de desarrollo de software libre analizados en los casos de estudio en su historia de vida, se han tenido en cuenta la cantidad de aportaciones de los desarrolladores, los momentos de desarrollo, las bifurcaciones de un proyecto, las versiones publicadas, los tamaños de la comunidad, el objeto de funcionamiento de cada proyecto y el campo de software al cual se orienta de esta forma se han podido establecer relaciones entre los niveles de cooperación alcanzados en determinados periodos del proyecto de desarrollo.

5.1.1 Los proyectos de software libre como dilema social

Se ha enmarcado esta investigación de la cooperación en proyectos de desarrollo de software libre, por las características intrínsecas de los casos de estudio vistos, como dilemas sociales de gran escala, logrando de este modo ubicar comportamientos, procesos y tendencias reconocidas en los proyectos de desarrollo de software libre, dentro de la teoría de la cooperación en dilemas sociales de gran escala.

Así mismo tener en cuenta en las observaciones realizadas tópicos esenciales de la teoría de la cooperación ha permitido reconocer en los casos de estudio dinámicas de cooperación a nivel interno y externo de los proyectos de desarrollo de software libre que afectan directamente el comportamiento de crecimiento o disminución del recurso en los proyectos.

La realimentación de información se toma en los casos de estudio expuestos como la capacidad del desarrollador de ver reflejada en una versión de software publicada sus aportaciones durante el periodo de desarrollo, hecho que hace de una versión publicada el recurso del cual dispone una comunidad de desarrolladores y que para los casos de estudio expuestos se comprueba un comportamiento incremental por cada versión publicada, característica dependiente del volumen de aportaciones efectuado por los miembros durante los periodos relacionados a esta.

Si bien el retardo de la información hace que los desarrolladores que cooperan en un proyecto de software libre perciban sus aportaciones en el proyecto como una parte de la versión publicada, este proceso de asimilación de información no es igual para todos los miembros como incentivo motivador de para cooperar en el proyecto. De la misma forma la reciprocidad encontrada en los miembros a través del análisis de valles de aportaciones en momentos de la vida de un proyecto de desarrollo de software libre no es ni inmediatamente continua, ni proporcional a la ejecución de actividades cooperativas por parte del resto de la comunidad de desarrollo.

Finalmente fue apreciable que en proyectos de desarrollo de software libre el desarrollador de software atraviesa por situaciones problemáticas que hacen que iniciarse, pertenecer o mantenerse activo en un proyecto de desarrollo de software libre sea una decisión oscilante que se incline por el peso que agreguen variables de motivación tanto personales como grupales.

5.1.2 Los mecanismos de cooperación son mejorables

En los casos de estudio analizados se pudieron observar individualidades que crean diferencias a la hora de crear paralelos en los proyectos de desarrollo de software libre, estas particularidades de cada proyecto de desarrollo de software libre han servido para hacer la identificación de mecanismos de cooperación dentro de estos casos representativos un proceso confiable mostrando posibles inconsistencias o circunstancias en las cuales los mecanismos de cooperación deben ser considerados de un modo especial llegando a la consideración de especificaciones propias para el caso de estudio en particular.

Los eventos de cooperación en la producción de software libre expuestos en este estudio hacen de los lineamientos precisas estrategias para promover la efectividad de los mecanismos de cooperación en los proyectos de producción de software libre, sustentando esto último en la historia de vida de los casos de estudio analizados en esta investigación que prueba la eficacia de resultados basados en comportamientos como la confianza, la reciprocidad, cooperar como norma, y la provisión de incentivos.

Teniendo en cuenta que los eventos presentados en los casos de estudio naturalmente no son situaciones idénticas, aun para los mismos proyectos de desarrollo si los situamos en otro periodo de tiempo diferentes u otra versión de software en desarrollo, es asumible que son situaciones que las comunidades de desarrollo de software libre experimentan en determinados momentos de la vida del proyecto, basando este hecho en las condiciones de escogencia que se efectuó para los casos de estudio analizados; suficientemente representativos para el mundo del software libre.

