Curso E-learning para el mejoramiento de las competencias de cooperación en el desarrollo de proyectos de software libre orientado a los lineamientos metodológicos de la programación extrema

E-learning Courses to Improve the Skills of Cooperation in the Development of Free Software Projects Oriented by Methodological Guidelines of the Extreme Programming

Recibido: 9 de mayo de 2013
Aprobado: 14 de mayo de 2013

Byron Cuesta Quintero* y Jorge Andrick Parra**

Resumen

Este artículo surge a partir de la interpretación de la realidad de las prácticas de cooperación en el desarrollo de software libre, en un contexto donde se identificaron desde la racionalidad individual de los desarrolladores, los factores que pueden mejorar en forma colectiva las prácticas de cooperación. Se realizó una revisión de la literatura

* M.Sc. Software Libre, Universidad Autónoma de Bucaramanga. Especialización en Informática Educativa, Universidad Francisco de Paula Santander. Ingeniero de Sistemas. Administrador de Sistemas de Información y docente, Universidad Francisco de Paula Santander. E-mail: byroncuesta@ufpso.edu.co

** Ph. D. Ingeniería de Sistemas, Universidad Nacional de Colombia, sede Medellín. M. Sc. Informática, Universidad Industrial de Santander (UIS). Ingeniero de Sistemas. Profesor Titular, programa de Ingeniería de Sistemas, Investigador Asociado, Grupo de Investigación en Pensamiento Sistémico, Universidad Autónoma de Bucaramanga Colombia, Presidente de Comunidad Colombiana de Dinámica de Sistemas. E-mail: japarra@unab.edu.co

sobre las teorías de la cooperación en general, la cooperación en el desarrollo de software libre y la programación extrema. Los resultados obtenidos muestran un inventario de competencias de fácil aplicación en el desarrollo de software libre, que se encuentran plasmados en un curso E-learning y sugieren un sistema de competencias para la cooperación.

Palabras clave

Software libre, cooperación, programación extrema, prácticas de desarrollo de software.

Abstract

This article arises from the interpretation of the actual practices of cooperation in the development of free software in a context where individual rationality identified from developers, the factors that can collectively enhance cooperation practices. A review of the literature on theories of general cooperation, cooperation in the development of free software and extreme programming. The results show an inventory of skills easily applied to the development of free software that are embodied in a course e-learning and suggest a system for cooperation skills.

Keywords

Free software, cooperation, extreme programming, software development practices.

I. Introducción

El proceso de desarrollo de software libre, se materializa gracias a su modelo cooperativo de producción en red [1]. En este contexto y ante el impacto del software libre en el mundo moderno, se examinaron las prácticas de cooperación de los desarrolladores, de manera que se pudieran reconocer las reglas propias de este modelo de contribución en red a partir del bien colectivo, fundado en la confianza y la reciprocidad. Los desarrolladores de software libre, se encuentran organizados en comunidades virtuales que comparten las mismas costumbres, creencias y aficiones y que repercuten en la contribución y mantenimiento de un bien común: el software libre.

Los proyectos de software libre, requieren de prácticas que contribuyan en la construcción de sólidas y efectivas acciones de cooperación. Para lograr este objetivo, la investigación se apoyó en las siguientes temáticas: Cooperación en el desarrollo de proyectos de software libre, Teoría de la cooperación en general y la programación extrema.

El resultado de la investigación, permitió identificar un sistema de competencias para la cooperación que dio paso al diseño del curso E-learning y a su implementación, el cual permite adoptar competencias para mejorar el trabajo cooperativo de los desarrolladores de software libre.

II. Antecedentes

2.1 Modelo de desarrollo del software libre

Los desarrolladores de software libre encuentran motivación en aprender y desarrollar nuevas habilidades de programación [2], permitiendo que el trabajo en colaboración desencadene en la necesidad de aportar a la comunidad y obtener reputación por ello [3,4]. La reputación es el engranaje del sistema que va de lo individual a lo compartido, funciona de una manera dual y motiva a los programadores a contribuir y si tienen relevancia en el proyecto se apropian de nuevas responsabilidades, definiendo patrones de colaboración e interacción [5]. Esto se puede corroborar en el resultado de la encuesta expuesta en-flossproject.org, que muestra el reporte final de la encuesta a desarrolladores y que se basa en dos preguntas: los motivos para unirse y por qué permanecer en una comunidad de software libre [6].

En «La catedral y el bazar» [7], se describen algunas características del modelo de desarrollo de software libre, haciendo especial énfasis en lo que diferencia estos modelos de los desarrollos propietario. También se expresa que el núcleo Linux, se desarrolló siguiendo un esquema similar al de un bazar oriental. En un bazar no existe una máxima autoridad que controle los procesos que se están desarrollando, ni que planifique estrictamente lo que ha de suceder. Por otro lado, los roles de los participantes pueden cambiar de manera continua y sin indicación externa y muestra que el éxito de Linux dentro del mundo del software libre es una sucesión de buenas maneras para aprovechar al máximo las posibilidades que ofrece la disponibilidad de código fuente y la interactividad mediante el uso de sistemas y herramientas telemáticas [8].

Uno de los principios básicos del modelo de desarrollo bazar, es considerar a los usuarios como desarrolladores [9]. Un proyecto de software libre suele surgir a raíz de una acción puramente personal. El desarrollador una vez que haya conseguido tener algo usable, con algo de funcionalidad, sencillo y a ser posible bien diseñado o escrito, lo mejor que puede hacer es compartir esa solución con la comunidad del software libre [10]. Este modelo de desarrollo basado en una cooperación en red, utiliza lo que se denomina la publicación temprana y que permite llamar la atención de otras personas, generalmente desarrolladores que tengan el mismo problema y que puedan estar interesados en la solución y además se consigue otra ventaja, como el hecho de que al integrar frecuentemente no haga falta una última fase de integración de los módulos que componen el programa [11].

