"Revista Virtual Universidad Católica del Norte". No. 37, (septiembre-diciembre de 2012, Colombia), acceso: [http://revistavirtual.ucn.edu.co/], ISSN 0124-5821 - Indexada Publindex-Colciencias (B), Latindex, EBSCO Information Services, Redalyc, Dialnet, DOAJ, Actualidad Iberoamericana, Índice de Revistas de Educación Superior e Investigación Educativa (IRESIE) de la Universidad Autónoma de México. [**Pp. 44-61**]

Una evaluación a las herramientas libres para pruebas de software

An Evaluation of Free Tools for Software Testing

Une évaluation des outils libres pour test logiciel

Edgar Serna M.
Ms.C. Ingeniería de Sistemas
Profesor investigador, Instituto Tecnológico Metropolitano (ITM)
Grupo Automática y Electrónica
edgarserna@itm.edu.co

Alexei Serna A.
Estudiante de último año de Ingeniería de Sistemas
Investigador del Instituto Antioqueño de Investigación (IAI)
Grupo CCIS
alexeiserna@gmail.com

Tipo de artículo: Reflexión
Recibido: 24-septiembre-2012
Evaluado: 03-octubre-2012
Aprobado: 08- octubre -2012

Contenido

Resumen
Antes de elegir una herramienta de prueba de software se debe pensar en los requisitos y necesidades del proyecto, comparar y probar algunas de ellas. Al parecer, el hecho de que no exista una tarifa de licencia para el software libre es justificación necesaria para seleccionar una u otra de ellas, sin embargo, existen otros factores

"Revista Virtual Universidad Católica del Norte". No. 37, (septiembre-diciembre de 2012, Colombia), acceso: [http://revistavirtual.ucn.edu.co/], ISSN 0124-5821 - Indexada Publindex-Colciencias (B), Latindex, EBSCO Information Services, Redalyc, Dialnet, DOAJ, Actualidad Iberoamericana, Índice de Revistas de Educación Superior e Investigación Educativa (IRESIE) de la Universidad Autónoma de México. [**Pp. 44-61**]

que influyen en el costo total de la herramienta. En este trabajo se analizan algunos mitos acerca de las herramientas de prueba de código abierto, se describen sus pros y sus contras, se refiere su evolución hasta convertirse en parte integral de las estrategias de desarrollo, se analiza el efecto que tendrá su evolución en el mercado, se evalúan algunas de las más populares y se proponen algunos trabajos futuros.

Palabras clave
Herramientas de prueba de código abierto, Desarrollo de proyectos, Desarrollo tecnológico, Prueba del software.

Abstract
Before choosing a software testing tool we must consider the requirements and needs of the project, compare some feasible tools and then try one or two of them. Apparently the absence of license fee costs for free software could explain the choice of one of the options, but there are other factors that influence the total cost of the tool. This paper analyze some myths about open source testing tools by describing their pros and cons, relating its evolution until become an integral part of development strategies, the effect it will have on the evolution market is analyzed, some of the most popular are assessed and some future work is proposed.

Keywords
Open source testing tools, Software testing, Project development, Technological development.

Résumé
Avant de choisir un outil de test logiciel on doit considérer les requêtes et nécessités du projet, on doit comparer quelques outils faisables et, finalement, tester une ou deux alternatives. Apparemment le fait de qu'il n'a pas un tarif de licence pour le logiciel libre peut être une justification suffisante pour choisir une ou autre des outils, mais, il y a des autres facteurs qu'influent sur le cout total de l'outil. Dans cet article on analyse quelques mythes au sujet des outils libres de test logiciel, on décrit ses pours et ses contres, on relate leur développements, jusqu'a leur transformation dans une partie intégrale des stratégies de développement, on analyse l'effet qu'aura sur le marché, on évalue quelques des plus populaires et on propose quelques travails futurs.

Mots-clés

Outils libres pour test logiciel, test logiciel, développement de projets, développement technologique.

"Revista Virtual Universidad Católica del Norte". No. 37, (septiembre-diciembre de 2012, Colombia), acceso: [http://revistavirtual.ucn.edu.co/], ISSN 0124-5821 - Indexada Publindex-Colciencias (B), Latindex, EBSCO Information Services, Redalyc, Dialnet, DOAJ, Actualidad Iberoamericana, Índice de Revistas de Educación Superior e Investigación Educativa (IRESIE) de la Universidad Autónoma de México. [**Pp. 44-61**]

1. Introducción

Cuando se realiza un proyecto de software, una de las actividades consiste en seleccionar una herramienta para las pruebas, por lo tanto, es necesario conocer su contexto. Algunas son multifuncionales, otras se construyen para propósitos limitados, unas son de código abierto y otras tienen un costo bastante elevado para la mayoría de proyectos. Todo esto crea un espacio multidimensional de métricas que se deben tener en cuenta al resolver el problema de la selección. Existen grandes empresas que tienen estándares para definir y seleccionar alguna de ellas, pero el interés de este trabajo es diferente.

