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

Nº 177, septiembre-octubre 2005, año XXXI

sumario

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 **UP**GRADE, revista digital de **CEPIS** (*Council of European Professional Informatics Societies*), en lengua inglesa, y es miembro fundador de **UP**ENET (**UP**GRADE ***E**uropean **NET**work*)

<http://www.ati.es/novatica/>
<http://www.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, 3-5, 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

Monografía del próximo número: "*Web Semántica*"

El reto de utilizar software de código abierto como estrategia de reutilización

Christian Neumann, Christoph Breidert
Universidad de Económicas y Administración de Empresas de Viena (Austria)

<{christian.a.neumann, christoph.breidert}@wu-wien.ac.at>

Traducción: César Recasens Salart (Grupo de Lengua e Informática de ATI)

Resumen: *este articulo compara los beneficios de la adopción del software de código abierto en las estrategias de reutilización interna y comercial. Proponemos un tipo de proceso que puede ser utilizado para una evaluación técnica y comercial. Se investigan y comparan las ventajas, desventajas y riesgos de estas estrategias básicas.*

Palabras clave: *reutilización de código, software de código abierto, software comercial empaquetado, software libre.*

1. Introducción

La reutilización de software es una parte importante de la ingeniería de software moderna pues ofrece reducción de costes, reducción del tiempo de salida al mercado y mayor calidad. Un estudio muestra que la productividad puede aumentar hasta un 113%, la proporción media de defectos disminuye el 33% y el tiempo de salida al mercado se reduce el 25% [11]. Otro estudio informa de una reducción de defectos del 51%, incremento de productividad del 57% y de un tiempo de salida al mercado más rápido (mejora del 57%) [16].

La decisión sobre cuándo debe ser adoptada la reutilización dentro del proceso de ingeniería depende de dos elementos principales: los componentes deben cumplir los requisitos técnicos y su utilización debe ser económicamente razonable. Este artículo describe el proceso técnico para seleccionar la mejor alternativa de entre tres estrategias de reutilización elementales: el desarrollo y utilización de componentes propios, la adopción de componentes comerciales externos (también llamados *Commercial-Off-The-Shelf*, COTS) y la integración de software libre y de código abierto (*Free and Open Source Software*, FOSS). Estudios recientes han tratado la utilización de software FOSS y sus efectos en el coste total de propiedad y calidad. Pero estos estudios se centran principalmente en la utilización de sistemas operativos como Linux o software de aplicación como el servidor web Apache. Por ejemplo, un estudio estima que la utilización de software FOSS deviene en el ahorro de 12 millones (una quinta parte del coste de utilización de aplicaciones comerciales) sobre un periodo de cinco años [7]. Otro estudio refiere el ahorro de aproximadamente un tercio en un periodo de dos años [19].

Aparte de esto, apenas existen conocimientos sobre la integración de software FOSS en el proceso de ingeniería de software. El concepto libre FOSS revela nuevas estrategias y modelos de negocio que pueden aplicarse en casi cualquier fase del proceso de desarrollo, como la utilización de FOSS como base de desarrollo para una futura linea de producto o la contribución de componentes software a la comunidad FOSS. El ultimo punto es especialmente interesante para las empresas pues hace posible la externalización del proceso de mantenimiento.

Este artículo se centra en dos modelos de negocio diferentes: vendedores de sistemas que desarrollan software individual por encargo del cliente y compañías de software que desarrollan software estándar.

2. Adaptando estrategias de reutilización

La **figura 1** ilustra el proceso que debe seguirse antes de tomar una decisión. El proceso de toma de decisión se divide en tres fases: prospección, adaptación y comparación. En las dos primeras fases, que se centran en aspectos técnicos, la decisión se toma en base a si un componente provee la funcionalidad deseada o si, en caso contrario, puede ser extendido. En la fase tres los aspectos económicos de los candidatos finalistas se investigan minuciosamente y se examinan las implicaciones de los diferentes modelos de negocio.

