

Ingeniería del Software Libre

Manuel Dávila Sguerra

“El software libre es el mundo de los programadores que saben trabajar de manera comunitaria en proyectos que nacen de sus intereses personales, académicos e investigativos. Las motivaciones que los mueven son diferentes a las tradicionales y su actividad ha creado una Ingeniería de software que crea estándares”

No es posible hablar de este tema sin citar el Libro “Desarrollo Catedral y Bazar” de Erik Raymond [1] que comienza diciendo: “Linux es subversivo. ¿Quién hubiera pensado hace apenas cinco años que un sistema operativo de talla mundial surgiría, como por arte de magia, gracias a la actividad *hacker* desplegada en ratos libres por varios miles de programadores diseminados en todo el planeta, conectados solamente por los tenues hilos de la Internet?”.

La mención de los cinco años determina una relación de tiempo con el momento en que se escribió este tratado.

No deja de tener cierto misterio ese mundo de personas anónimas, que trabajando de manera cooperativa han

logrado desarrollar estados del arte que hoy en día son líderes en muchos de los mercados en los que se utilizan y en el que Linus Torvalds, el desarrollador del núcleo de Linux, vino a descubrir la existencia de una Ingeniería de *software* creada en forma comunitaria.

El presente artículo intenta resumir el modelo que se ha venido construyendo para hacer *software* tomando como fondo las premisas de Erik Raymond, complementado con algunos comentarios propios.

Las premisas de Raymond

Basado en su propia experiencia en el desarrollo de *fetchmail*, Erik Raymond define una serie de premisas que conforman los mandamientos de

aquellos que se embarcan en un desarrollo de *software* abierto y explica los momentos que se viven, desde las motivaciones para iniciar un proyecto, las acciones que mantienen el liderazgo del gestor, las relaciones con los terceros, las sinergias que se crean alrededor del grupo, y el panorama de las herramientas preferidas por los desarrolladores que, a pesar de la aparente anarquía que los identifica, han venido creando estándares muy bien definidos.

Los intereses del programador

Si bien un desarrollo de tipo propietario nace por las necesidades del cliente, el *software* abierto surge de las necesidades personales del mismo programador, quien ha identificado un problema que a él le interesa resolver en forma prioritaria. Esta premisa conlleva una entrega sin precedentes para atacar el problema y cambia las prioridades que se dan en los desarrollos tradicionales, en los que predominan los intereses económicos.

Reutilización de código propio y de terceros

Esta famosa frase sobre la importancia de la reutilización de código cobra un valor superlativo en el *software* libre pues va más allá del desarrollo de las propias librerías, con el claro propósito de aumentar la productividad y de facilitar el mantenimiento futuro.

Trasciende a la reutilización del código de “otros programadores”. Si bien a los *hackers* se les identifica por tener una vanidad excesiva es curioso que no sientan ninguna pena de reutilizar el código de otros y considerar que la búsqueda del camino más fácil, como lo pregona Torvals, es un síntoma de mayor inteligencia.

Problemas de conciencia

Quienes programamos cotidianamente sabemos que la forma de plantear la programación de un algoritmo es sensible de tomar caminos tormentosos. En estos casos se aconseja tener en cuenta la opción extrema de desechar el trabajo, botarlo al cesto de la basura y recomenzarlo. De cierta manera es un momento de reconciliación con uno mismo, tan beneficioso para el espíritu y para el proyecto como cuando uno decide perdonar al enemigo. Casi siempre el nuevo camino está libre de cruces.

Todo debe ser interesante

En un trabajo en el cual el nivel de abstracción es tan alto como en la programación de computadores, no es de ninguna manera sano para la mente del programador tener como propósito único terminar lo que está haciendo. Así como un escritor solo debe estar interesado en el párrafo que en un instante está escribiendo, el programador debe estarlo en el módulo, función,

objeto o programa que esté programando, momento en el que esa ocupación debe ser la única importante.