Antes de seleccionar una herramienta para pruebas es prudente pensar primero en los requisitos y las necesidades del proyecto, luego comparar algunas candidatas y posteriormente probarlas hasta llegar a la selección más adecuada. De alguna manera, a menudo las herramientas de código abierto se aceptan sin ningún tipo de proceso de selección previo; al parecer la ausencia de una tarifa de licencia es la justificación necesaria. Sin embargo, existen otros factores que influyen en el costo total de este tipo de herramientas y que se deben tener en cuenta desde el principio, de lo contrario es posible que se termine pagando demasiado para lograr muy poco.

Este trabajo tiene la siguiente estructura: luego de la introducción, la sección dos presenta la evolución del código abierto, desde el dominio de las tecnologías *geeks* hasta su conversión en parte integral de las estrategias de desarrollo de muchas de las actuales organizaciones; en el apartado tres se describen algunos mitos y realidades de estas herramientas; en la sección cuatro se analizan y detallan los pros y contras de las mismas; en la cinco se proponen algunos conceptos para su evaluación y, finalmente, en la seis se presentan las conclusiones y el trabajo futuro.

2. Evolución y futuro del código abierto

En esta sección se describe cómo ha crecido el software de código abierto desde el dominio de la tecnología de los *geeks* hasta ser parte integral de las estrategias en software de muchas organizaciones actualmente. Debido a que el software de código abierto está migrando de la infraestructura de la empresa al mercado de aplicaciones (Ivan, 2009), también se analiza el impacto que esto tendrá en el desarrollo tecnológico de los principales proveedores de herramientas para pruebas.

"Revista Virtual Universidad Católica del Norte". No. 37, (septiembre-diciembre de 2012, Colombia), acceso: [http://revistavirtual.ucn.edu.co/], ISSN 0124-5821 - Indexada Publindex-Colciencias (B), Latindex, EBSCO Information Services, Redalyc, Dialnet, DOAJ, Actualidad Iberoamericana, Índice de Revistas de Educación Superior e Investigación Educativa (IRESIE) de la Universidad Autónoma de México. [**Pp. 44-61**]

2.1. Expansión del código abierto

La reciente recesión económica causó que las organizaciones de todo el mundo estrecharan sus presupuestos y que la mayor parte de la industria del software y de servicios sintiera su efecto. Sin embargo, las compañías de código abierto han superado este impacto al mostrar un fuerte crecimiento. En medio de la recesión un artículo en *The Economist* (2009) denominó "*zeitgeist*" a ese crecimiento. En general ha habido un incremento real de historias de éxito del código abierto en los medios de comunicación, mientras que los resultados de las investigaciones acerca de su adopción, realizadas por analistas de la industria como IDC y Gartner, muestran un fuerte crecimiento de este sector.

En 2008, Gartner encuestó a 274 empresas en el mundo y encontró que el 85% ya había adoptado el código abierto y que la mayoría restante esperaba hacerlo dentro del siguiente año (Henley & Kemp, 2008). Esa predicción se convirtió en realidad y el reporte Forrester de 2009 (Hammond, 2009) interrogó a 2000 tomadores de decisiones de software; la conclusión fue que el código abierto había llegado a la cima de la agenda ejecutiva, como señala el autor: "la orden viene desde lo más alto" y muchos jefes están demandando procesos "más rápidos, más baratos y mejores". Esta fue una importante etapa de cambio para el código abierto, que había entrado históricamente de abajo-arriba en el radar ejecutivo de las organizaciones, instalado por equipos de IT y desarrolladores y, gradualmente, había ganado terreno. En su reporte (Crosman, 2012) Accenture hizo eco de esto al expresar que las empresas de servicios financieros, ahora que están limitadas económicamente, están adoptando un punto de vista diferente acerca de las tecnologías de código abierto. Dicho reporte expresa además que los principales bancos no sólo utilizan el código abierto, sino que actualmente contribuyen a proyectos reconociendo un futuro cambio radical y una mayor madurez para su adopción, que va más allá del simple ahorro de costos.