5.1.3 Expectativa de los lineamientos

Cada lineamiento propuesto pretende ser una herramienta estratégica en aspectos vitales en la vida de los proyectos de desarrollo de software libre, a continuación se citaran los lineamientos y la derivación de resultados que producirán para un proyecto de desarrollo de software libre:

1. *Proveer al nuevo miembro en un proyecto de software libre, de múltiples beneficios personales en la cooperación para el desarrollo del proyecto*

Este lineamiento está enfocado en atacar el arribo de nuevos miembros a un proyecto de software libre, porque estos son un recurso vital de los proyecto de desarrollo; en los históricos de membresías de proyectos de desarrollo de software libre la deserción es una situación que es mitigada de forma directa cuando el miembro desarrollador coopera activamente con el proyecto de desarrollo, esta estrategia es una forma indirecta de mitigar el daño que ejerce la deserción sobre los proyectos de software libre.

2. *Unificar, compatibilizar o eliminar características de un proyecto de software libre en una versión acumulativa fortifica la cooperación en la comunidad.*

La necesidad de estar a la vanguardia no siempre una ventaja que se pueda alcanzar con liberar una versión particular de software con nuevas características, con amplias pruebas de funcionamiento o diferentes compatibilidades a la versión anterior; por otra parte en el ámbito del software libre aferrarse enfermizamente a una filosofía en particular de funcionamiento del software hace que las comunidades tengan que redoblar esfuerzos para mantener el proyecto de desarrollo activo y produzca en varios periodos de tiempo proyectos de desarrollo separados que mucho después tienen en común los aspectos que en algún

momento fueron el fruto de su separación, caso de las distribuciones BSD donde al día de hoy todas tienen características de usabilidad, seguridad y compatibilidad a un nivel mucho más compartido y no como cuando se produjeron sus correspondientes bifurcaciones[71].

Este lineamiento cubre dos situaciones básicas en los proyectos de software libre, la primera situación es la posibilidad de implementar nuevas características sin la necesidad de exponer al proyecto en su totalidad en un cambio que resulte en un error radical que tenga que ser enmendado por una siguiente versión completa, asegurando la confianza de la comunidad de desarrollo en el proyecto.

La segunda situación que cubre es unificar a la comunidad por medio de la decisión conjunta de las mejores características que deberían adoptarse y para las que ya los desarrolladores cooperaron en implementaciones y pruebas, haciendo una versión integral que asocie lo mejor de varias ramas de desarrollo al interior del proyecto se asocian también los desarrolladores asociados a cada rama de desarrollo, como también se puede observar en periodos de desarrollo del Kernel GNU/Linux, cercanamente en el paso de las versiones 2.x a la 3.x.

3. *Con una estructura que permita el reconocimiento de esfuerzos se hace más personal y estable la permanencia en la comunidad.*

Este lineamiento está orientado a reducir la virtualización que el miembro desarrollador tiene de la comunidad de desarrollo, mejorando la retroalimentación que pueda obtener de la comunidad, también presentando el reconocimiento público como un importante incentivo que apoye la permanencia para el desarrollador y un motivante explicito para el nuevo miembro. Aunque los sistemas de control de versiones disponen de un histórico de miembros activos porque son

[71] Ver Peter N.M. Hansteen 2011 Ed 2, The book of Pf, Pag 3

los que efectúan las aportaciones en las ramas de desarrollo, pobremente el volumen de aportaciones es reconocido como un factor de importancia personal en la participación del proyecto de desarrollo, cuestión que hace de la estructura misma de los proyectos de desarrollo de software libre una filosofía con menos posibilidades de incentivos motivacionales para un desarrollador que es una persona con realidad propia y motivaciones totalmente humanas; Entonces en el proceso de realimentación y de confianza se hace de la comunidad de desarrollo un espacio diferente para el logro de causas personales.