El software libre es un fenómeno que es posible debido a la colaboración de comunidades distribuidas y que, por tanto, necesitan herramientas que hagan efectiva esa colaboración. Las listas de correo juegan un papel crucial en el software libre, llegando en muchos casos a ser el único método de contribución. Hoy día, con la popularidad de la web, se utilizan los foros web o *weblogs*, donde se anuncia nuevo software libre o se discuten noticias relacionadas, que normalmente están integrados en sitios de colaboración con diversas herramientas. Asimismo, se ha hecho popular un mecanismo de colaboración basado en wikis, sobre todo cuando se trata de elaborar un documento conjunto [12].

2.2 Los dilemas sociales y el ambiente de la cooperación

El término «dilemas sociales», se refiere a las situaciones en que los humanos individualmente toman decisiones independientes en una situación interdependiente [13]. En los dilemas sociales, si todos contribuyen, todos reciben un beneficio neto positivo, pero a su vez todos se enfrentan a tentaciones que los pueden hacer cambiar tomando decisiones aisladas y en donde, nadie se encuentra motivado a cambiar su elección, dadas las elecciones de los demás participantes, así exista un resultado que da una mayor ventaja a todos [14,15].

Cuando se involucran múltiples actores, se distinguen dos grandes tipos de dilemas: el de los Bienes Públicos, que se refiere a la existencia de la tentación de disfrutar de lo bueno sin contribuir a su creación, es decir, obtener el mejor resultado posible sin cooperar y el de los Bienes Comunes, en el cual el individuo se siente tentado con un beneficio inmediato que produce un costo compartido por todos. Si todos sucumben a la tentación, el resultado es un desastre colectivo [16, 17].

Los seres humanos pueden ser egoístas en sus beneficios inmediatos y no siempre son altruistas para el beneficio en grupo. De esta manera, cuando el contexto lo hace posible, las personas se ganan la confianza de otros individuos y a partir de ahí se pueden resolver dilemas sociales y mantener la cooperación dentro de los grupos, pero también los cambios en ese contexto pueden destruir la cooperación que ha surgido. La cooperación en un grupo, depende fundamentalmente de normas de confianza y reciprocidad. Su interacción con las características institucionales y de contexto, desempeñan un papel clave en la creación de seguridad en cuanto a las actuaciones de los integrantes de un grupo [18]. En la teoría de los juegos se utiliza la estrategia *tit-for-tat*, para resolver de manera óptima el dilema del prisionero y evidenciar que cuando muchos individuos utilizan reciprocidad, hay un incentivo para adquirir una reputación basada en la confianza [19].

Son varios los elementos que contribuyen en el establecimiento de un ambiente cooperativo entre diferentes colectividades: la comunicación cara a cara, el intercambio de compromiso mutuo, el incremento de confianza, la creación y el refuerzo de las normas y el desarrollo de una identidad de grupo, son elementos que hacen que la comunicación sea eficaz pero no una condición suficiente para asegurar una acción colectiva exitosa bajo todas las condiciones [20, 21]. La mayor razón por la cual los individuos no cooperan en los dilemas sociales son las situaciones en que ellos no quieren ser absorbidos por las contribuciones cuando otros no contribuyen (*free-riders*) [22, 23, 24].

Ostrom [25, 26] explica la existencia de variables estructurales que al ser cambiadas pueden conducir a una cascada de modificaciones sobre el nivel de cooperación de los participantes. La figura 1, muestra las variables estructurales del mecanismo de cooperación y se puede interpretar como alcanzar niveles de cooperación altos, a partir de relacionar e incluir las expectativas que los individuos tienen sobre el comportamiento de los demás (confianza), las normas que los individuos aprenden de la socialización y las experiencias de la vida (reciprocidad) y cómo los individuos crean identidades que proyectan sus intenciones y normas (reputación).

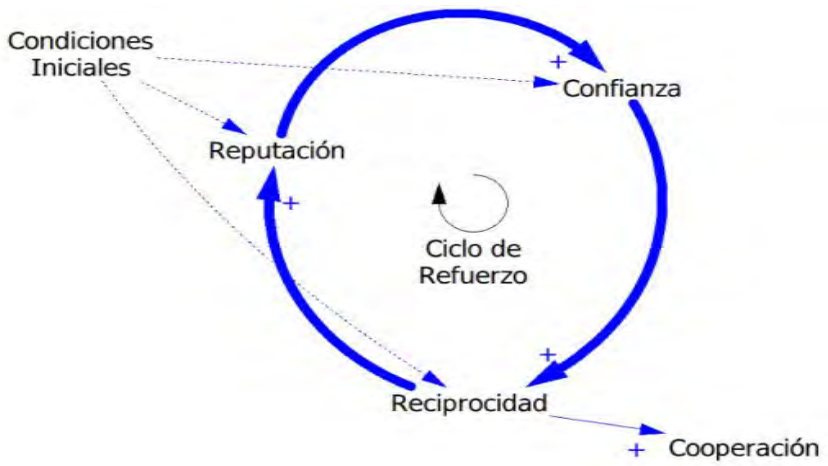

Fig.1. Versión dinámica del mecanismo de cooperación basado en la confianza. Fuente:(Parra, 2010)

2.3 Cooperación en el desarrollo de software libre

El software libre se crea a partir de un grupo de voluntarios organizados y cooperando en la construcción de software que los beneficie a todos.