2.2. Impacto del código abierto en el desarrollo tecnológico

Tradicionalmente, gran parte del empuje obtenido por el código abierto ha estado a nivel de infraestructura de TI. Sin embargo, otro gran cambio en su adopción se ha dado en la migración desde esa infraestructura a las aplicaciones. Algunos mercados de aplicaciones han sido penetrados significativamente por el código abierto, como SugarCRM en el mercado de CRM y Alfresco en el mercado de la gestión de documentos. El entorno de las herramientas para pruebas es un caso interesante en el que el código abierto ha tenido un significativo progreso. La organización Open

"Revista Virtual Universidad Católica del Norte". No. 37, (septiembre-diciembre de 2012, Colombia), acceso: [http://revistavirtual.ucn.edu.co/], ISSN 0124-5821 - Indexada Publindex-Colciencias (B), Latindex, EBSCO Information Services, Redalyc, Dialnet, DOAJ, Actualidad Iberoamericana, Índice de Revistas de Educación Superior e Investigación Educativa (IRESIE) de la Universidad Autónoma de México. [**Pp. 44-61**]

Source Testing inició labores en 2003 con cerca de 50 herramientas en su lista y en casi diez años ese número se ha incrementado a más de 450. El incremento en esta actividad es mucho más que lo ocurrido en el mercado CRM, por ejemplo, antes de que Sugar se convirtiera en la aplicación dominante. Sin embargo, el mercado de estas herramientas está más fragmentado en áreas variadas y especializadas, lo que explica el gran número de productos de código abierto existentes. Pero en un área específica, como el seguimiento de defectos, sólo un puñado de herramientas compite realmente por la primera posición.

Además, este mercado es joven y todavía no ha tiene una inversión comercial importante. Pero, independientemente de eso se espera que el progreso permanezca constante. La naturaleza influyente del código abierto en los mercados de software ha hecho que los proveedores comerciales evalúen su posición, y la respuesta fue innovar sus productos con mayor rapidez para mantenerse un paso adelante, o innovar su modelo de negocio. Microsoft Sharepoint es ejemplo de un producto comercial que adoptó el segundo enfoque con el objetivo de alcanzar el primero. Su núcleo es como siempre código cerrado, sin embargo, existen más de 3000 APIs disponibles para que los desarrolladores puedan extender el producto de nuevas e imprevistas formas, creando servicios, complementos y soluciones integradas. Como se señala en CMS Wire, esto se ha traducido en un ecosistema y en una comunidad vibrante para competir con la de cualquier producto de código abierto, un astuto movimiento por parte de Microsoft. El éxito de Sharepoint será replicado más ampliamente en los próximos años por los vendedores comerciales, frente a la competencia del código abierto. Es probable que las inclinaciones de Mercury —Hewlett Packard— se muevan en esa dirección, lo que más adelante le podría generar tiempos interesantes e innovadores.

3. Mitos y realidades de las herramientas libres

Es común escuchar al inicio de los proyectos de software que los directores lanzan expresiones como: "Vamos a excluir a las herramientas de código abierto de la lista que posiblemente utilizaremos para las pruebas" y, al preguntarles por qué, la respuesta que ofrecen es: "No quiero tener problemas adicionales en el futuro con esas herramientas. Vamos a estar bajo presión en este proyecto, por lo que quiero disminuir los riesgos de software generados por terceros" (Pocatilu, 2002).

Así es como los directores de proyectos viven el proceso del desarrollo de software: se tienen que enfrentar a presupuestos, tiempos, recursos humanos y problemas de motivación, y tratan de comparar su propio proyecto con los resultados obtenidos en alguno otro desarrollado bajo código abierto. Sin embargo, se debe tener en cuenta que cada proyecto es único y utiliza reglas diferentes. Este es el principal malentendido que se presenta (Pocatilu, 2006). A continuación se describen los mitos —malentendidos— que normalmente se tienen alrededor de las herramientas de código abierto.

3.1. Calidad

Estructurar un equipo de trabajo es un arte, como muchos directores lo pueden confirmar. Entonces, cómo pueden personas independientes, que ocasionalmente participan en un proyecto de código abierto, producir productos de alta calidad y trabajar como un equipo. Esta es la base de las dudas acerca de los productos de código abierto, sin embargo, cabe señalar que esas personas están lo suficientemente motivadas para participar en cualquier proyecto, que lo convierten en su propia responsabilidad y lo influencian, que lo ven de forma divertida y lo asumen como una especie de desafío. Tratan de dar lo mejor de sí porque lo que hacen es una pieza colaborativa del trabajo. Todos pueden inspeccionar y revisar el producto de los otros miembros del proyecto y cada uno lo comprende con claridad; por lo tanto, no se justifica la preocupación por una "baja" calidad.

3.2. Capacitación

Aquí es necesario ser honestos, ¿quién capacita a los probadores antes de comenzar la construcción de un nuevo plan de pruebas? La realidad es que a menudo se capacitan a través de artículos, *podcasts*, foros y muchas otras ayudas colaborativas de la comunidad. Sí, esa es la verdad. Así que, ¿cuál es la diferencia? Cuando se utiliza una herramienta para pruebas de código abierto usualmente se tiene la oportunidad de preguntarle al autor acerca de la misma y de sus características particulares, además, se le pueden sugerir nuevas y se puede conseguir un contacto más estrecho con los miembros del proyecto. Este es un beneficio y no se puede desaprovechar.