2.1. Análisis técnico

Antes que nada deben especificarse los requisitos técnicos del proyecto que, principalmente, serán deducidos de la funcionalidad deseada. Esta especificación contiene la arquitectura, la funcionalidad de entrada y salida (*input/output*, I/O) y la lógica de negocio. Basándose en esta especificación, el desarrollador de software tiene que identificar aquellas partes de la arquitectura de la aplicación en las que sea posible incorporar componentes. Un buen lugar donde empezar la búsqueda son los procedimientos de I/O ya que casi cualquier programa utiliza operaciones con ficheros, accesos a base de datos o una interfaz gráfica (*Graphical User Interface*, GUI).

La primera fase del análisis consiste en una prospección durante la cual se identifican los posibles componentes reutilizables. Fuentes de información para esta tarea son, en el caso de reutilización de componentes internos, los repositorios o el conocimiento implícito de proyectos anteriores. La búsqueda de componentes externos puede hacerse utilizando Internet, preguntando en los grupos de noticias, visitando exposiciones o leyendo revistas. Un buen lugar para empezar la búsqueda de software FOSS son los grandes repositorios en linea al estilo de *Sourceforge*, <http://www.sourceforge.net>, o *Apache*, <http://www.apache.org>. Aquí uno puede buscar software FOSS clasificado por temas, lenguajes de programación o palabras clave. Especialmente la *Apache Foundation* alberga un amplio rango de componentes a punto de ser usados, algunos de los cuales se han convertido en estándares de facto (log4j, xerces, xalan,etc.).

La segunda fase se ocupa de una visión más profunda de la funcionalidad provista por los componentes identificados en la fase uno. La documentación, especificaciones y ejemplos son posibles fuentes de información. Si la funcionalidad provista no cumple con los requisitos técnicos se ha de verificar si los componentes pueden extenderse. Naturalmente esto sólo es aplicable a los componentes FOSS o internos, debido a que los componentes COTS no proveen el código fuente para ser modificado. La utilización de los componentes sin cambios se denomina reutilización de

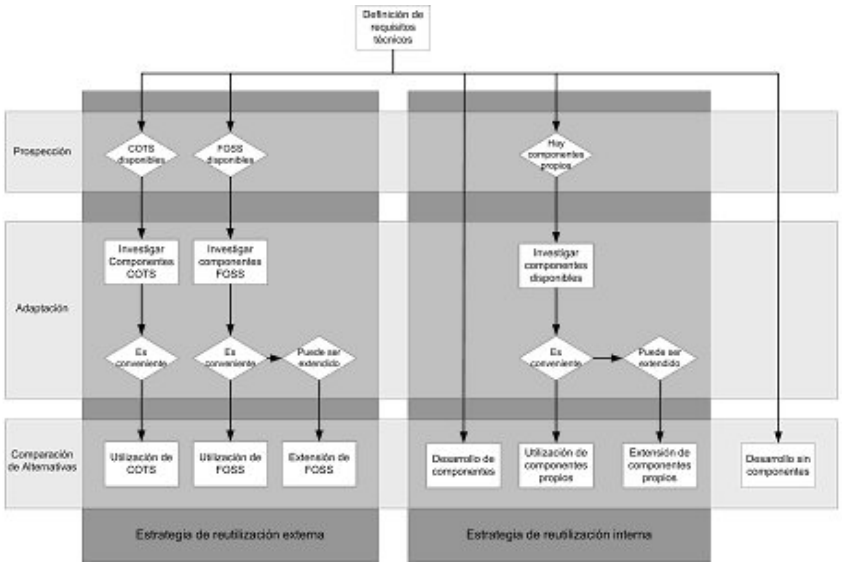

Figura 1. Modelo de evaluación de componentes reusables.

caja negra (*black-box*), la modificación de los componentes antes de su utilización se denomina reutilización de caja blanca (*white-box*) [22].