Tener este lineamiento como premisa en un proyecto de desarrollo de software libre, asegura al nuevo miembro un trato que lo haga sentir necesario e importante en las labores que desarrolle del proyecto. No hacerlo conlleva a que el nuevo miembro no vea remuneradas sus aportaciones en reconocimiento, lo que a la larga produce un efecto de desconfianza y decisión de no cooperación.

4. *Un miembro puede cooperar en variados aspectos de un proyecto de software libre, estableciendo como base de la comunidad la rotación de actividades.*

Cuando los miembros que cooperan en un proyecto de desarrollo de software libre ven vencidos los incentivos o motivaciones por las cuales llegaron en primera medida a formar parte activa en un proyecto de desarrollo, esta coyuntura no debe ser enfocada por el proyecto de desarrollo como el fin definitivo de la participación del miembro dentro del proyecto, sino como la oportunidad para que el miembro sea incluido en diferentes actividades y sea motivado hacia el alcance de nuevas metas e intereses personales a través de la cooperación activa dentro del proyecto de desarrollo. Cuando la comunicación activa mantiene al miembro en contacto este se vuelve un recurso para guiar a los miembros nuevos en actividades antiguamente desarrolladas por el mismo y crear en el miembro un sentido integral del desarrollo en las comunidades de software libre.

No todos los miembros tienen una misma forma de reaccionar ante un evento de disminución de confianza o de bajo incentivo en la labor que realiza, por eso este lineamiento no es una estrategia de inmunidad ante la deserción que se produce en los proyectos de software libre, sino más bien la proposición de adopción para la estructura de funcionamiento interno de los proyectos de software libre de una característica importante como lo es la rotación en labores que son monótonas y especializadas, cuestión que produce en los miembros desarrolladores un significativo desgaste de atención, motivaciones y expectativas, frente a las retribuciones alcanzadas. En otras palabras los miembros que no ven alguna clase de incentivo que los motive en mantenerse activos en el proyecto, deben de tener la facilidad de cooperar en otros campos de desarrollo del proyecto implementado esto como parte del sistema de funcionamiento interno del proyecto.

5. *Es necesario disponer de procedimientos de seguimiento a la calidad de cada versión.*

La retroalimentación que ofrece el ciclo de desarrollo del software libre parece suficiente para lograr mejorar en las versiones de software desarrolladas y evolucionar hacia nuevas características que el mercado va dictando como necesidades en los diferentes tipos de software existentes. El enfoque primario de este lineamiento es disminuir el tiempo de retroalimentación; siendo que los proyectos de software libre tienen un corto tiempo de publicación de parches para asuntos tales como seguridad, el tener implementado en la estructura misma de desarrollo de un proyecto de software libre procedimientos de medición de otros factores de calidad de software produce en el producto terminado una rápida forma de producir reportes de aceptación, desempeño y compatibilidad, cuestiones que hacen de cualquier software un producto robusto y de competitividad frente al software privado.

Para los miembros desarrolladores resulta en un ahorro de esfuerzo evitando la duplicidad en implementar soluciones similares, evitando desperdiciar esfuerzo que puede ser utilizado en asuntos igualmente importantes para el desempeño del software pero menos evidentes a simple vista por la comunidad de desarrollo.

Para los usuarios finales del software el uso de procedimientos de reporte de aceptación, fallas, mejoras y/o usos comunes contribuye a la calidad del proyecto de software libre porque empuja a la comunidad en su totalidad a buscar suplir nuevas necesidades y no redundar en las antiguas, eliminando al cliente final el problema de estar conforme y limitado con el uso que le permite el software, haciendo del usuario final un actor con mejor capacidad de cooperación en el proyecto.