3.3. Permanencia

Es factible que esto suceda con todo proyecto que no cumpla con las expectativas de los clientes. La ventaja de las herramientas de código abierto es que existe una estrecha relación con los usuarios finales. Los foros, consejos y otras formas similares de comunicación reflejan las expectativas y su posible desarrollo. Existen diversos productos de código

"Revista Virtual Universidad Católica del Norte". No. 37, (septiembre-diciembre de 2012, Colombia), acceso: [http://revistavirtual.ucn.edu.co/], ISSN 0124-5821 - Indexada Publindex-Colciencias (B), Latindex, EBSCO Information Services, Redalyc, Dialnet, DOAJ, Actualidad Iberoamericana, Índice de Revistas de Educación Superior e Investigación Educativa (IRESIE) de la Universidad Autónoma de México. [**Pp. 44-61**]

abierto que tienen éxito, como Linux y Open Office, pero aquí no es el espacio para debatir acerca de su supervivencia.

3.4. Actualizaciones

Nuevamente la calidad. Pero ¿será que los clientes proceden inmediatamente a conseguir una actualización? No, esa es la vida real. Conseguir una actualización para una herramienta de código abierto no es tan fácil ni tan difícil como para cualquier producto comercial. Para corroborarlo sólo se necesita dialogar con los desarrolladores. Es casi seguro que utilizan el software de código abierto para propósitos de uso personal y de trabajo y, tal vez, sean los evaluadores más activos de estos productos.

Los proyectos tienen un presupuesto limitado en lo que tiene que ver con la capacitación en el uso de herramientas para pruebas, por lo tanto, al disminuir los costos de estas herramientas es posible incrementar el presupuesto en capacitación y hacerlo más eficiente, más enérgico y más flexible. Además, para lograr productos de calidad se debe buscar el mejoramiento permanente de las pruebas y el proceso de desarrollo, y probar nuevos enfoques y metodologías. Por lo tanto, ese es el momento para ensayar las herramientas de prueba de código abierto. Además, se pueden personalizar para cada proyecto, lo que es importante cuando es tan específico que es difícil encontrar soluciones, o cuando se necesita sólo una característica particular y no las cientos de características incluidas en otros productos para cubrir aspectos que no se tienen en el proyecto.

En la tabla 1 se resumen los mitos y realidades de las herramientas de prueba de código abierto.

Tabla 1. Mitos y realidades de las herramientas de prueba de código abierto.

Aspecto	Mito	Realidad
Calidad	No tienen la misma calidad de su contrapartida comercial.	Los equipos de desarrollo se unen por iniciativa de colaboración simultánea.
Capacitación	No hay forma de capacitar al equipo antes de iniciar el proyecto.	La comunidad tiene múltiples medios colaborativos al servicio de los desarrolladores.
Permanencia	Son factibles de cancelación por falta de apoyo económico	Lo mismo le puede suceder a cualquier proyecto de software comercial. La sostenibilidad se garantiza a través de una comunidad motivada y con retos permanentes.
Actualizaciones	No ofrecen rápidamente paquetes de actualización.	Tampoco lo hacen los productos comerciales, la diferencia es que

		los usuarios tienen acceso al mismo autor para solicitar asesorías.

4. Pros y contras de las herramientas libres

En el proceso para seleccionar una herramienta para pruebas es necesario considerar factores que en el ciclo de vida del proyecto entran a jugar diversos papeles, especialmente durante las fases de implementación y de mantenimiento. Cuando estos factores se hayan tenido en cuenta será posible hacer una elección satisfactoria. Desde las fases iniciales del ciclo de vida de un proyecto hasta el momento en que se implementa es importante poder seleccionar una herramienta para pruebas; es un proceso que se hace por tanteo y que es conocido como Prueba De Concepto —PDC— (Korel, 1990), en el que la herramienta se pone a "prueba". Cuando en ese proceso se trata por igual a las herramientas comerciales y a las de código abierto será posible elegir a la que mejor se adapte a cada situación.

Para todo proyecto de software es necesario determinar el alcance de las pruebas, es decir, es para un proyecto o para todos los proyectos en la organización, qué tan compleja es la aplicación a probar, o si existe una sola aplicación bajo prueba o son varias. Además, estas herramientas se necesitan para articularlas en la arquitectura global de las pruebas, y se deben ver como facilitadoras del proceso, no como "respuestas". Con base en esos requisitos, a continuación se hace un análisis comparativo entre herramientas comerciales y de código abierto.