Cuando se trata de software FOSS es muy importante prestar atención a las siguientes cuestiones [28]: futura actualización funcional, compatibilidad con estándares abiertos, adaptabilidad y extensibilidad, y fiabilidad. Para responder a estas cuestiones es necesario profundizar en el diseño y la arquitectura, que con suerte estarán descritos en un manual de usuario.

Desafortunadamente, muchos proyectos FOSS sufren de mala documentación, lo cual hace muy difícil el aprendizaje o la extensión de los componentes, lo que dificulta su integración, y en especial la extensión o adaptación. Por lo tanto la existencia de buena documentación, ejemplos, artículos o grupos de noticias es indispensable para una integración rápida y rentable. Existen varios métodos de búsqueda sistemática de componentes FOSS (ejemplo, <http://www.amospro ject.org>), así como para su evaluación [9].

Para asegurar la calidad del software reutilizable, especialmente en proyectos a largo plazo como la arquitectura de una línea de productos, se ha de inspeccionar el código fuente. Esto puede hacerse utilizando indicadores de calidad como métricas orientadas a objeto o comentarios [4][5][10][18].

Todos los candidatos potenciales que hayan superado las dos primeras etapas se comparan en la tercera fase. Estas dos primeras fases pueden requerir un gran esfuerzo, no obstante, la información recogida es la base para el análisis final económico y de gestión.

2.2. Software específico

Imaginemos un vendedor de sistemas que implementa una solución altamente adaptada para un cliente, por ejemplo un sistema de monitorización para un escenario IT (*Information Technology*) diversificado.

La especificación de requisitos es la base de la oferta y cualquier funcionalidad adicional solicitada por el cliente conlleva una solicitud de cambio que tiene que ser negociada de nuevo cada vez.

La reducción del esfuerzo de desarrollo y el resultante decremento de coste es el aspecto de gestión más importante a ser considerado. El tiempo de salida al mercado puede ser otra restricción que debe ser tenida siempre en cuenta. Por lo tanto la utilización de componentes existentes, tanto internos como externos, es absolutamente aconsejable debido a que su reutilización es más barata que implementar la misma funcionalidad una y otra vez y además el calendario de desarrollo puede acortarse. El desarrollo interno de componentes reutilizables no es rentable ya que la funcionalidad deseada está ajustada a la medida de un caso especifico y su reutilización en proyectos futuros es muy improbable.

Adicionalmente la integración de software COTS y FOSS ofrece la capacidad de producir sistemas con tecnologías de elevada sofisticación, incluso cuando la empresa carece del conocimiento necesario. En particular se ha demostrado que la implantación correcta de los mecanismos de seguridad es tan difícil que los desarrolladores deben ser liberados del riesgo de producir defectos serios mediante la utilización de COTS apropiados. Empero, la integración de COTS viene siempre acompañada de los correspondientes costes de licencia.

El software FOSS provee las mismas ventajas que el software COTS pero no hay que abonar pagos por licencia. El impacto de la licencia de un componente FOSS no es importante en el caso de software específico. Incluso si la aplicación es una versión extendida de un componente FOSS que está cubierto por una licencia *copyleft* (**N. del T.**: término utilizado en contraposición a *copyright* y que define un modo de licencia de software libre que implica que cualquier desarrollo derivado debe ser también libre y ser distribuido conjuntamente con el código fuente y sin ninguna restricción) su impacto es poco significativo. Una licencia *copyleft* no impide al vendedor comercializar un producto derivado, pero el comprador tiene el derecho a redistribuir el programa y a obtener su código fuente. En el caso del desarrollo de aplicaciones específicas es habito común dar el código fuente al cliente de todas maneras. Y el cliente no obtiene ningún beneficio por entregar su solución personalizada a cualquier otro. No obstante debemos hacer notar que las diferentes licencias FOSS tienen implicaciones en el desarrollo de software propietario.

En resumen, en el caso de software específico el coste conjunto de los diferentes tipos de componentes debe ser estimado y se debe escoger la alternativa más barata.