5.2 Discusión de los hallazgos

Los hallazgos expuestos en este capítulo han sido el camino por medio del cual se relacionan a través del estudio de la cooperación en los casos de estudio del software libre procesos y circunstancias que son críticas en los proyectos de desarrollo de software libre. Aunque los casos de estudio que se han analizado en esta investigación son representativos en el mundo del software libre, teniendo en cuenta las particularidades que posee cada uno de estos por pertenecer a un tipo característico de mercado o de objeto de funcionamiento, las dinámicas de cooperación en el proceso interno de producción y también las externas presentan amplias similitudes como:

- Altos números de miembros desarrolladores.
- Retroalimentación retardada.
- Necesidad de confianza.
- Cooperación como requisito en el aprendizaje social.
- Baja cohesión de los miembros desarrolladores.
- Anonimato de los miembros desarrolladores.

Las características citadas hacen de los procesos de cooperación en el software libre un objeto de estudio apropiado para la teoría de la cooperación en los cuales se puede hacer inferencias que permitan recrear a través del conjunto de lineamientos propuestos situaciones positivas en los proyectos de desarrollo del software libre. En el siguiente cuadro se exponen los lineamientos propuestos y su aplicación para un resultado satisfactorio.

Lineamiento	**Aplicación**
Proveer al nuevo miembro en un proyecto de software libre, de múltiples beneficios personales en la cooperación para el desarrollo del proyecto	Los proyectos de desarrollo de software libre deben de mostrar beneficios alcanzables como resultados de cooperar en desarrollo de software libre. No solamente como la posibilidad de mejorar conocimientos o de contribuir de con espíritu altruista.
Unificar, compatibilizar o eliminar características de un proyecto de software libre en una versión acumulativa fortifica la cooperación en la comunidad.	Implementar la integración de diferentes ramas un proyecto de desarrollo de software libre es imprescindible para lograr que la vida del proyecto se mantenga en crecimiento estable y no se vea fragmentada inmediatamente a la ausencia de grandes cambios de funcionamiento del software o estructurales de la comunidad.
Con una estructura que permita el reconocimiento de esfuerzos se hace más personal y estable la permanencia en la comunidad.	Poder humanizar más el papel del desarrollador de software libre dentro del proyecto de software retribuye al proyecto con personas que mantengan un alto grado de compromiso que son regularmente los actores de producción que mantienen por largos periodos de tiempo activos dentro del proyecto de desarrollo de software libre.

	Algunos proyectos como joomla CMS hace de los grupos de desarrollo de actividades un subconjunto representativo con datos personales como fotos, nombres e historial de actividades de la comunidad de desarrollo como foros, conferencias asambleas entre otros (https://www.facebook.com/joomla?fref=ts).
Un miembro puede cooperar en variados aspectos de un proyecto de software libre, estableciendo como base de la comunidad la rotación de actividades.	Autores de la teoría de la administración como Taylor proponen una medición de tiempos en el desarrollo de procesos industriales, se hace necesaria esta propuesta de lineamiento como respuesta a la fatiga generada a largo plazo por la repetición frecuente de labores para las cuales el miembro desarrollador se encuentra o alcanza un grado de especialización; también como estrategia a la perdida de las motivaciones iniciales, siendo esta propuesta de lineamiento, el ofrecimiento para el miembro desarrollador de nuevas motivaciones e incentivos.
Es necesario disponer de procedimientos de seguimiento a la calidad de cada versión.	En la actualidad las aplicaciones de software libre, disponen de procesos de medición de factores de calidad no más detallados o completos que los que ofrecen las aplicaciones privadas, situación que es silenciada por la alta regularidad de parches y otros mejoras liberadas de forma automática o en foros de soporte. Establecer funcionalidades de reporte automático y/o manual que a corto plazo cuantifiquen los factores de calidad hacen que el producto de software realizado encuentre alto refinamiento y robustez.

Es de tener en cuenta que los lineamientos propuestos han sido formulados en condiciones practicas generadas por la gran aceptación de los proyectos de desarrollo de software libre vistos en los casos de estudio y la larga vida de desarrollo que estos han tenido, lo que establece unos condicionamientos bajo los cuales la aplicación de esta propuesta de lineamientos para la cooperación en los proyectos de desarrollo de software libre debe ser revisada y adaptada a cambios en el conjunto de variables.

