XXX aniversario
novática
UPGRADE Revista de la Asociación de Técnicos de Informática

Nº 175, mayo-junio 2005, año XXXI

sumario

Monografía del próximo número:
"Estandardización y normalización en Seguridad"

Novática, revista fundada en 1975 y decana de la prensa informática española, es el órgano oficial de expresión y formación continua de **ATI** (Asociación de Técnicos de Informática). ***Novática*** edita también ***UPGRADE***, revista digital de **CEPIS** (*Council of European Professional Informatics Societies*), en lengua inglesa, y es miembro fundador de **UPENET** (***UP****GRADE* ***E****uropean* ***NET****work*)

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

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

Coordinación Editorial, Redacción Central y Redacción ATI Madrid
Padilla 66, 3º, dcha., 28006 Madrid
Tlfn.914029391; fax.913093685 <novatica@ati.es>
Composición, Edición y Redacción ATI Valencia
Av. del Reino de Valencia 23, 46005 Valencia
Tlfn./fax 963330392 <secreval@ati.es>
Administración y Redacción ATI Cataluña
Ciudad de Granada 131, 08018 Barcelona
Tlfn.934125235; fax 934127713 <secregen@ati.es>
Redacción ATI Andalucía
Isaac Newton, s/n, Ed. Sadiel,
Isla Cartuja 41092 Sevilla, Tlfn./fax 954460779 <secreand@ati.es>
Redacción ATI Aragón
Lagasca 9, 3-B, 50006 Zaragoza.
Tlfn./fax 976235181 <secreara@ati.es>
Redacción ATI Asturias-Cantabria <gp-astucant@ati.es>
Redacción ATI Castilla-La Mancha <gp-clmancha@ati.es>
Redacción ATI Galicia
Recinto Ferial s/n, 36540 Silleda (Pontevedra)
Tlfn.986581413; fax 986580162 <secregal@ati.es>
Suscripción y Ventas
<http://www.ati.es/novatica/interes.html>, o en ATI Cataluña o ATI Madrid
Publicidad
Padilla 66, 3º, dcha., 28006 Madrid
Tlnf.914029391; fax.913093685 <novatica.publicidad@ati.es>
Imprenta
Derra S.A., Juan de Austria 66, 08005 Barcelona.
Depósito legal: B 15.154-1975 -- ISSN: 0211-2124; CODEN NOVAEC

Portada: Antonio Crespo Foix / © ATI 2005
Diseño: Fernando Agresta / © ATI 2005

Agilidad y desarrollo de Software Libre

Alberto Sillitti, Giancarlo Succi
Freie Universität Bozen / Libera Università di Bolzano (Italia)

<{Alberto.Sillitti,Giancarlo.Succi}@unibz.it>

Traducción: Rafael Martínez Martínez (Grupo de Lengua e Informática de ATI)

***Resumen**: las metodologías de desarrollo ágiles y el Software Libre son enfoques muy conocidos para el desarrollo de software. Aunque son muy diferentes, presentan muchas concordancias como, por ejemplo, los principios y valores básicos. En particular, hay muchas analogías entre el desarrollo de Software Libre y la programación extrema (enfoque al código e inclusión de cambios, por citar algunas). Este artículo presenta estos principios y valores básicos e identifica las concordancias entre ambas metodologías.*

***Palabras claves**: metodologías ágiles, programación extrema, Software Libre.*

Autores

Alberto Sillitti es Doctor en Ingeniería y Profesor Adjunto en la *Libera Università di Bolzano* (Italia). Participa en varios proyectos financiados por la Unión Europea en el área de la Ingeniería del Software relacionados con las metodologías ágiles y el software de código abierto. Sus áreas de investigación incluyen la Ingeniería del Software, la Ingeniería del Software basada en componentes, la integración y métricas de los servicios web, metodologías ágiles y código abierto.

Giancarlo Succi es Doctor en Ingeniería, Profesor de Ingeniería del Software y Director del Centro para Ingeniería Aplicada del Software en la *Libera Università di Bolzano* (Italia). Sus áreas de investigación incluyen las metodologías ágiles, el desarrollo de código abierto, Ingeniería empírica del Software, líneas del producto de software, reutilización del software e Ingeniería del Software aplicada a Internet. Es autor de más de 100 artículos publicados en conferencias y revistas internacionales, así como de un libro.

1. Introducción

Las metodologías ágiles (AMS, *Agile Methods*) se han desarrollado con mucho éxito en los últimos años [3] al igual que el Software Libre (SL) [1][8]. Aunque estos enfoques de desarrollo de software parecen muy diferentes, presentan muchas concordancias, como demostró Koch [11].