Las dificultades inherentes a esa tarea no deben verse como un problema para resolver, sino como un momento de recreación. Es parte de la diversión intelectual y de sentir la sensación de libertad que lo ha llevado a desarrollar este trabajo, sensación que según el filósofo Jiddu Krishnamuri es la que está más cerca de la felicidad [2]

Del interés por el proyecto al interés porque se lleve a cabo

Uno de los elementos más importantes en los grupos de desarrollo del *software* libre es el liderazgo que ejerce el gestor de un proyecto. Seguir al líder, ser su coautor, es parte de las motivaciones de los colaboradores. Pero cuando el líder pierde el interés este se transmitirá de manera invisible y contagiosa a los colaboradores y es un momento en el cual debe ser capaz de heredar el proyecto a un sucesor considerado competente. No hacerlo es acabar con el proyecto.

La transformación de los clientes en colaboradores

En el mundo propietario el perfil del cliente es lo más cercano al de un jefe. Y hay jefes buenos y jefes malos. La comunicación entre el programador y el cliente tiende a tener momentos

de presión que pueden deteriorar las relaciones y hacerle daño al proyecto. En el software libre, el cliente se llama colaborador. Y es un socio permanente. Un coautor potencial, un posible sucesor. No es él quien reclama por los errores del software. Es quien le ayuda a detectarlos y quien de manera gratuita conforma su equipo de pruebas. El error se considera como parte de la naturaleza de los proyectos de software, su existencia tiene un contexto de valor humano y siempre estará presente. Su aparición no golpea la vanidad del programador. Más bien da a los colaboradores la oportunidad de protagonizar una solución y de ser aceptados dignamente como coautores apreciados. Esta integración se logra de verdad solo cuando se reconocen y se escuchan las opiniones de los colaboradores.

El sociólogo Gerald Weimberg autor del libro *la Psicología del programador* [3] se permite hacer una discusión sobre la “programación sin ego” y concluye que cuando no hay propiedad del código y se estimula la búsqueda de los errores, el avance es más rápido.

Proyectos Beta permanentes

El desarrollo de *software* no debe verse -porque no lo es-, como un trabajo terminal. Es un proceso permanente cuyo estado preferido es el beta. Y un estado beta permanente tiene que

estar liberando versiones de manera frecuente. No hacerlo es transmitir desinterés por parte del líder o abandono por parte de los coautores. Este proceso se da cuando se escucha de verdad a los clientes, en este caso los colaboradores que cumplen la labor de beta-testers.

Enséñame tu código y te diré quién eres

En el *software* propietario el código fuente no está expuesto al público en general, por lo que no es común sentir ansiedad por la exposición de esta faceta de la personalidad. El mismo Weimberg en su libro sobre la *Psicología del programador*, ya citado, demuestra cómo el mismo problema es resuelto de diversas formas, según la persona que lo haya escrito y explica cómo en el código hay una expresión de la personalidad de quien lo haya hecho. En el *software* libre se debe estar sujeto a esta exposición de sí mismo ante un público casi siempre muy experto.

En la visita a Colombia de Rasmus Lerdorf el creador del lenguaje *php* tuve la oportunidad de preguntarle cómo era ese proceso en el cual un colaborador es o no aceptado para trabajar en un grupo de desarrollo abierto, y él me comentaba que era un proceso netamente social, en donde la aceptación o el rechazo era producido por la misma comunidad. “Enséñame

tu código y te diré quién eres”. Pero tal vez más enfático sería decir si seguimos las normas de Raymond: “Dime cómo son las estructuras inteligentes de tus datos y te diré quién eres”.

