Valoración de productos de Software Libre y de Fuentes Abiertas

Borrador de trabajo. v0.2

Francisco Palm (francisco.palm@gmail.com), Mariángela Petrizzo (petrizzo@gmail.com)

Introducción

Existe una preocupación en el mundo empresarial-corporativo con respecto a la producción de software porque no les ha sido posible controlar su desarrollo del mismo modo en que se controla y se garantiza la calidad en la industria de manufactura, de servicios o de construcción. Está claro que paradigma neoliberal de la industrialización, por sí solo bastante cuestionable por la evidente depredación del medio ambiente y la enajenación de lo que hace humano al ser humano bajo el modelo económico imperante en la actualidad, busca con la mayor prioridad esquemas productivos de economía de escala, predecibles y ajustables a conveniencia.

En este contexto, a lo largo del tiempo el desarrollo de software ha mostrado ser una actividad con un enorme potencial lucrativo, pero poco susceptible a ser normado y controlado. Los ejecutivos, actualmente CEOs, del ámbito de las tecnologías de información prefieren asumir que los programadores son meros obreros de la informática[1], pero el problema es que incluso un desarrollo simple en tecnologías de la información requiere un nivel de conocimiento y de experticia relativamente alto, de modo que no es difícil encontrar situaciones en las cuales un programador “junior” tiene un mejor conocimiento del negocio que un alto directivo.

Esta lucha por poder y reconocimiento entre ejecutivos y desarrolladores se acentúa cuando encontramos programadores de gran talento que no tienen interés en abandonar su trabajo para ocupar exclusivamente cargos ejecutivos[2], y además superan con creces a los directivos en el ejercicio de sus propias funciones. Este fue un factor clave del boom del Silicon Valley, el surgimiento de empresas como Google y Amazon, y mantiene una enorme relación con el movimiento hacker de las décadas de los 60s, 70s y 80s, indudable origen de los movimientos de software libre y de fuentes abiertas. No es muy descabellado pensar que el mundo

---

[1] Vale la pena insistir que ya es un problema en cualquier ámbito reducir a un ser humano a la condición de obrero, porque quizás es apenas una forma de legitimar la asignación de bajos salarios.

[2] Equivalente a que un obrero aún teniendo la oportunidad de no hacerlo, insista en seguir pegando bloques.

corporativo del software privativo sigue sin dar muestras para entender, o incluso intentar entender al software libre.

El modelo de Capacidad y Madurez

Con la finalidad de evaluar desde una perspectiva burocrática las capacidades de desarrollar proyectos de software para contratistas del gobierno estadounidense, en particular la Fuerza Aérea de los EEUU, se elaboró a mediados de los 80s el Modelo de Capacidad y Madurez (Capability Maturity Model o CMM) como un instrumento para formalizar y optimizar procesos involucrados en el desarrollo de software.

El modelo se fundamenta en suponer que si una organización que desarrolla software mejora sus procesos siguiendo el modelo prescrito por CMMi, mejorará la calidad del software resultante. La evidencia ha demostrado que este supuesto con frecuencia no se cumple, ya que es necesario considerar a las personas más allá de los procesos, de allí surgieron el *Personal Software Process* (PSP) y el *Team Software Process* donde el PSP debe encajar.

Lo más llamativo es que el modelo CMMi se ha extendido sin tener ningún tipo de base teórica más allá de representar la opinión de expertos. En realidad, no existe una evidencia clara y contundente de su efectividad ya que la evidencia recabada generalmente por el propio SEI[3] destaca casos de éxito de las empresas sometidas a CMMi, pero en realidad no muestra comparación entre éxitos y fracasos, ni estudios comparativos con respecto a otras estrategias para mejorar la calidad del software. De hecho, es muy difícil encontrar opiniones positivas del CMMi que provengan de fuentes neutrales, Tampoco se considera el hecho que *en términos prácticos, muchas empresas simplemente están obligadas a seguirlo para tener la posibilidad de ser contratadas* . Esto supone que los vendedores de certificaciones CMMi adapten su discurso, pese a que la metodología CMM-CMMi ha cambiado muy poco en los últimos 10 años.