2.3. Software propietario

Imaginemos ahora una compañía de software que ha acumulado un vasto conocimiento en un campo especifico (por ejemplo, en gestión documental), y desea comercializar una aplicación de sobremesa que ayude a organizar el papeleo diario. La división de marketing de la compañía ha identificado otros requerimientos adicionales (por ejemplo, otras tareas del trabajo de oficina) que deben ser integrados en la aplicación. La compañía desea extender su participación en el mercado y vende licencias que son validas para solo un puesto de trabajo.

Esta situación es completamente diferente a desarrollar software individual a medida debido a que el horizonte temporal es mucho más amplio y el modelo de negocio está basado en la venta de licencias.

Las tareas del ingeniero de software son las mismas de antes: identificar la funcionalidad requerida, señalar los componentes potenciales e investigar a los candidatos finalistas. Pero adicionalmente, es muy importante identificar y especificar los requisitos futuros de forma muy minuciosa, especialmente si los componentes son una parte importante del diseño, como es el caso del núcleo de la arquitectura en el desarrollo de una línea de productos. Para estar abierto a futuros cambios los componentes deben ser muy adaptables, modulares y confiables.

El desarrollo de componentes internos para su reutilización es la alternativa más flexible en lo que se refiere a la reusabilidad y funcionalidad. Como los componentes pueden ser diseñados desde el inicio, su funcionalidad se adapta para ajustarse a las necesidades de la compañía. La mayor desventaja de establecer una estrategia de reutilización interna es la gran inversión inicial para el desarrollo de componentes reutilizables. A causa de los elevados requisitos en cuanto a modularidad, calidad de software y documentación, el coste del desarrollo de software reutilizable puede ser significativamente mayor que el desarrollo de la misma funcionalidad para una aplicación única. Un estudio indica que el punto de equilibrio se alcanza como muy pronto después de reutilizar los componentes tres veces [1]. Para una estrategia de reutilización exitosa es obligatorio disponer de un repositorio de componentes y éste debe de estar integrado en el proceso de desarrollo. Esto puede conducir a mayores costes de organización.

Escoger COTS no requiere una inversión inicial aparte de adquirir la licencia. No obstante esta opción tiene varias desventajas que superan esta ventaja.

La flexibilidad está limitada ya que los componentes no pueden adaptarse para encajar con necesidades especiales. La funcionalidad adicional debe realizarse a través de capas envolventes. El desarrollo de este código de enlace puede necesitar hasta tres veces más tiempo que el desarrollo local del propio software [24][27].

Debido a que los COTS son cajas negras aparecen dos riesgos importantes: riesgos de comportamiento interior indeterminado y la interacción con el entorno. El primero puede conducir a resultados incorrectos o incluso a que la aplicación completa no sea fiable. Para prevenir esto se debe realizar un número excesivo de pruebas, lo que eleva los costes de integración. El segundo presenta varios riesgos de seguridad tal como el acceso a recursos restringidos, el acceso a recursos sin autorización o el abuso de los privilegios autorizados [30][17].

Otro gran problema es el mantenimiento de COTS debido a que los clientes dependen de la actividad del vendedor [2]. La frecuencia y número de actualizaciones pueden ser inciertos, lo cual hace muy difícil para el cliente mantener la aplicación. Por consiguiente es muy importante tener una visión de conjunto de los ciclos de desarrollo y mantenimiento de los COTS de manera que puedan sincronizarse con el proceso real de desarrollo. Por estas razones los costes adicionales necesarios para mantener los sistemas basados en COTS pueden igualar e incluso exceder los del software desarrollado a medida [24].

La adopción de componentes FOSS existentes puede ser la base para la estrategia de reutilización de una compañía, porque FOSS combina las ventajas de una reutilización interna y de la estrategia COTS. No requiere inversión inicial, pueden utilizase las funcionalidades existentes y el código fuente está disponible.