La experiencia es un archivo de errores bien organizado

Esta frase se refiere a la intervención que tienen los errores cometidos en el proceso de aprendizaje. Si uno está en un proceso de investigación aplicada, por ejemplo para la instalación y uso de una plataforma tecnológica, si corre con suerte no le funciona. Si está de malas le funciona en el primer intento. Esta aparente contradicción se explica porque el esfuerzo mental en la resolución de los problemas es el que hace que el conocimiento tome asiento en el cerebro. Sin embargo, afirmarlo tiene sus peligros, como le sucedió a Maturana el entrenador de nuestro equipo de fútbol Colombiano cuando dijo: “Perder es ganar un poco”.

Hay casos en que la búsqueda de la perfección hace daño. Hay quienes dicen que lo perfecto es enemigo de lo bueno. Pero más filosóficamente lo dijo Antoine De Saint- Exupéry [4], en el *Principito*, como lo comenta Raymond: “La perfección se alcanza no cuando ya no hay nada que agregar, sino cuando ya no hay algo que quitar”.

En tal sentido, existe una premisa en la Ingeniería del *software* libre que dice literalmente: “en los diseños de los programas se debe alterar lo menos posible el flujo de los datos y la información incluida no se debe eliminar, a menos que los receptores lo obliguen a hacerlo”.

Hay más premisas que definen el comportamiento de los colaboradores en el desarrollo de un *software* libre, las cuales aconsejan establecer relaciones no basadas en las luchas de poder; evitar acciones coercitivas; actuar bajo las normas del entendimiento y no del autoritarismo; identificar voluntades convergentes; entender la existencia de motivaciones que van más allá de lo económico, como la búsqueda de una satisfacción personal que supere el deber; y, el respeto a quienes actúan para satisfacer su ego basados en resultados exitosos y diferentes.

Lenguajes, herramientas y plataformas

Una variable que ha determinado la escogencia de los lenguajes, herramientas y plataformas es que deben ser asequibles, libres, de fácil aprendizaje con esfuerzos reducidos. Esto fue creando el famoso término LAMP que es un acrónimo de **L**inux, **A**pache, **M**ysql y **P**hp, aunque también debemos incluir a Perl y Python sin dejar de mencionar a Java, C y C++,

que tienen el más alto índice de uso en Internet.

Hay un sitio llamado <http://www.tiobe.com/tpci.htm> en donde se miden, mes a mes, las tendencias de los lenguajes, cuya consulta resulta interesante.

La escogencia de las herramientas es conservadora, muy orientada hacia modelos distribuidos y al manejo de comunicaciones interpersonales y de grupo; son preferentemente asincrónicas y respetan lugar y tiempo como está ocurriendo en el modelo de la educación virtual, en donde a pesar de que las personas trabajan en grupo, se respeta el desempeño individual, siempre y cuando no retrase a nadie, estimule el autoestudio y facilite la liberación de los resultados.

Para la construcción de los programas se usa una herramienta llamada **make** que dados los códigos fuente especifica los árboles de dependencias entre *software* archivos: *x.o* depende de *x.c* y de *x.h* por ejemplo, de tal manera que cuando se modifica un código fuente y luego se ejecuta **make**, se recompilarán solo los módulos afectados y se volverá a montar el objeto final. Otra alternativa menos usada es **ant** cuya popularidad está impulsada por el éxito de java.

Se usan herramientas dedicadas a ayudar a que los programas sean portables, es el caso de **Autoconf** para producir

shell scripts que configuran paquetes desde el código fuente para adaptarlos a sistemas tipo Unix, produciendo scripts independientes. Usa el programa **m4** para transformar archivos del tipo “configure.ac”, como es usual hacerlo al configurar **sendmail**, el agente de correo.

Automake es otra herramienta de programación para producir archivos portables usados por **make**.