6. Recomendaciones y trabajo futuro

Este trabajo de análisis de los casos de estudio de representativos en el mundo del software libre busco hacer una identificación de la teoría de la cooperación en los dilemas sociales; tomando algunas dinámicas de producción de software libre como prueba del uso de algunos mecanismos de cooperación en los casos representativos estudiados, de otra forma este ejercicio realizado es una etapa de exploración para la descripción de dinámicas de cooperación en los proyectos de desarrollo de software libre. Para la recopilación de material significativo de esta investigación se hizo uso de material bibliográfico, la observación de historiales estadísticos de producción, historial de versiones y características particulares de los casos de estudio.

Se produjo a través de los mecanismos de la teoría de la cooperación un marco teórico para poder determinar la relevancia cualitativa de ciertos aspectos y circunstancias en el dilema social de la cooperación en los proyectos de desarrollo de software libre; como proposición para el mejoramiento de estos mecanismos de cooperación se produjo una propuesta de lineamientos para ser implementados en proyectos de desarrollo de software libre, esta propuesta a su vez, puede ser mejorada definiendo variables de forma cuantitativa que puedan dar fe por medio de razones matemáticas de comportamientos consecuentes con cambios en parámetros, esto se puede lograr a través de recolección de datos específicos de los miembros desarrolladores, siendo la interpolación matemática un procedimiento que deba producir comportamientos similares a los encontrados en la realidad; lo anterior supone riesgos y dificultades para el investigador, ya que por la misma naturaleza de las comunidades de software libre los individuos se encuentran dispersos en varios países, hablan varios idiomas entre otras

dificultades que pueden en el momento practico de tomar muestras presentar una baja confiabilidad.

Después de encontrar inconvenientes para la obtención de datos confiables que fueran secundarios a los movimientos de aportaciones dentro de los casos de estudio analizados, que pudiesen prestar más confiabilidad practica a esta investigación es aconsejable que en próximos estudios de cooperación en el software libre se propongan casos de estudio igualmente representativos que puedan ser enmarcados como dilemas sociales de pequeña escala por que permitirían hacer una recolección de datos más exitosa y confiable que pueda ser usada como punto de partida al estudio de dilemas sociales de gran escala en el software libre. Aunque esto supondría la dificultad principal de obtener datos de comportamiento demasiado limitados en el tiempo, es decir los periodos de tiempo son más largos en proyectos de pocos integrantes hecho que extiende la recolección de datos confiables demasiado, o los acota a un plazo corto de tiempo.

Existen herramientas de recolección de datos que harían más profundo el análisis de la cooperación en el software libre como las entrevistas y los cuestionarios, sin embargo por la dificultad de la naturaleza de los proyectos de desarrollo de software libre para permitir estas actividades como proceso confiable en un corto periodo de tiempo. Es posible recomendar este análisis para cuantificar la retroalimentación presentada entre las versiones publicadas para el usuario final y el proyecto de desarrollo, esto es en parte alcanzable mediante una pequeña aplicación que efectué mediciones de factores de calidad en cada programa usado como numero de errores (cuelgues), preferencia frente a otros sistemas, desempeño de funcionamiento, encuesta de satisfacción entre otros.

Siendo el miembro desarrollador de software libre el actor principal en el dilema social de desarrollo de software libre es necesaria la búsqueda de métodos prácticos que puedan dar cuenta cuantitativa de factores personales que incidan

en la decisión de ingresar y/o permanecer en una comunidad de desarrollo de software libre, para hacer de argumentos, modelos y simulaciones productos más eficaces en la correlación de la teoría de la cooperación en los proyectos de desarrollo de software libre.

En la búsqueda de reducir la comprensión de las dinámicas complejas de cooperación en los proyectos de desarrollo de software libre es deseable dirigir próximas investigaciones hacia un conjunto de proyectos de desarrollo de software libre que pertenezcan a una gama de mercado en especial, o que posean objetivos de producción analógicos para en este contexto poder producir observaciones que conduzcan a deducciones especificas del genero estudiado, cuestión que puede sacar a la luz de forma cuantitativa diferencias en el software libre que a la larga pueden producir situaciones positivas en un proyecto y negativas en otros.