Tanto las AMs como el SL empujan hacia una organización menos formal y jerárquica en el desarrollo de software y más centrada en la persona, con un énfasis mayor en:

- centrarse en el objetivo principal del desarrollo — producir un sistema de gestión con la cantidad correcta de funcionalidades. Esto significa que el sistema final tiene que incluir sólo el mínimo número de características necesarias para satisfacer por completo al cliente real.
- eliminar actividades que se relacionaron con algunos documentos 'formales' de especificaciones que no tienen una relación directa clara con el resultado final del producto.

Este enfoque está claramente vinculado a la "gestión ligera" (*Lean Management*) [16]. Las AMs reconocen explícitamente sus vínculos con la gestión ligera [13], mientras que el SL los mantiene implícitos.

Más aún, el desarrollo de software mediante las AMs o el SL parecen similares bajo varios puntos de vista, como éstos:

1. Los orígenes de ambos enfoques son bastante antiguos, pero ahora se han revitalizado con renovado interés, como reconoce explícitamente Beck [4] para las AMs (en particular la programación extrema — XP, *eXtreme Programming*) y prueba Fuggetta para el SL [10].
2. Ambos son rompedores [6], en el sentido de que alteran valores establecidos en la producción de software.
3. Ambos han tenido éxito, mientras que enfoques más tradicionales han fallado (el proyecto C3 para AMs [4] y el navegador de Mozilla/Firefox para el SL [7][12])).
4. Los promotores de las AMs también están participando en el desarrollo del SL (por ejemplo, Beck con JUnit).

Este artículo aspira a proporcionar una visión general de las concordancias entre las metodologías ágiles (XP, *eXtreme Programming*, en particular) y el SL desde el punto de vista de los principios fundamentales y de los valores que ambas comunidades comparten.

El artículo está organizado como sigue: la **sección 2** identifica los principios ágiles generales para el SL; la **sección 3** se centra en los valores y principios específicos de XP y los identifica en el campo del SL; finalmente, la **sección 4** traza las conclusiones y propone posteriores investigaciones.

2. Principios ágiles en el Software Libre

Los principios básicos compartidos por todas las AMs están enumerados en el llamado *Agile Manifesto* [2]. La **tabla 1** identifica los principios de las AMs en el SL.

En conjunto, es evidente que el SL adopta muchos de los valores fomentados por los partidarios de las AMs.

Tales evidencias reclaman un análisis subsecuente para determinar el grado y la profundidad de dicha adopción. Por otra parte, las AMs y el SL son clases de métodos de desarrollo de software que incluyen un amplio número de métodos específicos.

Por tanto, es importante considerar los casos específicos para determinar cómo se dan realmente en la práctica las interacciones entre las AMs y el SL, más allá de consideraciones que, por sí solas, resultan bastante inútiles al final.

3. Valores y principios de XP en el Software Libre

En general, además de las concordancias entre el SL y las AMs, es interesante analizar estas relaciones entre el SL y una de las más populares metodologías ágiles: la programación extrema (XP, *eXtreme Programming*).

XP está centrada en cuatro valores principales (hay una exposición my amplia en las dos ediciones del libro de Beck [4][5]):

1. **Comunicación**: los desarrolladores necesitan intercambiar información e ideas sobre el proyecto, a los directivos, y a los clientes de forma honrada, confiable y fácil. La información debe fluir de manera continua y rápida.
2. **Sencillez**: siempre que sea posible hay que elegir soluciones simples. Esto no significa estar equivocado o aplicar enfoques simplistas. Beck utiliza a menudo el siguiente aforismo "*simple pero no demasiado simple* ".
3. **Retroalimentación**: en todos los niveles las personas deberían obtener una retroalimentación muy rápida sobre lo que hacen. Los clientes, los directivos y los desarrolladores tienen que alcanzar una compren-