Cuando se baja un paquete fuente, si está escrito en C, empaquetado con **tar**, comprimido con **gzip**, hecho portátil con **autoconf** u otras herramientas asociadas, y construible e instalable con **make**, la instalación se llevará a cabo por medio de un proceso como al siguiente:

Compresión: tar xzvf
paquete-1.3.5.tar.gz

Entrar al directorio descomprimido:
cd paquete-1.3.5

Configurar: ./configure

Compilar: make

Instalar: make install

Mecanismos de colaboración

La historia de los sistemas de colaboración se inicia en los años 70 con **uucp**, el protocolo de transferencia de archivos Unix; en 1979 aparece el primer enlace USENET sobre UUCP para noticias y para foros temáticos

estructurados jerárquicamente; en 1981 aparece BITNET.

Hoy en día se utilizan listas de correos como **Mailman**, Foros Web o **weblogs** genéricos como SlashDot o BarraPunto, en los que se anuncia un nuevo software libre o se discuten noticias relacionadas, mecanismo de colaboración basado en Wikis, como Media wiki: <http://www.mediawiki.org/wiki/MediaWiki> o MoinMoin: <http://moinmoin.wikiwikiweb.de/>

Hay mecanismos de interacción con los que los desarrolladores conversan en tiempo real, no muy práctica por ser sincrónica IRC (*Internet Relay Chat*), que normalmente comunica gente por medio de canales temáticos y no es común que se utilicen herramientas multimedia (sonido, imagen) por disponibilidad y complejidad en su instalación y uso.

Gestión de fuentes

Desarrollar software en grupo sin un sistema de control de versiones es imposible pues se deben mantener bases de datos con la historia del desarrollo, mecanismos que posibiliten la recuperación de versiones anteriores, registro de los cambios, manejo de versiones catalogadas como estables y no estables, forma de determinar las diferencias entre versiones y trabajo concurrente sin interferir en el código de los demás componentes del grupo.

La herramienta o mejor la plataforma más utilizada es el **cvs** que fue diseñado por los años 80.

Reconociendo que el trabajo sobre todo el del administrador es arduo y delicado de llevar toda vez que debe mantener el repositorio, dar proyectos de alta, permisos a los desarrolladores, definir versiones, crear ramas del desarrollo, configurar sistemas de mensajes por correo de manera automática y muchas otras tareas adicionales.

Sin embargo, existen otros sistemas libres que solucionan varios de estos problemas. Podemos destacar el posible sucesor de **cvs** llamado **subver-**

sión, del cual uno de sus autores Karl Fogel [5], escribió un libro en donde referencia su experiencia de manera interesante siguiendo las normas de Raymond. Un gestor de **cvs** orientado a la web muy usado se llama **cvsweb**.

Documentación

En el mundo del *software* libre no se usan mucho los procesadores de texto de tipo gráfico, debido a que el rey de la documentación es el texto y hay preferencia por herramientas que no consumen muchos recursos, de las cuales se dice que usan formatos transparentes.



Figura 1. Control de versiones cvsweb.

La documentación es tan extensa que elimina cualquier duda con respecto a la actitud de compartir los resultados de parte de los *hackers* participantes en los proyectos. Del proyecto GNU de Stallman nació el formato **texinfo** que permite leer los documentos desde el programa **info** y **emacs**, el editor de Stallman, así como del procesador de textos **TeX**, de Donald Knuth. El formato **texinfo** puede pasarse a **html** para los que quieren ver los documentos en un navegador.

Docbook que es una aplicación basada en **sgml** es un estándar para muchos de los proyectos con críticas basadas en las dificultades en el manejo de marcas que conlleva.

Últimamente se ha popularizado el uso de **wikis** invento de Ward Cunningham y liberado en 1995 que por su sencillez está tomando ventajas a otras formas de documentación colaborativa como se explica en:

<http://twistedmatrix.com/users/jh.twistd/moin/moin.cgi/WikiSandBox>

La gestión de errores es un tema de gran fortaleza en el desarrollo del *software* libre porque como ya dijimos el error es fuente de salud para los resultados de los proyectos. Es necesario almacenarlos en bases de datos y bajo el cuidado de un sistema de información que le de el valor esperado. Una herramienta muy importante desarro-

llada por Debian se llama **reportbug** que maneja un excelente control del tráfico de información relacionado con los errores y se puede ver un ejemplo de la base de datos en: <http://bugs.debian.org>.