La libre disponibilidad del código fuente es lo que hace a esta estrategia tan poderosa. Por un lado provee de la flexibilidad necesaria para extender o adaptar los componentes FOSS y por otro lado posibilita la inspección y depuración, lo cual reduce el riesgo de comportamientos inesperados. A pesar del coste de aprendizaje y adaptación no se requiere una inversión inicial para integrar los componentes en el proceso de ingeniería. Adicionalmente la compañía puede recortar sus costes de mantenimiento ya que éste lo realiza la comunidad. Pero estos pasos deben considerarse cuidadosamente ya que el conocimiento puede estar disponible para cualquiera (ver más abajo).

El principal inconveniente de los componentes COTS, el comportamiento no especificado, también es aplicable a los componentes FOSS. Pero como el código fuente está disponible, se pueden arreglar inmediatamente los errores y defectos. E incluso si la comunidad ha dejado de dar soporte al código, los usuarios pueden extenderlo y mantenerlo con sus propios medios.

Del proceso de desarrollo FOSS surgen otras ventajas caracterizadas por el principio de revisión por iguales (*peer-review*) [23]. Con este método de revisión por iguales el programa lo revisan como mínimo dos expertos (probablemente más en un comunidad FOSS activa). Este juicio critico ayuda a mejorar el diseño, funcionalidad y calidad del programa. Además, la exposición del código fuente es necesaria para las aplicaciones de seguridad, debido a que el usuario del software puede verificar la funcionalidad y fiabilidad del programa, por ejemplo las transmisiones cifradas de datos.

Pero la disponibilidad del código fuente no implica de ninguna manera buenos programas. Un programa sólo puede ser tan bueno como sus programadores. Por lo tanto es necesario obtener una visión clara de las actividades de la comunidad en lo que se refiere a número de desarrolladores involucrados, madurez del proyecto (alfa, beta, estable) y frecuencia de actualizaciones [14][15]. Sólo proyectos con varios desarrolladores, elevada actividad de programación, y planes bien definidos son merecedores de ser utilizados en las lineas de producto de una compañía [21]. Proyectos que sólo tienen un puñado de contribuidores o en el que el trabajo está muy concentrado pueden morir o permanecer en un estado inestable. En este caso los beneficios del software FOSS desaparecen dado que no existe una comunidad que mantenga y haga evolucionar el producto. Debe evitarse la incorporación de versiones temprana (*snapshot release*, beta, etc.) ya que esos componentes pueden ser defectuosos o los siguientes cambios en el diseño o la funcionalidad pueden requerir un esfuerzo adicional de integración.

Otros aspectos importantes a considerar son las implicaciones legales. La utilización en aplicaciones comerciales suscita algunas preguntas que deben considerarse. Richard Stallman fue el primero que introdujo el termino "software libre" [26]. Desde su punto de vista, el software FOSS, y en especial los programas que lo adaptan, debe distribuirse libremente. Esta idea conduce a la bien conocida "licencia publica general GNU" (GNU *General Public License*, GPL), la cual implica que un programa que utiliza software GPL también está cubierto por esta licencia. Tal como se ha mencionado más arriba, vender una aplicación cubierta por la licencia GPL no está prohibido, pero la compañía vendedora no posee el derecho exclusivo de copia (*copyright*) que impediría la redistribución del software por parte del usuario. Y todavía más, el código fuente del trabajo derivado debe de estar disponible para cualquiera que use el software. Éste puede ser un criterio decisivo para no utilizar productos GPL en una aplicación propietaria. La licencia LGPL (*Library/Lesser GNU public license*) es menos restrictiva y permite distribuir software cerrado que utilice productos LGPL, pero cualquier cambio en el componente FOSS debe ser cedido a la comunidad. La licencia menos restrictiva es la licencia Berkeley BSD (y sus derivadas), que permite utilizar y modificar los componentes FOSS pero el código fuente de las modificaciones no tiene que ser publicado [6][[9][8].

En el caso de la aplicación de gestión documental mencionada más arriba, la utilización de una licencia *copyleft* puede conducir a una disminución de la participación en el mercado, debido a que el software puede ser distribuido libremente y, aun peor, el conocimiento interno de la gestión documental debe estar disponible para cualquiera que use el programa. La pregunta de cuándo un programa que utiliza componentes GPL es un trabajo derivado es muy difícil de responder y es actualmente objeto de discusión [29][12][25].