4.1. Costos y licencias

Además de los derechos de licencia inicial otros costos del código abierto son potencialmente más bajos. Por ejemplo, generalmente tiene menores requisitos de hardware que las alternativas comerciales, como OpenSTA, utilizado para pruebas simples de páginas ASP y que soporta hasta 3000 usuarios virtuales sobre un generador de carga de 1 Gb de RAM con un único procesador P4 de 1Ghz con Windows 2000, mientras que LoadRunner, en la misma configuración, puede operar un máximo de 1000 usuarios virtuales (Ivan, Boja & Ciurea, 2007).

La etiqueta de código abierto no implica que el software sea gratuito, porque los desarrolladores pueden y de hecho lo hacen cobrar por el mismo. "Código abierto" se refiere a la disponibilidad del código fuente, lo que permite cambiar el programa y adaptarlo a cada situación específica.

Este software se distribuye bajo diferentes licencias, por ejemplo, Berkeley Software Distribution —BSD— permite a cualquier persona modificar el código fuente sin ningún tipo de obligaciones, mientras que General Public License —GPL— indica que los cambios en el código fuente se deben entregar a la comunidad.

Linksys es un ejemplo de lo que puede suceder si se utiliza código modificado y no se respeta la GPL (Ciurea, 2009). En 2003 esta empresa lanzó el *router* inalámbrico WRT54G y algunas personas de la Linux Kernel Mailing List lo revisaron y encontraron que su *firmware* se basaba en componentes Linux. Debido a que Linux se distribuye bajo GPL, los términos de esta licencia obligaron a que Linksys dejara disponible ese código. No es claro si Linksys estaba consciente de la herencia Linux del WRT54G y sus requisitos fuente asociados al momento en que lanzó el *router* pero, en última instancia, bajo la presión de la comunidad del código abierto tuvo que liberar el código *firmware*. Con el código en la mano los desarrolladores aprenden exactamente cómo hablar con el hardware dentro del *router* y la forma en que las características adicionales del código pueden apoyarlo.

Si algún desarrollador tiene planes de modificar el código fuente de una herramienta de código abierto, para aplicarla en funciones personalizadas, debe saber bajo qué licencia fue liberado. Si el software comercial requiere un valor de licencia, el software de código abierto puede venir con obligaciones de otro tipo.

4.2. Plataformas y características

El software comercial maduro usualmente ofrece más características que sus contrapartes de código abierto, que van desde una instalación más fácil y una mejor secuencia de comandos manuales, hasta ejemplos de *scripts* y reportes de fantasía que, a menudo, soportan una amplia gama de plataformas. Actualmente, las herramientas de código abierto sólo pueden probar aplicaciones basadas en web, mientras que las herramientas comerciales también se pueden utilizar con aplicaciones cliente-servidor local. Cada vez es más común que los sistemas adelantados utilicen protocolos de Internet como HTTP/S, IMAP y POP3, sin embargo, a menudo los menos adelantados están conformados por un *mainframe* viejo pero en buen estado que puede trabajar por diez o más años.

Una de las características importante de una herramienta es el soporte que trae asociado. Cuando surgen problemas el proveedor tendrá que resolverlo para el cliente. Este proceso puede tardar algún tiempo, pero

conocerlo mantiene tranquilo al cliente. Con herramientas de código abierto él es el que debe solucionar el problema. La comunidad puede o no ayudarle, pero el peso de la solución está sobre sus hombros. Al elegir una de estas herramientas para automatizar las pruebas se debe estar preparado para soportar el tiempo necesario antes de conseguir que funcione plenamente. Si se subestima este esfuerzo extra, la herramienta puede llegar a ser más costosa que su contraparte comercial. Por otro lado, si se cuenta con la capacidad para resolver cualquier problema que surja, ¿por qué pagar por características que no son necesarias?

Los usuarios admiten que para implementar y mantener herramientas de código abierto se requiere más habilidad y más ingeniería, en comparación con lo requerido por las comerciales. Pero, después de una inversión en la capacitación inicial para adquirir esas habilidades, los costos a largo plazo son más bajos. Obviamente, eliminando el valor de las licencias se ahorra dinero para rubros como la capacitación, sin embargo, encontrarla para herramientas de código abierto es más difícil que para las comerciales, y sólo algunas tienen programas de certificación.

El código abierto se vuelve más importante cada vez que se hace un PDC, porque no siempre está bien documentado y puede ser más difícil de instalar. El nivel de habilidad técnica requerido para instalar una de estas herramientas podría ser mayor debido a que muchos proyectos no se centran en los asistentes de instalación. Una vez instaladas, estas herramientas se actualizan automáticamente, mientras que con otras el seguimiento a las actualizaciones se debe realizar manualmente.