Principios de las AMs	Concordancias con el Software Libre
Personas e interacciones por encima de procesos y herramientas	El proceso del desarrollo en comunidades de código abierto sin duda pone más énfasis en la persona y la interacción que en procesos y herramientas. Las interacciones de las comunidades de código abierto acostumbran a basarse principalmente en el correo electrónico; sin embargo, el orgullo y la individualidad del programador se hacen predominantes, mientras que en las metodologías ágiles hay una fuerte tendencia a establecer un espíritu de equipo entre los desarrolladores.
Software funcionante por encima de documentación omnicomprensiva	Tanto las metodologías ágiles como el código abierto ven el código que funciona bien como la fuente principal de documentación. En las comunidades de código abierto las formas más comunes de documentación de usuario son las capturas de pantalla y los foros de usuarios (Twidale y Nichols [15]), procedentes ambos del uso directo de los sistemas, y la fuente más común de la documentación técnica son las jerarquías de clases extraídas directamente del código fuente, de las bases de datos de seguimiento de errores, y de las salidas entre diferentes versiones.
Colaboración con el cliente por encima de la negociación del contrato	En el Software Libre los clientes y los desarrolladores coinciden a menudo. Esto era especialmente cierto en los inicios del SL, cuando se decía abiertamente por ejemplo, que Unix (y después Linux y las herramientas de GNU) eran sistemas desarrollados por desarrolladores para desarrolladores. En estos casos, los sistemas son claramente conducidos por el cliente. Hay ahora situaciones donde los clientes están claramente separados de los desarrolladores. Nuevos sistemas como Subversion, ArgoUML, etc., tienen una clara orientación al cliente, separada de los desarrolladores. Todavía, viendo cómo se producen los lanzamientos de las versiones, se agregan funcionalidades y se resuelven los problemas, parece que el sistema se desarrolla centrándose en la colaboración con el cliente. Por otra parte, en Europa se está haciendo muy popular la reivindicación de de que los sistemas desarrollados con fondos públicos sean realizados con licencias de Software Libre de diferentes categorías.
Responder al cambio por encima del seguimiento de una planificación	Siguiendo la discusión del punto anterior "Colaboración con el cliente por encima de la negociación del contrato", la evolución de un proyecto de código abierto es típicamente conducido por el cliente. Parece que los sistemas de SL no tienen un "gran diseño inicial"; se promueven desde abajo en vez de desde arriba y su evolución depende de las verdaderas necesidades de los clientes. Sin embargo, muchos de los análisis están basados en situaciones donde coinciden clientes y desarrolladores. Sería interesante ver cómo evolucionará esta situación en los nuevos escenarios los clientes están separados de los desarrolladores.

Tabla 1. Principios de las AMs en el Software Libre.

sión común de la meta del proyecto, y también acerca del estado actual del proyecto, sobre qué necesitan realmente los clientes en primer lugar primero y sobre sus prioridades, y qué desarrolladores pueden hacerlo y en que tiempo. Esto está fuertemente conectado con las comunicaciones. También debería haber una retroalimentación inmediata del trabajo que está haciendo la gente, es decir, del código que se está produciendo – todo lo cual exige pruebas, integraciones, versiones y entregas frecuentes.

4. Valor: cada persona implicada en el proyecto debería de tener el valor (y el derecho) de expresar su valoración sobre el proyecto. Todos deberían de tener el valor de ser abiertos y dejar que todos examinasen e incluso modificasen su trabajo. Los cambios no deberían ser vistos con terror y los desarrolladores deberían tener el valor de encontrar mejores soluciones y modificar el código siempre que sea necesario y factible.

Estos valores están presentes de varias maneras en la descripción de Raymond de *Open Source* [14], resumidos en la **tabla 2**.

Por otra parte, según lo observado en [9], ocultos en el interior de la primera versión del libro de Beck [4] hay 15 principios, divididos en 5 principios fundamentales y otros 10 principios.

Los principios fundamentales son:

1. Retroalimentación rápida: volviendo al valor de la retroalimentación, ésta debería ocurrir tan pronto como fuera posible, tener el impacto más alto en el proyecto y limitar lo más posible las interrupciones potenciales.

2. Asumir la sencillez: según lo mencionado, la sencillez es un valor muy importante. Por lo tanto, la sencillez debería ser asumida en todas las fases del desarrollo.

3. Cambios incrementales: el cambio (en su mayor parte procedente de la retroalimentación) no debería hacerse todo de una vez. Por consiguiente debería ser un proyecto permanente e incremental, dirigido a crear un sistema evolutivo.

4. Adopción del cambio: el cambio debería ser manejado con valor y no ser evitado. El sistema en su totalidad, y el código, debería ser organizado para facilitar el cambio más amplio posible.

5. Calidad del trabajo: la calidad debería ser la principal preocupación. La carencia de calidad genera revisiones y derroches que deberían ser evitados en la mayor medida posible.

Otros principios de XP son:

1. Enseñe a aprender: la identificación de requisitos es un proceso de aprendizaje global. Por lo tanto, el aprendizaje es de suma importancia en el sistema.