Hay alternativas a CMMi, que se muestran como versiones ligeras de éste aunque en el fondo parten de los mismos principios y exhiben las mismas carencias. Pese a ello, CMMi se ha mostrado especialmente inadecuado para equipos de desarrollo pequeños, sobre todo si lo comparamos con los esquemas de trabajo ágiles.

Pero además, son muchos los proyectos libres y no libres que han logrado ser muy exitosos sin necesidad de satisfacer CMMi, desde Microsoft hasta el desarrollo del kernel de Linux. De hecho, ninguno de los proyectos de Software Libre más significativos, con millones de usuarios satisfechos en el planeta, tales como el servidor web Apache, el manejador de bases de datos PostgreSQL o el navegador web Mozilla Firefox están certificados CMMi pues su interés no se centra en participar en licitaciones públicas. Sin embargo, es innegable la calidad de estos

[3] El Software Engineering Institute de la Carnegie Mellon University, cuya principal fuente de financiamiento es el Departamento de Defensa de los EEUU.

productos de software. Es evidente, entonces, que lo que impulsa la calidad de estos proyectos son los procesos abiertos a su comunidad y la búsqueda permanente de la excelencia.

Las principales críticas a las certificaciones estilo CMM y sus derivados son (Bach, 1994):

- No hacen énfasis en la calidad del producto sino en los objetivos de la organización, y desplaza los objetivos de la organización hacia el cumplimiento del propio CMMi.
- Se enfoca más en los procesos (administrativos) de soporte al desarrollo que en las prácticas del propio desarrollo, y en las personas que lo realizan.
- Se mide la conformidad con los procesos existentes, sin constatar que estos sean los procesos idóneos.
- Como están diseñados hacia lo que resulta predecible, distorsionan la capacidad de innovación.
- Es una receta "*entubada*" que no percibe ni se adapta a las condiciones particulares de cada cliente, ni siquiera considera su historial de éxitos o fracasos.

La metodologías de software concebidas como la base de un desarrollo de calidad no consideran el hecho de que el principal factor en la calidad del software son las destrezas de los equipos de desarrollo y de sus miembros, las cuales se refuerzan en el ámbito del software libre de fuentes abiertas en virtud del trabajo colaborativo. En consecuencia, en una primera instancia *no hace tanta falta metodologías y certificaciones como el fortalecer las destrezas individuales y colectivas de los equipos de desarrollo*. En especial, cuando ni siquiera se conocen cuáles son las destrezas de los actores involucrados, o de cuáles destrezas carecen.

En este sentido, es importante comprender que hay procesos organizacionales que, aplicados de forma correcta, pueden mejorar el trabajo de un grupo talentoso, pero un grupo de desarrolladores no se hace talentoso sólo mejorando sus procesos actuales.

Se recomienda entonces promover un modelo de desarrollo abierto y transparente que pueda convertirse en un proceso de aprendizaje y descubrimiento constante e incremental, articulado en sistemas abiertos y, por tanto, naturalmente entrópicos. Por ello, fomentar y desarrollar las destrezas básicas de los equipos de trabajo resulta una clave.

Modelos que permiten valorar el software libre

El índice de gobernanza abierta.

En este contexto, la gobernanza tiene que ver con los derechos de visibilidad, de influencia y de creación de derivados a partir de un software. Es una ampliación de la definición de las licencias, las cuales se restringen a una interpretación sobre derechos de uso, copia y distribución de la aplicación.

La gobernanza responde a preguntas clave para el proyecto que tienen que ver con la toma de decisiones sobre el curso de acción del proyecto, el nivel de transparencia de la toma de

decisiones, la posibilidad de cualquier persona de seguir los debates que suceden en la comunidad o de iniciar una derivación del proyecto matriz. En suma, muestra quien(es) tiene(n) influencia y control sobre el proyecto.