Editores

Siguiendo la característica conservadora, para la escogencia de las herramientas no se acostumbra el uso de editores muy complejos y aún hay muchos que usan **vi** (hoy **vim**), **geditor** y, en algunos casos, **blue fish** con opciones que ayudan a escribir programas orientados a la web. Así mismo, se usa Eclipse, una herramienta más integral aunque se le critica lo “pesada” y compleja en su uso. Estas dos van más allá de los simples editores. Valga decir que el gran valor de los desarrollos está en el código fuente escrito y en los mecanismos para resolver los algoritmos, más que en las mismas herramientas.

Repositorios de aplicaciones

Cuando se habla del *software* libre casi siempre se hace referencia a unos pocos desarrollos debido a que son los que conforman la plataforma de base como Linux, Apache, Firefox, Squid, Iptables, Open Office, mysql, postgresql, Gimp, Asterisk y otros más. Pero ¿qué sucede con el resto de proyectos que residen en los repositorios como es el caso de Sourceforge?

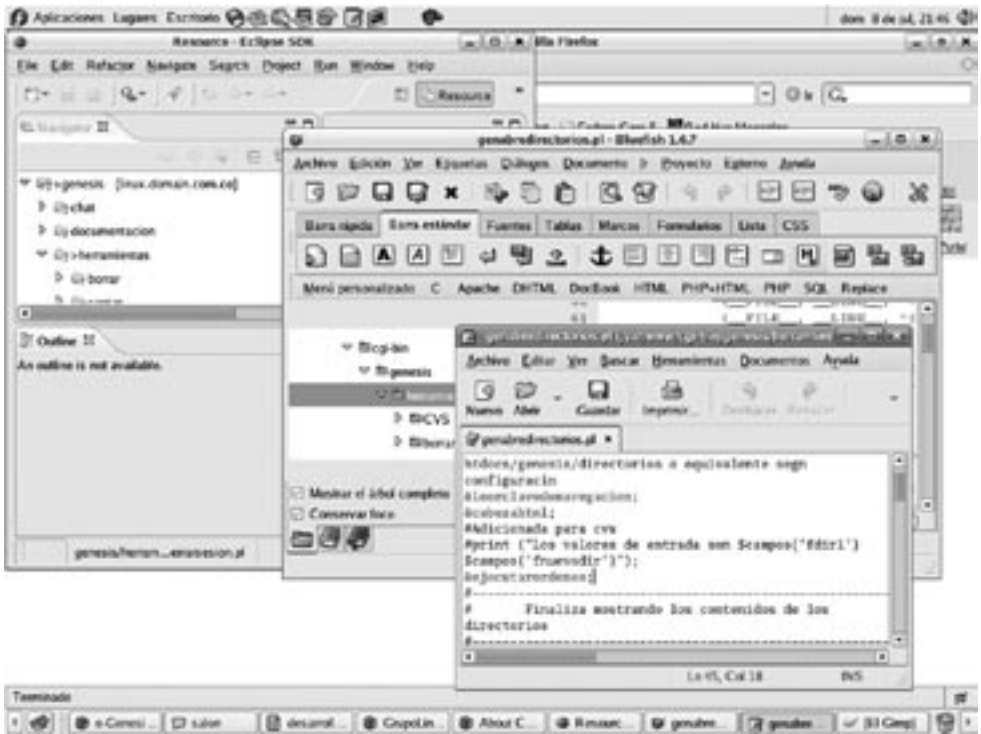


Figura 2. Editores y herramientas de desarrollo vi (vim), bluefish, Eclipse.