S. Koch & Schneider [27, 28, 29], presentaron un estudio utilizando los datos relacionados con las aportaciones de los participantes que se encuentran en el repositorio CVS del proyecto GNOME y analizaron las asociaciones entre los programadores y los archivos, observando un bajo nivel de cooperación sobre los archivos básicos. Para los archivos más grandes, más programadores activos hacen un mayor porcentaje de su trabajo total y suponen que, dentro de este círculo reducido de personas la cooperación es posible.

Otro caso de estudio es el del servidor Web Apache [30, 31]. La mayoría de sus módulos fueron escritos por un grupo central correspondiente a quince desarrolladores que contribuyeron y compartieron el código, con el fin de evitar hacer cambios conflictivos en el mismo. De esta manera en lugar de que un solo individuo escribiera todo el código para un módulo dado, el grupo central tuvo un nivel suficiente de confianza mutua para realizar contribuciones en varios módulos de código según fuera necesario.

M. Elliott & W. Scacchi [32, 33], presentaron una investigación sobre el proyecto GNU Enterprise (GNUe) en donde muestran que la comunicación mediada por ordenador con elementos como: Internet Relay Chat (IRC), el sistema de control de versiones (CVS) y las listas de correo, facilitan el trabajo en equipo y la cooperación para construir comunidad. Así mismo muestran que la construcción de una comunidad en línea mediante una cultura organizacional sugiere principios de diseño, ya que los participantes no tienen la oportunidad de convivir cara a cara en su vida laboral cotidiana y muestran que los conflictos se resuelven simultáneamente a través de herramientas telemáticas.

2.4 Programación extrema

La Programación Extrema (XP) es una metodología ágil centrada en potenciar las relaciones interpersonales y en la que cada colaborador del proyecto es un miembro del equipo

que se encarga de todos los aspectos del desarrollo y el cliente es un integrante más que trabaja con ellos a diario. XP se basa en cuatro valores: simplicidad para las soluciones implementadas, la comunicación fluida entre todos los participantes, la retroalimentación continua entre el cliente y el equipo de desarrollo y coraje para enfrentar los cambios [34, 35].

La figura 2 muestra las cuatro fases de la Programación Extrema sobre las que se va iterando hasta que el proyecto finaliza [36, 37].

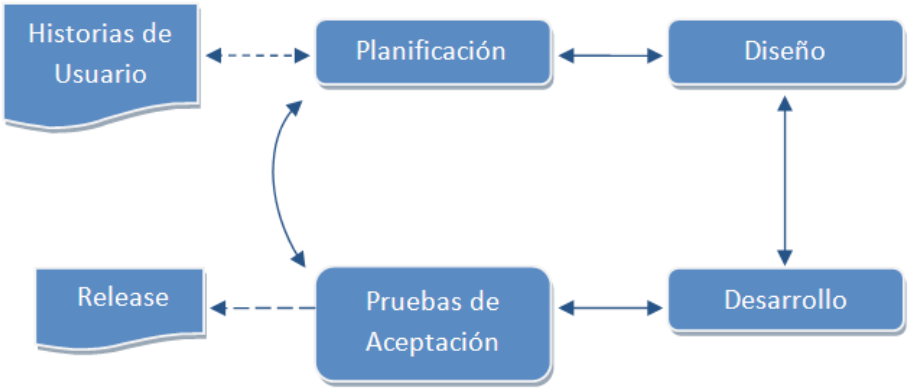

Fig. 2. Fases de desarrollo XP. Fuente: (Aycart Pérez, D, et ál., 2007).

En la fase de Planificación [38], se redactan las historias de los usuarios que tienen el mismo propósito que los casos de uso, pero no son lo mismo. Las escriben los propios clientes, tal y como ven ellos las necesidades del sistema siendo descripciones cortas y escritas en el lenguaje del usuario. También conducirán al proceso de creación de los test de aceptación y uno o más de uno de estos test se utilizarán para verificar que las historias de usuario han sido implementadas correctamente.

Se crea un plan de entregas a partir de las historias de usuario de manera que se trace plan de entregas en función del tiempo de desarrollo ideal y el grado de importancia para el cliente. Las iteraciones individuales son planificadas en detalle justo antes de que comience cada iteración [39].

Cada iteración corresponde a un periodo de tiempo de desarrollo del proyecto de entre una y tres semanas. Al principio de cada iteración se debería convocar una reunión para trazar el plan de iteración correspondiente. Está prohibido intentar adelantarse e implementar cualquier cosa que no esté planeada para la iteración en curso. Habrá suficiente tiempo para añadir la funcionalidad extra cuando sea realmente importante según el plan de entregas.

Si solo una persona del equipo es capaz de trabajar en un área concreta, existirá un riesgo enorme si esa persona abandona por cualquier circunstancia [40]. De esta forma, las rotaciones permitirán que todo el mundo conozca cómo funciona el sistema en general y ayudarán a realizar un reparto más equitativo del trabajo. Además el hecho de que se asignen por parejas permitirá entrenar a un nuevo miembro, dividiendo el grupo original en dos.

La comunicación entre las diferentes partes que interviene en un proyecto resulta fundamental para sacar a la luz los problemas, las soluciones y centrar el objetivo del equipo. La recomendación es que las reuniones sean diarias, de poca duración y a ser posible delante de la pantalla del ordenador.

En la fase de Diseño [41], se trata siempre de realizar las cosas de la manera más sencilla posible. Se elige una metáfora para el sistema, que es una historia acerca de cómo el sistema funciona. De esta manera es fácil mantener la coherencia de nombres de todo aquello que se va a implementar.