Incluye 13 indicadores ajustados a cuatro dimensiones de la gobernanza abierta de proyectos de software libre y que permiten medir su nivel de apertura. Lo que se sugiere es que, en el largo plazo, los proyectos con mayor índice de apertura serán los más exitosos.

Las dimensiones y los indicadores respectivos son los siguientes:

1. **Acceso:** Disponibilidad del código fuente actualizado, mecanismos de apoyo a desarrolladores, transparencia en la toma de decisiones y hoja de ruta con carácter público.
    a. ¿El código está disponible libremente a desarrolladores al mismo tiempo?
    b. ¿El código está disponible bajo una licencia aceptada por la OSI (Open Source Initiative)?
    c. ¿Están disponibles para todos los desarrolladores los recursos de apoyo? (por ejemplo, listas de correo, bases de datos de seguimiento de errores, repositorios de código fuente, documentación y herramientas de desarrollo)
    d. ¿La hoja de ruta del proyecto es pública?
    e. ¿Se evidencia transparencia en los mecanismos de decisión (minutas de reuniones y discusiones), de forma que sea posible comprender cómo y por qué se toman las decisiones del proyecto?
2. **Desarrollo**: capacidad de los desarrolladores de influir y direccionar al proyecto
    a. ¿Existe transparencia en el código de contribución y aceptación?¿Se evidencia actualización de progresos en la contribuciones realizadas? (via Bugzilla o similar)
    b. ¿Son las contribuciones transparentes de forma que sea posible evidenciar la procedencia de cada una de las contribuciones?
    c. ¿Los procedimientos para ser un colaborador del proyecto se encuentran documentados y se trata de un proceso equitativo? (es decir, ¿pueden todos los desarrolladores ser colaboradores potenciales?) Colaborador en este caso tiene que ver con los desarrolladores que hacen “commits” al proyecto, también llamados “mantenedores” o “revisores” del proyecto.
    d. ¿Es fácilmente identificable quiénes son los colaboradores del proyecto?
    e. ¿La licencia de contribución requiere una asignación o licencia de copyright o garantía de patente?
3. **Derivados**: capacidad de los desarrolladores de hacer "branches" o derivados del proyecto central en forma de aplicaciones, handsets o proyectos "spin-off" a partir del proyecto matriz.
    a. ¿Se usan las maras para controlar cómo y donde se utiliza el proyecto o plataforma a través de un proceso de aplicación de cumplimiento previo a la distribución?
    b. ¿Se restringen, en términos de aprobación, distribución o apertura, los canales

de mercadeo para derivativos de aplicaciones?

4. **Comunidad**: una estructura de comunidad que no discrimine entre los desarrolladores.
    a. ¿Qué forma tiene la estructura de la comunidad: plana, jerárquica? (por ejemplo, ¿hay derechos vinculados al estatus de la membresía?)

Queda claro que la sola licencia no hace abierto a un proyecto. Es un tema de gobernanza abierta el que debe ser asumido. Por ello, de los anteriores criterios, hay algunos que son comunes a proyectos exitosos de software libre:

1. Acceso oportuno al código fuente,
2. Herramientas sólidas de desarrollo,
3. Transparencia del proceso,
4. Accesibilidad a contribuir código, y
5. Accesibilidad a convertirse en un colaborador

Pero además, la igualdad y trato justo entre los desarrolladores (la llamada "meritocracia") es la norma y, por tanto, algo esperado por parte de los desarrolladores en lo que respecta a su participación en proyectos de software libre.

Certificación de procesos de software libre Open Logic

Reconociendo la importancia que reviste la certificación de software de cara a empresas, se diseñó una biblioteca de código libre OpenLogic Certified Library que indexa código que es validado y certificado en términos de garantizar su calidad y su integridad de cara a los usuarios finales (OpenLogic, 2008).
En esta biblioteca se busca asegurar que el software allí incluido:

- Provenga de una comunidad viable y respetable.
- Utilice un modelo de licenciamiento de código abierto bien entendido.
- Cumpla con estándares de calidad básicos.
- Incluye documentación útil.
- Provea un origen para descargar distribuciones de paquetes confiables.
- Cumpla con estándares de indemnificación, soporte y mantenimiento.