2. Inversión inicial pequeña: el trabajo previo debería ser lo más escaso posible, puesto que subsiguientes cambios pueden destruirlo.

3. Jugar a ganar: todos los desarrollos deberían ser guiados por la clara convicción de qué lo que hacemos es realmente factible.

4. Experimentos concretos: las ideas deberían no ser validadas a través de discusiones largas y teóricas sino vía experimentaciones concretas en el código base.

5. Comunicación abierta, honesta: la comunicación debería ser siempre simple y fácil. El cliente no debería ocultar sus prioridades ni los desarrolladores y directivos deberían ocultar el estado actual del trabajo.

6. Trabajar con los instintos de la gente - no contra ellos: el papel de los directivos es obtener lo mejor de los desarrolladores, así que deberían explotarse las inclinaciones naturales de éstos. Un espíritu de equipo fuerte debería ser aprovechado. Por otra parte, en las relaciones entre los directivos, desarrolladores y clientes no deberían ignorarse los miedos, ansiedades e incomodidades sino ser manejados correctamente.

7. Aceptar responsabilidades: todo el personal del proyecto (clientes, directivos y desarrolladores) debería aceptar voluntariamente sus propias responsabilidades. Tales responsabilidades deberían entonces ser asignadas con completa confianza.
8. Adaptación local: la metodología debería ser adaptada sabiamente a las necesidades de cada contexto de desarrollo.
9. Viaje con poco equipaje: en los proyectos XP es importante mantener la mínima cantidad de documentos posible, evidentemente sin comprometer la integridad del proyecto.
10. Honradez en las métricas: el proyecto debería ser seguido con métricas objetivas y comprensibles. Las métricas deberían ser recogidas mediante un procedimiento ligero que no altere la naturaleza de XP.

En esta sección repasamos la aplicación al código abierto de los principios fundamentales: la retroalimentación rápida, aceptar la sencillez, cambio incremental, aceptar los cambios, la calidad del trabajo.

Hemos discutido ya la aplicación de la **retroalimentación** y la **sencillez** desde punto de vista de Beck. Fowler [9] comparte gran parte del punto de vista de Back y hace énfasis en la mejora continua del código fuente creado para que sea tan sencillo como sea posible.

Respecto a los **cambios incrementales**, Raymond [14] los reconoce abiertamente como uno de sus principios guía desde su temprana experiencia con Unix: "*He estado predicando durante años el evangelio de Unix de herramientas pequeñas, prototipado rápido y programación evolutiva* ".

En lo que se refiere a **aceptar los cambios** propuestos por otros, hemos ya mencionado la opinión de Raymond [14] sobre la necesidad de escuchar a los clientes incluso si no "*te pagan en dinero*". Raymond [14] llega más lejos y en la regla número 12 declara el papel central de la aceptación del cambio: "*a menudo, las más llamativas y más innovadoras soluciones vienen de darse cuenta que tu concepto del problema era incorrecto* ".

Raymond [14] va más lejos que Beck [4] en esta materia. Ambos están de acuerdo en que los prototipos (*spikes* en la jerga de Beck) pueden ser muy útiles para alcanzar una mejor comprensión de un dominio de aplicación complejo. Raymond [14] también proclama que el sistema que se está desarrollando puede ayudar a identificar nuevas ideas para nuevos desarrollos – regla 14: "*cualquier herramienta debería ser útil de la manera prevista, pero una herramienta verdaderamente grande lleva por si misma a usos que usted nunca esperó* ". Es innecesario decir que cuando redactó el borrador de la regla 14 Raymond [14] no se preocupó de asegurar al cliente que no malgastará sus recursos.

Respecto al trabajo de **calidad**, en Raymond [14] no hay una referencia explícita al rol esencial de la calidad como lo hay en [4]. Sin embargo, a lo largo de su ensayo hay una evidencia constante del orgullo que los desarrolladores de código abierto depositan en su código, un orgullo que proviene únicamente de la calidad del trabajo realizado.

Ahora dirigimos nuestra atención a los otros principios: enseñe a aprender; inversión inicial pequeña; jugar a ganar; experimentos concretos; comunicación abierta, honrada; trabajar con los instintos de la gente personal - no contra ellos; aceptación de responsabilidades; adaptación local; vieaje con poco equipaje; honradez en las métricas.

Raymond acentúa el rol de escuchar y aprender de los comentarios de los otros. Sin embargo, no hay una mención explícita a **aprender a enseñar**.