Para el diseño del sistema, se llevan a cabo reuniones entre todos los desarrolladores implicados, donde se toman las decisiones mediante el uso de tarjetas CRC (class, responsabilities y collaboration). Cada tarjeta representa un objeto del sistema, en la que su nombre aparece escrito en la parte superior, sus responsabilidades en la parte izquierda y los objetos con los que colabora en la parte derecha.

El uso de tarjetas CRC ayuda a que todos participen y aporten sus ideas en el diseño moviendo las tarjetas encima de la mesa según este va progresando.

Fase de Desarrollo [41], el cliente está siempre disponible en todas las fases que se realizan en un proyecto, preferiblemente cara a cara, en persona, sin intermediarios. Durante la reunión del plan de entregas, el usuario propondrá qué historia de usuario se incluye en cada plan. También se negociarán los plazos de entrega y debe colaborar en la realización de los test. Estos test comprobarán que el sistema está listo para pasar a la fase de producción. El usuario comprobará los resultados obtenidos y tomará decisiones en cuanto a la utilización o no del sistema realizado.

El sistema se pone por primera vez en producción en unos pocos meses, antes de estar completamente terminado. Las sucesivas versiones serán más frecuentes (entre un día y un mes). Algo importante es centrarse en la tarea que se ha fijado para hoy.

A medida que se vaya avanzando en iteraciones en el proyecto, es posible modificar o ampliar partes de código ya escritas anteriormente. Así se hace el esfuerzo de refactorizar los módulos existentes, dejándolos iguales de simples pero con la nueva funcionalidad añadida. Será siempre mucho más fácil de probar, de explicar y de comprender para el resto del equipo [42].

Debido a la movilidad de los desarrolladores durante el proyecto y como el código es común a todos los programadores, es importante que se definan elementos propios de los lenguajes de programación como la indentación, la sintaxis y nombres de variables entre otros, de manera que todo el mundo se sienta cómodo con el código escrito por cualquier otro miembro del equipo.

Todo el código es desarrollado por dos personas que trabajarán de forma conjunta en un ordenador. Mientras uno de ellos se encarga de pensar la táctica con la que se va a abordar el problema, el otro se encargará de pensar las estrategias que permiten llevar dichas tácticas a su máximo exponente. Ambos roles son intercambiables. Sólo una pareja se encargará de integrar el código para evitar problemas debidos a la integración de los módulos que se han desarrollado y no han sido testeados todavía.

Los programadores deberán actualizar sus módulos con las versiones más recientes del trabajo realizado tan pronto como les sea posible. De esta manera, todo el mundo trabajará siempre con la última versión. Esta integración se llevará a cabo cuando el éxito en las pruebas para su test correspondiente sea del 100%, o cuando se trate de una parte que constituye un todo funcional, en cuanto esté acabada.

Trabajar horas extras absorbe el espíritu y la motivación del equipo, por eso lo ideal es trabajar 40 horas semanales y es muy importante definir claramente el plan de entregas. También es una mala idea incorporar nueva gente al proyecto, una vez que este ya ha comenzado.

En la fase de Pruebas [43, 44], todo el código debe ir acompañado de su unidad de pruebas. Descubrir todos los errores que pueden aparecer lleva tiempo y más si se deja la depuración total para el final.

Una prueba de aceptación es como una caja negra. Cada una de ellas representa una salida esperada del sistema. Es responsabilidad del cliente verificar la corrección de las pruebas de aceptación y tomar decisiones acerca de las mismas. Una historia de usuario no se considera completa hasta que no supera sus pruebas de aceptación. Esto significa que debe desarrollarse un nuevo test de aceptación para cada iteración o se considerará que el equipo de desarrollo no realiza ningún progreso.

III. Resultados

Este artículo presenta un curso e-learning para el mejoramiento de las competencias de cooperación en el desarrollo de proyectos de software libre. El resultado de la revisión de la literatura fue un listado de competencias de cooperación para las siguientes temáticas: cooperación en el desarrollo de proyectos de software libre, teorías de la cooperación en general y la programación extrema que fueron el insumo principal para el diseño de las unidades del curso virtual.

3.1 Teorías de la cooperación

La tabla 1, presenta las competencias para la cooperación a partir de la revisión de las teorías de la cooperación y que se basan en las siguientes fuentes (Kollock, 1998), (Ostrón, 2001) y (Ostrom, 1997).

Competencias	Descripción
Evitar estrategias individuales	Lograr que todos obtengan un beneficio neto positivo si se coopera
Contribuir o mantener bienes comunes	Cooperar para disfrutar del bien por igual y hay un beneficio inmediato que produce un costo compartido por todos
Desarrollar una identidad de grupo	Creación y refuerzo de las normas compartidas sobre intereses y recursos comunes
Conocer información acerca de acciones del pasado de los otros individuos	Conocer acciones anteriores de los individuos
Crear grupos pequeños	Permiten una fácil comunicación y la cooperación.
Comunicar ideas cara a cara	Para encontrar intereses comunes y evitar que no haya cooperación cuando se trabaja en la construcción de un bien público.
Promover la confianza en grupo	Compromiso mutuo, expectativas que los individuos tienen sobre el comportamiento de los demás.
Promover la reciprocidad en grupo	Correspondencia mutua, creación y refuerzo de las normas que los individuos aprenden de la socialización y las experiencias de la vida.
Obtener reputación a partir de la confianza y la reciprocidad.	Los individuos crean identidades que proyectan sus intenciones y normas.

Tabla 1: Competencias para la cooperación usando teorías de la cooperación

3.2 Cooperación en el desarrollo de software libre

La tabla 2, presenta las competencias para la cooperación a partir de la revisión de la cooperación en el desarrollo de software libre y que se basan en las siguientes fuentes (Raymond, 1999), (González Barahona, J., *et ál*., 2003), (Mockus et ál., 2002), (Koch, Schneider, 2002) y (Elliott, Scacchi, 2004).