El proceso de certificación OpenLogic pasa por cuatro fases:

- Selección de los paquetes, que es básicamente un monitoreo de nuevas tendencias y paquetes propuestos por los usuarios de la biblioteca.
    - ¿El software está ampliamente aceptado?
    - ¿Satisface una necesidad de los clientes?
    - ¿Es una alternativa a otro software seleccionado?
    - ¿Ha sido nominado por los clientes o usuarios del sitio OpenLogic Exchange (OLEX)?
- Precalificación, verificación de requerimientos mínimos de licenciamiento y disponibilidad del código fuente.

    - ¿Utiliza una licencia de de código abierto conocida?
    - ¿Los términos de la licencia son muy restrictivos?
    - ¿La licencia es freeware, propietaria o libre/abierta?
    - ¿Existen restricciones para la (re)distribución del software?
    - ¿Se puede acceder al código fuente o únicamente a los binarios?
    - ¿Se dispone del código fuente de la versión estable actual?
- Aceptación inicial, basada en cinco evaluaciones: viabilidad, riesgos legales, riesgos de seguridad, soporte y distribuciones de las fuentes.
    - ¿El software recibe el apoyo de una empresa, fundación y/o comunidad?
    - ¿Qué tan grande es la comunidad que desarrolla, soporta, y utiliza el paquete?
    - ¿Cuántos desarrolladores y colaboradores aparecen listados en el sitio web?
    - ¿Se proveen los nombres y la información de contacto?[4]
    - ¿Cómo está organizada la comunidad para la entrega de nuevas versiones?
    - ¿El proyecto utiliza gestores del código fuente y sistemas de seguimiento de fallos?
    - ¿Desde cuándo existe el software? ¿Tiene un desarrollo activo?
    - ¿En qué etapa del ciclo de vida se encuentra actualmente el software?
    - ¿Con qué frecuencia se realizan grandes o pequeñas actualizaciones?
    - ¿Cuáles han sido las versiones liberadas y en qué fechas?
    - ¿Hay alternativas libres o privativas comparables?
    - ¿Tiene el software información estandarizada en directorios como OHLOH, Souceforge, Freshmeat, o Tigris?. ¿Está listado en recursos como la Wikipedia, FOSSBazar o SWiK?
- Certificación completa, basada en una revisión minuciosa de la licencia, disponibilidad de documentación
    - ¿El proyecto provee un wiki, un documento de preguntas frecuentes (FAQ), y/o una base de conocimiento?
    - ¿El proyecto posee redifusión web (e. g. RSS), listas de discusión o foros de discusión? ¿Qué tipos de listas de discusión se mantienen: anuncios, usuarios, desarrolladores?
    - ¿Qué tipos de documentación hay disponibles: usuario final, desarrollador, API? ¿Se dispone de publicaciones libres y/o propietarias?
    - ¿Se dispone de planes de formación? ¿Hay tutoriales en línea? ¿Se dispone de entrenamiento de pago?
    - ¿Se dispone de soporte de pago?
    - ¿El proyecto provee los fuentes, bibliotecas de desarrollo y distribuciones binarias?

[4] Es importante considerar que las unidades productivas TI en Venezuela en el apartado “Nosotros” de sus sitios web suelen colocar una información ambigua pero rara vez aparecen los nombres y la información exacta de contacto directa de los miembros, esto contrasta con unidades productivas, por ejemplo, brasileñas o argentinas que exhiben de modo claro y abierto su recurso humano como un valor importante para la organización.

- ¿Cuál es el principal lenguaje de desarrollo? ¿Se utiliza un lenguaje estable y bien conocido?
- ¿Se ofrece al menos una versión estable? ¿Si no hay una versión estable, de qué tipos de versiones se dispone; desarrollo, alfa, beta, candidata, etc.?
- ¿Se ofrece una versión propietaria alternativa?
- ¿Se conocen vulnerabilidades de seguridad para el software? ¿Cuál es la susceptibilidad a riesgos de seguridad?