Hay también poca preocupación por no tener una **pequeña inversión inicial** y **viajar con poco equipaje**. La razón es que los

Valores de XP	Concordancias con el Software Libre
Comunicación	El concepto mismo de código abierto está basado en compartir ideas mediante el código fuente, el cual llega a convertirse en un propulsor de la comunicación. Así, sin duda la comunicación es un valor decisivo en el trabajo de Raymond [14]. El rol de las comunicaciones es reforzado por Raymond a lo largo de todo su ensayo [14]. Éste indica claramente la destacada importancia de escuchar a los clientes "*pero si estás escribiendo para el mundo, necesitas escuchar a tus clientes - esto no cambia sólo porque no te están pagando en dinero*." De ello se evidencia que para dirigir un buen proyecto de código abierto son muy importantes la buena comunicación y las habilidades personales: presenta como ejemplos a Linus Torvald y a sí mismo, supuestamente, dos personas capaces de motivar y de comunicar.
Sencillez	La sencillez del sistema es muy apreciada en la comunidad de código abierto. En general, Raymond [14] menciona la "pereza constructiva," que ayuda a encontrar soluciones preexistentes que pueden adaptarse a las nuevas situaciones. El concepto de sencillez de Beck [4] está claramente reflejado en la regla número 13 de Raymond [14]; es una cita Antoine de Saint'Exupéry: "La perfección (en el diseño) no se alcanza cuando no hay nada más añadir, sino más bien cuando no hay nada más que quitar".
Retroalimentación	Trabajando en una comunidad distribuida, Raymond [14] reconoce el valor de una retroalimentación rápida en todos los niveles: • entre los desarrolladores distribuidos, potencialmente trabajando en la misma solución. • entre los desarrolladores y los clientes -- la regla número 11 es un claro ejemplo: "*lo más próximo a tener buenas ideas es reconocer las buenas ideas de sus usuarios. A veces lo último es mejor*". La retroalimentación se consigue especialmente ejecutando y probando el código, por ello es por lo que entregas tempranas y frecuentes son útiles – la regla 7 dice "entregue pronto, entregue a menudo. Y escuche a sus clientes." Es innecesario decir que muchos de los comentarios hechos sobre la retroalimentación podían aplicarse también a la comunicación. No es algo complicado. Beck [4] reconoce explícitamente que los dos conceptos se solapan.
Valor	El mérito del valor está menos presente en la presentación de Raymond [14]. Hace alusión al valor cuando presenta la dificultad inicial de conseguir el trabajo expuesto a los "*millares de codesarrolladores impacientes que aporrean a cada nueva entrega*".

Tabla 2. Valores de XP en el Software Libre.

> Las metodologías de desarrollo ágiles y el Software Libre presentan muchas concordancias como, por ejemplo, los principios y valores básicos

proyectos de código abierto son dirigidos sobre todo por desarrolladores, menos inclinados a emplear siglos en la "*parálisis del análisis*" o en producir documentación inútil y están más preocupados por entregar código útil. La atención de Raymond [14] se concentra más bien en probar que se requiere poco trabajo previo. "*Cuando empiezas a construir una comunidad, lo que necesitas es ser capaz de presentar una promesa plausible. Tu programa no tiene que funcionar especialmente bien. Puede ser primitivo, plagado de errores, incompleto y pobremente documentado. Lo qué se tiene que hacer sin falta es (a) ejecutarlo y (b) convencer al resto de potenciales desarrolladores de que puede evolucionar hasta convertirse en algo realmente bueno en un futuro previsible* ".

Jugar a ganar y los **experimentos concretos** son una parte integral de cualquier esfuerzo automotivado, así que no es necesario extenderse en la explicación.

Si hablamos de valores, es evidente el papel que Raymond [14] otorga a **una comunicación abierta y honrada**. Al ser un enfoque centrado en el desar-rollador, el código abierto también aboga por **trabajar con los instintos de la gente - no contra ellos** y confía en la **aceptación de responsabilidades**.

Las primeras dos reglas de Raymond son "*Todo buen trabajo de software se inicia rascando el picor personal de un desarrollador*", y los "*Los buenos desarrolladores saben qué escribir. Los grandes saben qué reescribir (y reutilizar)*". La regla 4 parece también bastante aplicable: "*Si tienes una actitud correcta los problemas interesantes te encontrarán*".

Si bien no hay una métrica formal en el ensayo de Raymond [14], sí que hay un énfasis en hacer entregas de código con frecuencia, de tal manera que queden claros el estado del proyecto y los errores todavía existentes. Esto se asemeja a la **honradez en las métricas**.