Competencias	Descripción
Acceder libremente al código fuente	Para estudiar, adaptar y redistribuir, permitiendo colaborar en comunidad.
Participar para aprender y desarrollar nuevas habilidades	Estímulo para participar en proyectos de software libre
Cambiar los roles de los participantes continuamente	Como se tiene acceso al código fuente, un participante puede trabajar inicialmente en un módulo y luego pasar a trabajar en otro.
Publicar versiones de prueba de forma temprana y frecuente	Permite llamar la atención de otros desarrolladores que tengan el mismo problema y que puedan estar interesados en la solución.
Integrar el código fuente frecuentemente	Evita una última fase de integración de los módulos que componen el programa.
Realizar pruebas unitarias al código fuente	El usuario será el que tome el software y lo pruebe en su máquina bajo unas condiciones específicas, minimizando el esfuerzo para el equipo de desarrollo de probar en diferentes arquitecturas
Crear grupos pequeños	Los programadores hacen un mayor porcentaje de su trabajo total, en archivos de mayor tamaño y se realiza cooperación dentro de este círculo reducido de personas.
Comunicar mediante el uso del computador	Se facilita el trabajo en equipo, la resolución de conflictos, y la cooperación para construir comunidad. Uso de Internet Relay Chat (IRC), CVS y las listas de correo.
Crear comunidades	Para el desarrollo de una identidad de grupo.

Tabla 2: Competencias para la cooperación en el desarrollo de software libre

3.3 Programación extrema

La tabla 3, presenta las prácticas y competencias para la cooperación a partir de la revisión de las prácticas de la programación extrema que son de fácil adaptación en los proyectos de software libre y que se basan en las siguientes fuentes (Aycart Pérez et ál., 2007), (Lindstrom, Jeffries, 2004), (Beck, 1999) y (Beck, 2000).

Prácticas	Competencias	Descripción
XP funciona mejor con pocos desarrolladores.	Cambiar los roles de los participantes continuamente.	Las rotaciones en grupos de más de 10 personas podrían ser un verdadero problema.
Potenciar al máximo el trabajo en equipo (comunicación).	Desarrollar una identidad de grupo	El cliente es uno más que se sienta con el equipo de desarrollo y trabaja con ellos a diario. Se crean identidades que proyectan intensiones y normas.
La propiedad colectiva del código fuente	Acceder libremente al código fuente	Cada programador mejora cualquier código en cualquier lugar en el sistema en cualquier momento si se ve la oportunidad.
Refactorizar	Aplicar la refactorización en el código fuente.	Permite modificar o ampliar partes de código ya escritas anteriormente dejándolo igual de simple pero con la nueva funcionalidad añadida.
Pequeñas entregas	Publicar versiones de prueba de forma temprana y frecuente	Liberaciones tempranas y frecuentes de versiones de prueba.
Pruebas unitarias	Realizar pruebas unitarias al código fuente.	Realizar pruebas unitarias dado que es muy habitual modificar código escrito por otros. Mayor confianza en el código.
El cliente está siempre disponible.	Integrar al cliente en el equipo de desarrollo del software.	Todas las fases que se realizan en un proyecto XP requieren de comu- nicación con el usuario, preferiblemente cara a cara, sin intermediarios.
Diseño simple	Realizar diseños simples	Tratar siempre de realizar las cosas de la manera más sencilla posible. Si alguna parte de la implementación resulta especialmente compleja, se debe replantear (divide y vencerás).
Estándares de programación.	Desarrollar el código fuente con estándares de programación	Es indispensable que se sigan ciertos estándares de programación para mantener el código legible por la propiedad colectiva del mismo.
Integrar frecuentemente.	Integrar el código fuente frecuentemente	Los programadores deberán actualizar sus módulos con las versiones más recientes. Trabajar siempre con la última versión.

Tabla 3: Prácticas y competencias para la cooperación usando Programación Extrema

3.4 Diseño del curso virtual

El diseñodel curso E-learning tuvo varias fases en las que se desarrolló un material didáctico para un entorno virtual de aprendizaje. La figura 3, muestra el modelo de diseño instruccional ADDIE –Acrónimo de Analysis (análisis), Design (diseño), Development (desarrollo), Implementation (implementación) y Evaluation (evaluación)– que fue el utilizadocomo base para describir el proceso de diseño del curso y que se caracteriza por ser tanto interactivo como recursivo, en donde los resultados de la evaluación formativa de cada fase pueden conducir al diseñador instruccional de regreso a cualesquiera de las fases previas [45].

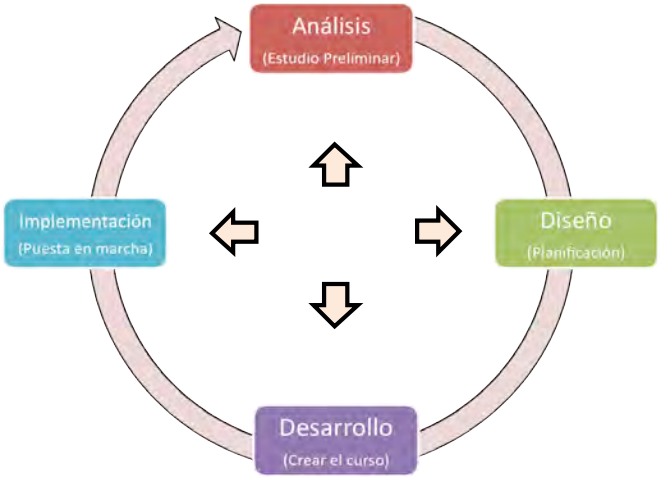

Figura. 3. Modelo de diseño instruccional ADDIE. Fuente: (P. Williams, et ál., 2003)