Apuntes para la Certificación de Software Libre en Venezuela.

Lo anterior es apenas una guía de los criterios que se podrían utilizar para evaluar la calidad de los proyectos de desarrollo de software libre. Esta guía con facilidad se puede ampliar con mesas de trabajo y/o discusiones en línea con expertos nacionales e internacionales en desarrollo de software, a fin de incluir otros componentes de la producción de software y equilibrar la balanza de los criterios antes expuestos. Es importante discriminar entre los criterios que se pueden utilizar para evaluar las herramientas utilizadas en el desarrollo, y los criterios para evaluar el software desarrollado. De hecho, en un entorno marcado por la aplicación de tecnologías ágiles para la gestión de proyectos y desarrollo, es lógico pensar que las decisiones sobre estos criterios puedan ir variando con el tiempo según se gana experiencia y los equipos de trabajo se compenetran más.

Nótese que la sugerencia de fondo es que en lugar de imponer cargas administrativas adicionales a las unidades productivas para satisfacer un modelo de mejoramiento de procesos cuya efectividad carece de fundamentos científicos más allá de de representar las opiniones de de una parte interesada, el objetivo es promover en los equipos de trabajo la aplicación de buenas prácticas de desarrollo ampliamente reconocidas por las comunidades de desarrollo *de software libre*. De igual manera, se recomienda ir más allá y promover la aplicación de prácticas propias del socialismo, como la colaboración entre las unidades productivas, mejorar la calidad de vida en el trabajo y modelos de toma de decisiones abiertos y participativos, entre otros.

En todo este proceso, lo que resulta, quizás, mucho más clave, es que el ente o instancia encargada de la observación y valoración del software, comprenda sobre cuáles son exactamente las condiciones que debe reunir el software a utilizar en cada contexto y cuál es su papel en todo el proceso de verificación que realmente las cumpla, y no que simplemente confíe en las certificaciones como una solución “llave en mano” que les permita evadir responsabilidades.

Por otro lado, un equipo de trabajo fortalecido con prácticas de excelencia, verá la búsqueda de la satisfacción del deseo de quien será usuario final de su aplicación, como algo natural y cuyo logro supera, en mucho, el mero cumplimiento de una lista de requerimientos sobre

procedimientos seguidos durante el desarrollo. En ese proceso, incluso, será oportuna la identificación, de la mano con el usuario final de la aplicación, de elementos inherentes a sus propios procesos que quizás no sean aún identificados por éste y que resulte vital evidenciarlos.

Referencias

AJ Center. (2009). CMMI requisito para trabajar con la Administración Pública en España. *Centro Internacional Angel Jordan para la Competitttividad del Software*. Recuperado 13 de febrero de 2013, a partir de http://ajcenter.net/cmmi-requisito-para-trabajar-con-la-administr

Bezroukov, N. (2013). A Slightly Skeptical View on CMM: Capability Maturity Model as an example Of Cargo Cult Software Engineering. *Softpanorama.info*. Retrieved February 10, 2013, from http://www.softpanorama.info/SE/CMM/index.shtml

Chemuturi, M. (2011). New Paradigm for Software Quality. *Mastering software quality assurance : best practices, tools and techniques for software developers* (pp. 217–233). J. Ross Publishing.

Garzás, J. (2010, diciembre 22). Hoy, sin una certificación de la calidad software es muy difícil ganar proyectos. . Recuperado 14 de febrero de 2013, a partir de http://www.javiergarzas.com/2010/12/certificacion-software-competitividades-muy-difcil-ganar-proyectos.html

Open Logic (2012). Guide to Adopting & Distributing Open Source Software. Disponible desde www.openlogic.com/ Consulta realizada el 5 de febrero del 2013.

Open Logic (2008). OpenLogic Certification Process for Open Source Software. Disponible desde www.openlogic.com Consulta realizada el 5 de febrero del 2013