Por lo tanto debe evitarse la utilización de componentes GPL. La licencia LGPL es menos restrictiva pero la obligación de que

> Apenas existen conocimientos sobre la integración de software libre o de código abierto en el proceso de ingeniería de software

los cambios se hagan públicos a la comunidad FOSS debe ser bien evaluada. Incluso pequeños cambios pueden revelar información critica sobre el negocio central de la compañía. Así pues es preferible la adopción de FOSS estilo BSD ya que ésta es la licencia más flexible.

3. Conclusión y trabajo posterior

Hemos investigado y comparado tres estrategias de reutilización diferentes e identificado los costes globales por la adopción de componentes reutilizables, especialmente para el desarrollo de software a medida muy adaptado al cliente. Se han de considerar aspectos adicionales en el caso del software propietario. La integración en líneas de producto requiere componentes muy flexibles, modulares y fiables. Por lo tanto los componentes COTS no son apropiados debido a los inconvenientes de la reutilización de caja negra. Utilizar componentes FOSS como reutilización de caja blanca es una buena alternativa para una estrategia de reutilización interna ya que promete ahorros tanto para el mantenimiento como para la implementación de características futuras. No obstante deben considerarse minuciosamente los impactos de las diferentes licencias FOSS.

Desgraciadamente no sabemos de ningún estudio empírico que haya comparado los costes globales de las tres estrategias de reutilización entre sí. Las razones para este vacío de conocimiento es obvio. La funcionalidad de las aplicaciones investigadas debería, idealmente, ser idéntica y además se tendría que medir el esfuerzo de desarrollo. Es muy difícil encontrar suficientes compañías, y productos comparables, que estén dispuestas a publicar métricas internas que puedan ser utilizadas para determinar su productividad o métodos de desarrollo.

Otra posibilidad para recoger información es la vía experimental. Un experimento se puede utilizar para comparar cuando menos dos escenarios diferentes supervisados por un científico independiente. El carácter de la reutilización de software implica una actividad a largo plazo, lo que hace muy difícil la realización de ese tipo de prueba en un escenario académico (tres grupos de estudiantes desarrollando la misma aplicación y utilizando diferentes estrategias de reutilización). Es más, un experimento de este tipo produciría un indicador en vez de un resultado confirmado ya que el universo es demasiado pequeño.

Sugerimos comparar las diferentes estrategias de reutilización con modelos existentes de estimación de esfuerzo, por ejemplo, el modelo COCOMO II [3]. Este modelo está basado en la evaluación de sobre un centenar de proyectos software y contiene un modulo para reutilización de software. Puede integrarse el impacto de varios indicadores de calidad, por ejemplo, documentación y comprensibilidad. Hay alguna investigación en curso orientada a la evaluación económica de software FOSS utilizando modelos estimativos[13][20].

Referencias

[1] T. J. Biggerstaff. Is technology a second order term in reuse's success equation? In *Proceedings of Third International Conference on Software Reuse*, 1994.

[2] Barry Boehm, Chris Abts. COTS integration: Plug and pray. *IEEE Computer*, 32(1):135–138, 1999.

[3] Barry W. Boehm, Chris Abts, A. Windsor Brown, Sunita Chulani, Bradford K. Clark, Ellis Horowitz, Ray Madachy, Donald Reifer, Bert Steece. *Software Cost Estimation with CoCoMo II*. Prentice Hall PTR, Upper Saddle River, 1 edition, 2000.

[4] Samuel Daniel Conte, H.E. Dunsmore, V.Y. Shen. *Software Engineering Metrics and Models*. The Benjamin/Cummings Publishing Company, Menlo Park, CA, 1986.

[5] Norman E. Fenton. *Software Metrics - A Rigorous Approach*. Chapman & Hall, London, 1991.

[6] Martin Fink. *The Business and Economics of Linux and Open Source*. Prentice Hall, Upper Saddle River, 2002.