Para la estructura delcurso E-learning se tuvo en cuenta el aporte de Puello & Barragán [46]. La figura 4, muestra la estructura del curso virtual y comprende la organización del curso, las unidades de aprendizaje y las herramientas educativas para un entorno de aprendizaje integrado. Cada unidad del curso virtual se organizó en un paquete SCORM construido con el software eLearning XHTML editor eXe Version 1.04.1. Para el curso se aplicó un modelo de navegación secuencial, iniciando con los contenidos, las actividades, la evaluación que incluye retroalimentación. El modelo pedagógico utilizado en el curso fue el constructivista [47, 48], ya que se ajusta al modelo de producción en red de los desarrolladores de software libre permitiendo tener un ambiente virtual que es gestionado completamente por el participante y explota las funciones del aprendizaje en línea, teniendo el participante un rol activo [49, 50]. El curso virtual se desarrolló sobre la plataforma Moodle y se utilizaron varias de sus herramientas educativas para facilitar la motivación y participación del estudiante a través de elementos como los foros, la wiki y otros recursos multimedia que permiten garantizar la compresión y feedback de los contenidos del curso.

Figura. 4. Estructura del curso virtual. Fuente: (Autor)

La figura 5, presenta el curso E-learning que se encuentra alojado en plataforma virtual sobre Moodle de la Universidad Francisco de Paula Santander, Ocaña. Se puede acceder a través del siguiente link: http://uvirtual.ufpso.edu.co/login/index.php, colocando como nombre de usuario cursovirtual y como contraseña cursovirtual.

Figura 5. Curso e-learning Fuente: (Autor)

IV. Discusión

La tabla 4, ilustra la confrontación de los resultados obtenidos con los resultados similares de la literatura, entre el software libre (SL), las teorías de la cooperación en general y la programación extrema (XP) lo que representa un sistema de competencias básicas para la cooperación aplicable en el desarrollo de proyectos de software libre.

En un ambiente de cooperación para contribuir o mantener bienes comunes se encuentran elementos que modifican su estado y que están precedidos por conflictos de racionalidad individual y grupal. Esto lo confirma (Kollock, 1998), indicando que los conflictos con el uso de bienes públicos lleva a que los participantes no siempre asuman un costo compartido y quieran usar los recursos sin cooperar. (Ostrom, 1997), ilustra en las relaciones básicas con variables estructurales de la figura 1, que para alcanzar niveles de cooperación con beneficios netos para todos los participantes se deben desarrollar normas que al compartirlas y al tener la seguridad de la actuación de los demás se logra reciprocidad, logrando que cada participante acumule información que genera confianza acerca de sus acciones pasadas y que esto se puede lograr con la comunicación cara a cara que es uno de los componentes facilitadores para la cooperación. En la tabla 1 se ilustraron aquellos elementos que se transforman en competencias básicas para la cooperación.

Las prácticas de cooperación que los desarrolladores de software libre utilizan en sus proyectos se ilustraron en la tabla 2 y estas se afirman a partir de diferentes aportes. El punto de partida para la cooperación en el desarrollo de software libre es la disponibilidad del código fuente, y esta surge como la necesidad que tiene un desarrollador para solucionar un problema propio. De esta forma una vez tenga algo usable funcional y sencillo comparte la solución con la comunidad de software libre, pues en palabras de (Raymond, 1999) varias cabezas piensan más que una. (Koch, Schneider, 2002), concluyen cómo se evidencia poca cooperación en el desarrollo de archivos básicos y cómo los desarrolladores hacen un mayor porcentaje de trabajo en archivos de mayor tamaño y complejidad evidenciándose cooperación en este círculo reducido de personas. La comunicación mediada por ordenador en un ambiente en el que los desarrolladores se encuentran dispersos geográficamente en el mundo, facilitan la cooperación a través del uso de herramientas de Internet como las listas de correo, el sistema de control de versiones y el chat entre otros que permiten la creación de comunidades en línea con sus propias normas generando una identidad de grupo.

La programación extrema es una metodología ágil de desarrollo de software que utiliza prácticas cercanas al desarrollo de proyectos de software libre y que se caracteriza por potenciar al máximo el trabajo en equipo donde el cliente hace parte del mismo. (Beck, 1999), explica las prácticas de XP y cómo se ejecutan en las diferentes fases de la metodología y se puede encontrar que efectivamente existen prácticas de fácil adaptación para potenciar la cooperación en los proyectos de software libre, como se ilustró en la tabla 3. Las intrínsecas al software libre son la propiedad colectiva del código fuente, las pequeñas entregas y los estándares de programación. Hay otras que se pueden adoptar de forma parcial partiendo del hecho de que el desarrollador es el mismo cliente como la refactorización, el cliente en sitio, las pruebas unitarias y la integración frecuente.