4.3. Probadores y desarrolladores

Las herramientas comerciales apelan a los probadores con la promesa de que no requieren secuencias de comandos. Dado que esas secuencias son su negocio principal, los desarrolladores no son susceptibles a esta promesa y tienden a preferir las herramientas de código abierto para automatizar las pruebas, porque cuentan con las habilidades y los conocimientos suficientes para ajustarlas a sus necesidades.

En un entorno de desarrollo de código abierto los probadores y los desarrolladores cooperan más estrechamente que en un entorno en cascada. Los primeros ayudan a los segundos a escribir las pruebas unitarias y éstos les ayudan a automatizar las pruebas funcionales. La prueba se considera como una parte integral del proceso de producción y

como responsabilidad de cada miembro del equipo. En este contexto el papel del probador cambia de "portero" a "jugador de campo".

La automatización de las pruebas y la integración continua de nuevo código son prácticas esenciales en los proyectos de código abierto. Parte del ciclo de desarrollo puede ser la primera prueba, es decir, las pruebas se escriben antes que el código, por lo que los desarrolladores escriben código que pasa las pruebas. Posteriormente se integra en el producto y las pruebas se añaden a la suite de regresión, que se aplican automáticamente al nuevo producto. El código base y la suite de pruebas crecen de forma altamente interactiva.

Estos proyectos utilizan herramientas de código abierto para sus propósitos de automatización de las pruebas; lo que se puede explicar por el hecho de que esa actividad no es responsabilidad exclusiva de los probadores, sino que es conjunta entre probadores y desarrolladores trabajando juntos y en equipos multifuncionales.

4.4. Objetos y protocolos

Existe una diferencia fundamental entre las herramientas para automatización de pruebas y las de pruebas de rendimiento. Las primeras trabajan a nivel de la interfaz gráfica de usuario —GUI—, mientras que las otras lo hacen a nivel de protocolo. La pregunta para las de automatización es si reconocen correctamente todos los objetos en las ventanas de aplicaciones diferentes, y el porcentaje logrado en las respuestas no es del 100%. Si ese reconocimiento es inferior al 50% los ingenieros de automatización se verán forzados a ejecutar tantas soluciones que no alcanzarán el objetivo de la misma (Vail, 2005), y establecer y mantener la herramienta tomará más tiempo que el requerido para hacerlo manualmente.

Para una herramienta de pruebas de rendimiento la pregunta es si reconoce correctamente los protocolos de comunicación cliente-servidor. Esta pregunta sólo se puede responder haciendo un PDC en su propio entorno y con la aplicación misma. Incluso si la documentación de una herramienta, comercial o de código abierto, dice que soporta el protocolo, se puede encontrar que no es así, por ejemplo, debido a las diferencias entre versiones.

4.5. Herramientas e infraestructura

Un PDC también es útil para comparar los resultados de las pruebas de diferentes herramientas, porque también son software y contienen errores que pueden aparecer en cualquier momento. En un experimento con WebLOAD, OpenSTA y LoadRunner instaladas en el mismo equipo y

ejecutando el mismo script, en un laboratorio aislado y por fuera de la red corporativa (Ivan *et al.*, 2007) y utilizando un sólo usuario virtual, se encontró diferencias de un 30% en los tiempos de operación reportados. Este experimento pone de manifiesto el hecho de que las herramientas no coinciden entre sí. Entonces, ¿cómo se puede saber si coinciden en la realidad?

Las diferencias en los tiempos de operación las causan las diferentes formas en que las herramientas trabajan y cómo se miden los tiempos de respuesta. Es posible tener una idea confiable sobre el rendimiento, de cualquiera de las herramientas, con base en las diferencias en los tiempos de operación bajo cargas diferentes. Los tiempos de operación absoluta pueden o no ser realistas pero, por lo menos, el rendimiento relativo bajo cargas diferentes lo debe ser y las mediciones reales lo respaldan. Además, las herramientas de monitoreo, como Perfmon, Rstatd, Tivoli, HPOpenview, por nombrar algunas, ayudan a determinar la carga que se debe colocar sobre una infraestructura determinada.

En la selección de la herramienta también se debe tener en cuenta la infraestructura de la aplicación bajo prueba. Si se utiliza un nivelador de carga en la infraestructura, sería muy útil poder emular el comportamiento de las diferentes direcciones IP que acceden al sistema —IP *spoofing*—; pero no todas las herramientas soportan esto. Además, si se tiene diferentes grupos de usuarios que ingresan a la aplicación a través de distintos canales —WAN, LAN y ADSL—, se necesita la capacidad de emular el comportamiento de la infraestructura desde diferentes redes. Por lo general, este tipo de servicios sólo se encuentra en herramientas comerciales (Cristescu, 2009).