Con base a este estudio y a las propuestas para investigaciones futuras es provechoso proponer el modelamiento matemático de comportamientos observados en los proyectos de desarrollo de software libre con amplios tiempos de vida y de esta forma poder inferir comportamientos a largo plazo en el desarrollo de proyectos de software libre.

BIBLIOGRAFÍA

Andrick Jorge 2010, Constructo para la evaluación de la cooperación en dilemas sociales de gran escala.

Andrick Jorge 2012 , Evaluación de la Cooperación en Dilemas Sociales de Gran Escala.

Cangiano Antonio 2009, Ruby on Rails for Microsoft Developers.

Delisle Marc, Mastering phpMyAdmin 3.4 for Effective MySQL Management.

Fogel Karl, Producing Open Source Software, How to Run a Successful Free Software Project.

González B Jesús M 2000, Software libre 2ª Ed. Febrero 2008.

Hansteen Peter N.M. 2011 Ed 2, The book of Pf,.

Hofstede Geert., Software libre, Cultures and organizations, Software of the mind.

Kollock 1998, SOCIAL DILEMMAS: The Anatomy of Cooperation.

Love Robert 2010, Linux Kernel Development.

Marriott Jennifer 2010, Waring Elin, The Official Joomla! Book

Matellán Olivera Vicente Universidad Rey Juan Carlos , Compilación de ensayos sobre software libre.

Moineau Laurent y Papathéodorou Aris, Cooperación y producción inmaterial en el software libre. Enero 2000.

Ostrom Elinor, 1998, EL GOBIERNO DE LOS BIENES COMUNES Ed. en español.

Pressman Roger S. 5a Ed, Ingeniería del Software un Enfoque Práctico.

Roberto Hernández Sampieri, Carlos Fernández Collado, Pilar Baptista Lucio, Metodología de la Investigación, 4ta Edición, México

Sádaba Igor, Dominio Abierto, Conocimiento libre y cooperación.

Stallman M. Richard, Por qué el software no debe tener propietarios.

Stallman M. Richard, Software Libre - Sociedad Libre, Por qué el software no debe tener propietarios.

Stallman M. Richard, Software libre para una sociedad libre.

Vermeir Dirk 2005, How Open is the Future?.

LISTADO DE ILUSTRACIONES

LISTADO DE TABLAS

ANEXOS

ANEXO A-1

Estadísticas relacionadas a la operación adicionar en el caso de estudio GNU/Linux

ANEXO A-2

Estadísticas relacionadas a la operación commit en el caso de estudio GNU/Linux

ANEXO B-1

Estadísticas relacionadas a la operación adicionar en el caso de estudio FreeBSD

ANEXO B-2

Estadísticas relacionadas a la operación commit en el caso de estudio FreeBSD

ANEXO-C1

Estadísticas relacionadas a la operación adicionar en el caso de estudio Gentoo/KDE

ANEXO C-2

Estadísticas relacionadas a la operación commit en el caso de estudio Gentoo/KDE

ANEXO D-1

Estadísticas relacionadas a la operación adicionar en el caso de estudio Joomla-CMS

ANEXO D-2

Estadísticas relacionadas a la operación commit en el caso de estudio Joomla-CMS

ANEXO E-1

Estadísticas relacionadas a la operación adicionar en el caso de estudio Joomla CMS

ANEXO E-2

Estadísticas relacionadas a la operación commit en el caso de estudio PhpMyAdmin

ANEXO F-1

Estadísticas relacionadas a la operación adicionar en el caso de estudio Ruby on Rails

ANEXO F-2

Estadísticas relacionadas a la operación commit en el caso de estudio Ruby on Rails

ANEXO G

Histórico de versiones de los casos de estudio