Competencia	Software Libre (SL)	Teoría Cooperación	Programación Extrema (XP)	Diferencias	Sistema de Competencias
Evitar estrategias individuales	X	X			X
Contribuir o mantener bienes comunes	X	X			X
Desarrollar una identidad de grupo	X	X			X
Conocer información acerca de acciones del pasado de los otros individuos		X			
Crear grupos pequeños	X	X	X		X
Comunicar ideas cara a cara	X	X	X	En SL a través de Internet	X
Promover la confianza en grupo	X	X			X
Promover la reciprocidad en grupo	X	X			X
Obtener reputación a partir de la confianza y la reciprocidad.	X	X			
Acceder libremente al código fuente	X		X		X
Participar para aprender y desarrollar nuevas habilidades.	X				

Esta tabla continúa en la siguiente página —>

Competencia	Software Libre (SL)	Teoría Cooperación	Programación Extrema (XP)	Diferencias	Sistema de Competencias
Cambiar los roles de los participantes continuamente.	X		X	En XP facilita la rotación de personal	X
Publicar versiones de prueba de forma temprana y frecuente.	X		X		X
Integrar el código fuente frecuentemente	X		X		X
Realizar pruebas unitarias al código fuente	X		X		X
Comunicar mediante el uso del computador	X		X	En XP en parejas	X
Crear comunidades	X	X		En SL a través de Internet	X
Aplicar la refactorización en el código fuente			X		
Integrar al cliente en el equipo de desarrollo del software	X		X	En SL desarrollador es el mismo usuario	X
Realizar diseños simples			X		
Desarrollar el código fuente con estándares de programación	X		X		X

Tabla 4. Sistema de competencias para la cooperación

Este trabajo ofrece una respuesta a la pregunta sobre cuáles son las competencias para mejorar la cooperación en el desarrollo de proyectos de software libre orientado a los lineamientos metodológicos de la programación extrema. El sistema de competencias ofrece un aporte al trabajo de Scacchi [51] y Berrueta [52] sobre estudios a prácticas de proyectos de software libre y prácticas de la pprogramación extrema cercanas al software libre respectivamente.

V. Conclusiones

Este artículo presentó un sistema de competencias para la cooperación aplicables en el desarrollo de proyectos de software libre. El sistema de competencias se encuentra integrado en un curso E-learning para el mejoramiento de las competencias de cooperación de los desarrolladores de software libre.

El sistema de competencias presenta las prácticas de la programación extrema que son totalmente compartidas por el modelo de desarrollo del software libre y el sistema como una unidad, muestra cómo a través de la cooperación se pueden resolver los conflictos de racionalidad individual para el bienestar colectivo de los equipos de desarrollo de software libre.

El sistema de competencias para la cooperación se convierte en una buena herramienta para diseñar objetos virtuales de aprendizaje que faciliten y promuevan la adopción de prácticas de cooperación en software libre.

Referencias

[1] R. Krishna. Analysis of new paradigms for the development and application in free and open source software Engineering. 2009. http://krishnarajpm.com/pdf/foss.pdf - 10/01/2013.

[2] G. Robles et ál. Who is doing it? knowing more about libre software developers. 2001. http://widi.berlios.de/paper/study.pdf - 28/02/2013.

[3] P. Himanen. The Hacker Ethic and the Spirit of the Information Age. Random House. 2002.http://eprints.rclis.org/12851/1/pekka.pdf - 10/02/2013.

[4] Free Software Foundation, 1985. The GNU manifesto. http://www.gnu.org/philosophy/fs-motives.es.html - 08/12/2012.

[5] A. Cox. Cathedrals, Bazaars and the Town Council.Slashdot. 1998. http://news.slashdot.org/story/98/10/13/1423253/featurecathedrals-bazaars-and-the-town-council-14/12/2012.

[6] A. Rishab, et ál..«Free/libre and open source software: Survey and study - part IV: Survey of developers». International Institute of Infonomics University of Maastricht, The Netherlands. 2002, pp. 8-46.

[7] E. Raymond. *The Cathedral and the Bazaar*. Knowledge, Technology & Policy, Fall99, Vol. 12 Issue 3.1999, pp. 23-49.

[8] G. Robles. Ingeniería del software libre. Una visión alternativa a la ingeniería del software tradicional. 2002. http://es.tldp.org/Presentaciones/200211hispalinux/robles/robles-ponencia-hispalinux-2002.pdf- 24/11/2012.

[9] M. Marco. Free software contributions to improve traditional software management projects. Universitat Oberta de Catalunya. 2005. http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=AF62D346BA8B0AAA1430719451D41259?doi=10.1.1.107.6426&rep=rep1&type=pdf - 27/03/2013.

[10] A. Narduzzo, A. Rossi. Modularityin action: Gnu/Linux and free/open source software development model unleashed. 2003. http://opensource.mit.edu/papers/narduzzorossi.pdf - 02/31/2013.

[11] J. Ehrenkrantz. «Release management within opensource projects». En: Proceedings of the 3rd Works-hop on Open Source Software Engineering at the 25th International Conference on Software Engineering, USA: Portland, 2003.

[12] J. González Barahona, et ál. *Introducción al software libre: el desarrollador y sus motivaciones*. Universitat Oberta de Catalunya. 2003. pp. 91-245.

[13] E. Ostrom. *Social dilemmas and human behavior.* Economics in Nature. Social Dilemmas. Mate Choice and BiologicalMarkets. 2001, pp.23-41.

[14] M. Foddy. *Resolving social dilemmas: dynamics, structural, and intergroup aspects*. Psychology Press. 1999.

[15] J. Parra. Constructo para la evaluación de la cooperación en dilemas sociales de gran escala. Ph. D. thesis, Universidad Nacional de Colombia. Doctorado en Ingeniería, área Sistemas. 2010.

[16] P. Kollock. «Social dilemmas: The anatomy of cooperation». *Annual Review of Sociology*, Vol. 24, n.° 1, 1998. pp. 183-214.

[17] E. Ostrom. *Governing the commons: The evolution of institutions for collectiveaction*. Cambridge University Press. 1990.

[18] A. Biel et ál. *Norm perception and cooperation in large scale social dilemmas. Resolving social dilemmas: Dynamic, structural, and intergroup aspects*. 1999. pp. 245-252.