[7] Brian Fitzgerald, Tony Kenny. Developing an information systems infrastructure with open source. *IEEE Software*, 21(1):50–55, 2004.

[8] Christina Gacek, Budi Arief. The many meanings of open source. *IEEE Software*, 21(1):34–40, 2004.

[9] Bernard Golden. *Succeeding with Open Source*. Addison-Wesley, Boston, 2005.

[10] B. Henderson-Seller. *Object-Oriented Metrics: Measures of Complexity*. Prentice Hall, Upper Saddle River, NJ, 1996.

[11] Emmanuel Henry, Benoit Faller. Large-scale industrial reuse to reduce cost and cycle time. *IEEE Software*, 12(5):47–53, 1995.

[12] Till Jaeger, Carsten Schulz. Gutachten zu ausgewählten rechtlichen aspekten der open source software. Technical report, JBB, 2005. <http://opensource.c-lab.de/files/portaldownload/Rechtsgutachten-NOW.pdf>.

[13] S. Koch, C. Neumann. Evaluierung und aufwandsschätzung bei der integration von open source software-komponenten. In *Informatik* 2005 -Beiträge der 35. Jahrestagung der Gesellschaft für Informatik e.V. (GI), Lecture Notes in Informatics (LNI) , Gesellschaft für Informatik (GI), 2005. (De pronta aparición.)

[14] Stefan Koch. Profiling an open source project ecology and its programmers. *Electronic Markets*, 14(2):77–88, 2004.

[15] Stefan Koch, Georg Schneider. Effort, cooperation and coordination in an open source software project: Gnome. *Information Systems Journal*, 12(1):27–42, 2002.

[16] Wayne C. Lim. Effects of reuse on quality, productivity, and economics. *IEEE Software*, 11(5):23–30, September 1994.

[17] Ulf Lindvist, Erland Jonsson. A map of security risks associated with using COTS. *IEEE Computer,* 31(6):60–66, June 1998.

[18] M. Lorenz, J. Kidd. *Object Oriented Metrics*. Prentice Hall, Upper Saddle River, N.J., 1995.

[19] T.R. Madanmohan, Rahul De. Open source reuse in commercial firms. *IEEE Software*, 21(1):62–69, 2004.

[20] C. Neumann. Bewertung von open source frameworks als ansatz zur wiederverwendung. In *Informatik* 2005 - Beiträge der 35. Jahrestagung der Gesellschaft für Informatik e.V. (GI), Lecture Notes in Informatics (LNI) , Gesellschaft für Informatik (GI), 2005. (De pronta aparición.)

[21] Jeffrey S. Norris. Mission-critical development with open source software: Lessons learned. *IEEE Software*, 21(1):42–49, 2004.

[22] Ruben Prieto-Diaz. Status report: software reusability. *IEEE Software*, 10(3):61–66, May 1993.

23] Eric S. Raymond. *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. O'Reilly and Associates, Sebastopol, California, 1999.

[24] Donald J. Reifer, Victor R. Basili, Barry W. Boehm, Betsy Clark. Eight lessons learned during COTS-based systems maintenance. *IEEE Software*, 20(5):94–96, 2003.

[25] Gerald Spindler, Christian Arlt. *Rechtsfragen bei Open Source*. Schmidt, Köln, 2004.

[26] Richard Stallman. *Free Software, Free Society: selected essays of Richard M. Stallman*. GNU Press, Boston, 2002.

[27] Jeffrey M. Voas. The challenge of using cots software in component based development. *IEEE Computer*, 31(6):44–45, June 1998.

[28] Huaiqing Wang, Chen Wang. Open source software adoption: A status report. *IEEE Software*, 18(2):90–95, March/April 2001.

[29] Ulrich Wuermerling, Thies Deike. Open source software: Eine juristische risikoanalyse. *Computer und Recht*, (2):87–92, 2003.

[30] Qun Zhong and Nigel Edwards. Security control for COTS components. *IEEE Computer*, 31(6):67–73, June 1998.