4. Conclusiones

En conjunto, observamos que hay un nivel considerablemente alto de solapamiento entre los valores adoptados por las AMs (XP en particular) y los de desarrollo de código abierto según Raymond. La comunicación, la retroalimentación y la sencillez son apoyadas completamente por ambos enfoques. El valor también se supone implícitamente al realizar un proyecto de código abierto.

Yendo a los principios, hay también un buen nivel de acuerdo en los principios fundamentales, aparte de la calidad que se da por supuesta en el trabajo de Raymond, aunque no abogue por ella.

En cuanto a los "demás principios" de XP, las únicas diferencias vienen de los diferentes puntos de vista: Raymond trata sobre todo con voluntarios, mientras que Beck lo hace con empleados. Conceptos tales como viaje con poco equipaje, poco trabajo previo, etc., no preocupan particularmente a Raymond, que, por otra parte, está más interesado en que los desarrolladores de código abierto realicen por lo menos un poco de trabajo previo.

En cuanto a las prácticas, la situación es claramente diferente. Las prácticas relacionadas con el proceso, el entendimiento compartido y el bienestar del programador son un tanto similares en los dos casos. Las prácticas relacionadas con una cuidadosa retroalimentación no están tan extensamente presentes en la descripción de Raymond.

Como nota final, nos gustaría poner de manifiesto que ambas experiencias, la de Beck y la de Raymond, proceden de un uso temprano de lenguajes de programación de muy fácil uso, expresivos y poderosos: Smalltalk y Lisp, respectivamente. Un análisis del papel de los lenguajes de programación en las AMs y en el desarrollo de SL podía ser un tema interesante para otro estudio.

Referencias

[1] P. Abrahamsson, O. Salo, J. Ronkainen. *Agile software development methods*, VTT Publications, 2002. <http://www.inf.vtt.fi/pdf/publications/2002/P478.pdf> [accedido el 15 de junio de 2005].

[2] Agile Alliance, Agile Manifesto, 2001. <http://www.agilemanifesto.org/> [accedido el de junio de 2005].

[3] L. Barnett. "Teams Begin Adopting Agile Processes". *Forrester Research*, noviembre 2004.

[4] K. Beck. *Extreme Programming Explained: Embracing Change*, Addison Wesley, 1999.

[5] K. Beck. *Extreme Programming Explained: Embracing Change*, Second Edition, Addison Wesley, 2004.

[6] C.M. Christensen. *The Innovator's Dilemma*, Harper Business, 2003.

[7] M.A. Cusumano, D.B. Yoffie. *Competing on Internet Time: Lessons From Netscape & Its Battle with Microsoft*, Free Press, 1998.

[8] J. Feller, B. Fitzgerald. *Understanding Open Source Software Development*, Addison-Wesley, 2002.

[9] M. Fowler. *Principles of XP*, 2003. <http://www.martinfowler.com/bliki/PrinciplesOfXP.html> [accedido el 15 de junio de 2005].

[10] A. Fuggetta. "Open Source Software – an Evaluation", *Journal of Systems and Software*, 66(1), 2003.

[11] S. Koch. "Agile Principles and Open Source Software Development: A Theoretical and Empirical Discussion", *5th International Conference on eXtreme Programming and Agile Processes in Software Engineering (XP2004)*, Garmisch-Partenkirchen, Germany, 6 - 10 June, 2004.

[12] S. Krishnamurthy. "The Launching of Mozilla Firefox - A Case Study in Community-Led Marketing", 2005. <http://opensource.mit.edu/papers/sandeep2.pdf> [accedido el 15 de junio de 2005].

[13] M. Poppendieck, T. Poppendieck. *Lean Software Development: An Agile Toolkit for Software Development Managers*, Addison Wesley, 2003.

[14] E.S. Raymond. *The Cathedral and the Bazar*, Version 3.0, 2002. <http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/> [accedido el 15 de junio de 2005]. Publicado también por O'Reilly en 2001.

[15] M.B. Twidale, D.M. Nichols D.M. (2005). ""Exploring Usability Discussions in Open Source Development", *38th Hawaii International Conference on System Sciences*, 2005. <http://csdl.computer.org/comp/proceedings/hicss/2005/2268/07/22680198c.pdf>.

[16] J.P. Womack, D.T. Jones. *Lean Thinking: Banish Waste and Create Wealth in Your Corporation*, Revised and Updated, Free Press, 2003.