[19] J. Walker, E. Ostrom. «Trust and reciprocity as foundations for cooperation: Individuals, institutions, and context». In *Capstone Meeting of the RSF Trust Initiative at the Russell Sage Foundation*. 2007.

[20] J. Cardenas. «How do groups solve local commons dilemmas? Lessons from experimental economics in the field». *Environment, Development and Sustainability*, 2 (3). 2000. pp. 305-322.

[21] E. Ostrom et ál. *Rules, games, and common-pool resources*. University of Michigan Press. 1994.

[22] G. Hardin. «The tragedy of the commons». *Journal of Natural Resources Policy Research*, 1 (3). 2009, pp. 243-253.

[23] G. Hardin. «The tragedy of the commons. The population problem has no technical solution; it requires a fundamental extension in morality». *Science* (New York, NY), 162 (859), 1243. 1968.

[24] C. Baldwin, K. Clark. «The architecture of participation: Does code architecture mitigate free riding in the open source development model?». *Management Science, 52,* 7, 2006. pp. 1116-1127.

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

[26] E. Ostrom. «A Behavioral Approach to the Rational Choice Theory of Collective Action: Presidential Address, American Political Science Association. The American Political». *Science Review*, Vol. 92, n.° 1, 1997, pp. 1-22.

[27] S. Koch, G. Schneider.«Effort, co-operation and co-ordination in an open source software project: GNOME». *Information Systems Journal. Vol.* 12, n.° 1, 2002. pp. 27-42.

[28] S. Koch, G. Schneider. «Results from softwareengineering research into open source development projects using public data, DiskussionspapierezumTätigkeitsfeldInformationsverarbeitungundInformationswirtschaft». En: H.R. Hansen, W.H. Janko (Hrsg.), *Wirtschaftsuniversität Wien,* n.°. 22, 2002.

[29] D. Germán. «The evolution of gnome». En: Proceedings of the 2nd Workshop on Open Source Software Engineering at the 24th International Conference on Software Engineering, USA: Florida. 2002.

[30] A. Mockus. «Acase study of open source software development: the apache server». En: Proceedings of the 22nd International Conference on Software Engineering, Irlanda: Limerick, ACM Press. 2000.

[31] J. Roberts et ál. *Understanding the motivations, participation, and performance of open source software developers: A longitudinal study of the Apache projects*. 2006.

[32] M. Elliott, W. Scacchi. «Free software development: Cooperation and conflict in a virtual organizational culture», Free/Open Source Software Development. 2004. pp. 152-172.

[33] W. Scacchi. Understanding open source software evolution:Applying, breaking and rethinking the laws of software evolution. 2003. http://www.ics.uci.edu/~wscacchi/Papers/New/Understanding-OSS-Evolution.pdf - 11/04/2013.

[34] L. Lindstrom, R. Jeffries. «Extreme Programming and Agile Software Development Methodologies». *Information Systems Management*, Vol. 21, Issue 3, 2004. pp. 41-52.

[35] R. Pressman. *Ingeniería de software un enfoque práctico*. Madrid. 5.ª edición, McGraw-Hill, 2004. pp. 55-66.

[36] D. Aycart Pérez et ál. *Ingeniería del software en entornos de software libre*. Universitat Oberta de Catalunya, 2007, pp. 28-37.

[37] R. Jeffries et ál. *Extreme Programming Installed*. Addison Wesley Longman, 172. 2001.

[38] K. Beck, M. Fowler. *Planning Extreme Programming*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA. 2000.

[39] J. González, G. Robles.«Unmounting the «code god» assumption». En: *Proceedings of the Fourth InternationalConference on extreme Programming and Agile Processes in Software Engineering*, Italia: Génova, 2003.

[40] K. Beck. *Embracing change with extreme programming*. Computer, Vol. 32, n.º 10, 1999, pp. 70-77.

[41] K. Beck. *Extreme Programming Explained: Embrace Change*. Adison-Wesley, 2000.

[42] M. Fowler. *Refactoring -Improving the design of existing code*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA. 2000.

[43] W. Lewis. *Software Testing and Continuous Quality Improvement*. CRC Press LLC. 2000.

[44] G. Meyers. The Art of Software Testing. John Wiley & Sons. 2000.

[45] P. Williams et ál. Modelos de diseño instruccional. Material didáctico web de la UOC. 2003. Publicación en línea. 17/03/2013.

[46] J. Puello, R. Barragán. Un modelo para el diseño de cursos virtuales de aprendizaje por competencias y basados en estándares de calidad. 2008. www.revistas.unal.edu.co/index.php/email/article/view/12624- 22/02/2013.

[47] J. Stephenson, A. Sangrà. *Modelos pedagógicos y e-learning: La pedagogía en el e-learning*. Universitat Oberta de Catalunya. 2003, pp. 13-28.

[48] S. Hernández Requena. «El modelo constructivista con las nuevas tecnologías: aplicado en el proceso de aprendizaje». En: Comunicación y construcción del conocimiento en el nuevo espacio tecnológico, *Revista de la Universidad y Sociedad del Conocimiento (RUSC)*. Vol. 5, n.º 2. UOC. 2008.

[49] D. Argüelles, N. Nagles. *Estrategias para promover procesos de aprendizaje autónomo*. Bogotá: Alfaomega, 2007, pp. 302-306.

[50] M. Unirrago. *Educación virtual encuentro formativo en el ciberespacio*. Bucaramanga: UNAB, 2001, pp. 123-132.

[51] W. Scacchi et ál. *Understanding Free/Open Source Software Development Processes. Software process improvement and practice*. 2006. pp.95-105.

[52] D. Berrunta. Programación extrema y software libre. 2006. http://es.scribd.com/doc/110617623/Ponencia-Sl-y-Xp.- 10/12/2013.