Sourceforge.net, es uno de los repositorios más importantes. Fue iniciado por la OSDN o Open Software Development Network, una subsidiaria de VA Software que a 18 de Agosto de 2007 tiene 155.681 (ciento cincuenta y cinco mil seis cientos ochenta y un) proyectos registrados que, comparados con los 60.000 (sesenta mil) de Junio de 2003 registra una tasa de crecimiento promedio del 27% anual.

Es en general un portal orientado al desarrollo de aplicaciones con todos los servicios relacionados: listas de correos, noticias, rastreadores para seguimiento de errores, peticiones de soporte y mejoras, manejo de priori-

dades, gestores de tareas, cvs para el control de versiones y para el acceso de los colaboradores, servicios para subir y bajar paquetes y manejar copias de seguridad. Usa como *software* para el seguimiento del repositorio cvs el ya mencionado *cvsweb* que es libre y de uso extensivo en los servidores utilizados para el desarrollo de aplicaciones.

Conclusiones

Los proyectos de *software* libre surgen por una necesidad del programador antes que una necesidad del cliente, lo que determina un tipo de comportamiento libre que se rige por normas

diferentes a las tradicionales usadas en los desarrollos propietarios.

El líder, que casi siempre es el gestor del proyecto, es un elemento esencial para evitar que se anarquice el manejo de dichos repositorios y se asegure la continuidad del proyecto; pero sobre todo, ejerce una presencia de autoridad benevolente, como la llama Linus Torvalds, que influenciará positiva o negativamente a los colaboradores.

Cabe destacar que las normas y estándares que se usan en el *software* libre por parte de los programadores han creado mecanismos de mejores prácticas aceptados ya por los desarrolladores y que en suma, han venido conformando la Ingeniería de *software* en este tipo de desarrollo.

El gran crecimiento de aplicaciones bajo licencias abiertas dio pie a la aparición de grandes repositorios en donde se publican los proyectos que están disponibles para quienes los quieran usar, creándose así una modalidad para la administración de portales orientados al desarrollo de *software*,

en los cuales se ven plasmados los estándares que las comunidades de programadores han venido construyendo.

Este modelo de trabajo tiende a ser emulado en muchas empresas que desarrollan *software* propietario, porque sus resultados evidencian que existe otra forma de producir *software* u otra manera para interpretar las características profesionales y personales de los programadores, con miras a una mayor productividad.

Referencias

[1] Raymond Erik, *The Cathedral & the Bazaar*, O'reilly 2000

[2] Jiddu Krishnamurti, *Total Freedom: The Essential Krishnamurti*, HarperOne; 1 edition Octubre 4,1996

[3] Gerald M. Weinberg, *The Psychology of Computer Programming*, Dorset House Publishing Company, Incorporated; Anl Sub edition September 1998

[4] Antoine De Saint-Exupéry, *El principito*, Paperback Septiembre 4 2001

[5] Kerl Fogel, *Producing Open Source Software, How tu run a Successful Free Software Project*, 2005, <http://www.producingoss.com/>

Manuel Dávila Sguerra. Ingeniero de Sistemas de la universidad de Los Andes;, Director del Departamento de Informática Redes y Electrónica, Uniminuto; Ex Director Uniminuto Virtual; gestor y ex presidente de la red Decanos y Directores de Ingeniería de sistemas y afines – REDIS-; Gerente Desarrollo Grupo Linux S.A.; Coordinador cursos tecnologías de punta – Acis-; co director salones sobre *software* libre - Acis 2001 y 2004-; columnista *Computer World* y *Blog eltiempo.com*. Autor de e-Genesis- *El Generador de sistemas y de los textos para el Curso virtual sobre Software Libre del Distrito de Bogotá*; mención especial Premio Colombiano de Informática 2006 y 1987; conferencista ley del *Software Libre* Congreso de la República; mención *Revista Semana* Mayo 31 de 2004 por la introducción de los primeros micros en 1980, Miembro Fundador *Indusoft* y *Acis*.