Para la validación final es necesario conseguir algo de tiempo en el hardware de producción antes de entregar la aplicación bajo prueba, especialmente cuando difiere del entorno mismo. Para lograrlo es necesario incorporar el entorno de producción en el PDC, lo que no debe ser difícil de conseguir cuando no se utiliza el hardware de producción, por ejemplo, en un fin de semana. Un probador de desempeño siempre se preguntará por los datos que producen las herramientas y, de vez en cuando, interactúa manualmente con la aplicación bajo prueba para ver por sí mismo lo que experimentará el usuario cuando la aplicación esté en operación.

Una herramienta de automatización de pruebas o una de pruebas de rendimiento sólo es una pequeña pieza de un *framework* de pruebas más grande. Puede ser difícil integrar una herramienta de código abierto con

el software comercial en uso, pero si es necesario este tipo de integración, entonces también debe ser parte de la PDC. Lo mismo ocurre con las herramientas comerciales, porque algunas de ellas requieren de otras de soporte de pruebas para hacer seguimiento a los defectos, para controlar su procedencia y para administrar las pruebas, lo que obliga a que también se deban comprar y al final se termina pagando más de lo que inicialmente se presupuestó.

Regularmente, los proveedores de herramientas comerciales organizan PDC para mostrar lo que sus productos son capaces de hacer. Con el código abierto también existe quien esté dispuesto a ayudar pero su objetivo es diferente porque no quiere vender la herramienta, debido a que probablemente es libre de todos modos, pero sí querrán vender servicios de consultoría. Si no existe esta ayuda se debe organizar la PDC de cuenta propia pero primero es necesario asegurar que la herramienta esté al alcance y que se ajusta a la situación específica.

4.6. Conocimientos y compromiso

Antes de comenzar a utilizar herramientas para automatizar pruebas o para pruebas de rendimiento se necesita cierto nivel de documentación de la prueba. Al mejorar la documentación se puede ganar algo de tiempo para la selección de la herramienta, lo que se reflejará en todo el proceso de automatización; este paso se debe realizar con independencia de las herramientas a utilizar. Para la automatización de pruebas se debe contar con el estado de los resultados esperados en los casos de prueba y, aunque a menudo no están documentados, se espera que los probadores comprendan lo que pueden obtener. Estas herramientas no son inteligentes y se les debe indicar explícitamente lo que se espera como resultado. Para las pruebas de rendimiento se necesita saber lo que cada usuario está haciendo y cuándo lo está haciendo, es decir, se necesitan los perfiles de uso.

Otro inconveniente es la ausencia de un ingeniero de pruebas dedicado y experimentado. Sin importar cual herramienta se seleccione, se necesita tiempo y habilidad para implementarla y utilizarla y, si el equipo no cuenta con un ingeniero de este tipo para la automatización, sin importar la herramienta que se utilice el proyecto estará condenado al fracaso. Herramientas infalibles no existen, pero si se toma suficiente tiempo para pensar antes de iniciar, será posible elegir la más adecuada para el proyecto. Además, siempre se debe estar preparado para invertir en una herramienta complementaria.

"Revista Virtual Universidad Católica del Norte". No. 37, (septiembre-diciembre de 2012, Colombia), acceso: [http://revistavirtual.ucn.edu.co/], ISSN 0124-5821 - Indexada Publindex-Colciencias (B), Latindex, EBSCO Information Services, Redalyc, Dialnet, DOAJ, Actualidad Iberoamericana, Índice de Revistas de Educación Superior e Investigación Educativa (IRESIE) de la Universidad Autónoma de México. [**Pp. 44-61**]

En la tabla 2 se resumen los pros y los contras de las herramientas de prueba de código abierto.

Tabla 2. Resumen los pros y los contras de las herramientas de prueba de código abierto.

Aspecto	Pros	Contras
Costos y licencias	Licencia inicial libre. Costos asociados bajos.	Puede requerir inversiones posteriores en asesorías y actualizaciones.
Plataformas y características	Plataformas y características limitadas al contexto.	Los usuarios pueden necesitar características que se adecuen a plataformas por fuera del contexto.
Probadores y desarrolladores	El trabajo de los probadores y los desarrolladores es colaborativo y en equipos integrales.	No es fácil conseguir que en los contextos de uso se puedan conformar equipos con estas características.
Objetos y protocolos	Se adaptan a los contextos específicos de uso.	Las modificaciones en objetos y protocolos generan nuevos desarrollos y modificaciones.
Herramientas e infraestructura	Son compatibles con la mayoría de herramientas de código abierto y se pueden instalar en todo tipo de infraestructura.	La compatibilidad puede disminuir cuando se trata de proyectos integradores de diferentes productos y en diferentes infraestructuras.
Conocimientos y compromiso	El mantenimiento y las actualizaciones al código las puede realizar el mismo usuario.	Se requieren amplios conocimientos de ingeniería para instalarlas, modificarlas y mantenerlas.

5. Cómo evaluar herramientas de código abierto

Una pregunta frecuente de los usuarios potenciales del código abierto se relaciona con su fiabilidad. El código disponible para los productos de código abierto más populares y maduros tiende a ser revisado por cientos de desarrolladores, mucho más de los que la mayoría de empresas de software comercial puede tener para probar sus propios productos, por lo que lo que la calidad y la fiabilidad de este código tienden a ser altas.

Generalmente, el desarrollo de código abierto evita muchos de los procedimientos que normalmente se asumen como "buenas prácticas" en el desarrollo de software, pero sin afectar la calidad del producto. Por ejemplo, es poco probable que en proyectos de código abierto se documenten detalles acerca de esfuerzos y planes de programación, plazos determinados y evaluaciones de riesgo, pero sí se encontrarán procesos de liberación rápidos e iterativos, que resultan de las mejoras

continuas propuestas por los desarrolladores que contribuyen con iteraciones, mejoras y correcciones. Un estudio académico a 100 aplicaciones de código abierto (Ivan & Ciurea, 2009) encontró que la calidad del código estructural resultaba ser mayor de lo esperado, que era comparable con el software comercial y que los proyectos más ampliamente conocidos, como Apache y el kernel de Linux, en realidad muestran una densidad de defectos sustancialmente más bajos que los productos comerciales comparables. Esto hace posible que el código abierto ofrezca productos de alta calidad.

Sin embargo, ¿cómo separar lo bueno de lo malo? Con un poco de investigación en la web es fácil demostrar un número de áreas clave que respaldan la calidad del software de código abierto:

- Una comunidad sostenible que desarrolla y depura código de forma rápida y eficaz. Todos los proyectos de código abierto tienen áreas comunitarias en sus sitios, donde se puede ver el número de usuarios registrados y medir el volumen de actividad en los foros, wikis y páginas de las versiones, lo mismo que repositorios de código abierto legibles que permiten ver con qué frecuencia se entrega dicho código.

- Documentación visible en el sitio del proyecto. Un código modular soportado por buena documentación servirá para atraer a un nuevo cuerpo de desarrolladores y ampliar/atraer constructores. Como ejercicio, basta con preguntarle a un desarrollador comercial si es posible echarle un vistazo a su código para evaluar la calidad.

- Un equipo central bien dirigido que responde rápidamente a los comentarios de los revisores pares y a las contribuciones de código. Esto da como resultado una innovación rápida y que se incremente la calidad, lo que es visible a través de los foros de los proyectos y de las listas públicas de correo.

- Reutilización de librerías de código establecido y probado en lugar de escribir todo el programa desde el principio. Lo que ayuda a unificar una alta calidad porque estas librerías deben ser obvias para la revisión de la documentación del desarrollador.

También existen algunos modelos formales que se pueden utilizar para lograr un marco de evaluación (Cristescu, 2009), como Navica's Open Source Maturity Model —OSMM— que se publica bajo una licencia abierta y se utiliza para evaluar la madurez de los elementos clave del proyecto. El Business Readiness Rating es un modelo de evaluación que se

encuentra en fase de desarrollo por organizaciones que buscan expandir los modelos OSMM para desarrollar un estándar abierto para el índice OSS.

6. Conclusiones y trabajo futuro

El código abierto es un dominio amplio y un enfoque reducido no puede conducir a una comprensión completa de todo el fenómeno; para lograrlo se debe realizar un análisis coherente de sus productos, de la forma en que se difunden y, lo más importante, de los efectos que su uso tiene sobre el dominio de las actuales TI. Sin importar cuál sea el beneficio que se logre siempre habrá problemas, los cuales se deben incluir en el portafolio de proyectos de los desarrolladores de software cuya actividad se oriente a la eficiencia. Estos problemas se convierten en preguntas, procedimientos y soluciones que poco a poco se transformarán en productos de software libre de gran escala para uso general.

El área de problemas para la que se desarrollan herramientas de código abierto se ha convertido en una fuerte competencia para las empresas que producen software licenciado. Bajo ninguna circunstancia los desarrolladores de software licenciado abren su código, aunque tengan los peores resultados reportados. Es una competencia tácita entre código abierto y software con licencia en la que el primero es el motor principal.

La automatización de las pruebas da lugar a un proceso eficiente y ayuda a disminuir los costos de desarrollo y, aunque algunas pruebas todavía se tienen que realizar de forma manual, existen algunas específicas en las que las herramientas automatizadas son inútiles. Al utilizar menos tiempo en la creación y aplicación de los casos de prueba la automatización del proceso podría reducir los costos asociados, por lo que es importante seleccionar una herramienta adecuada para el éxito del proyecto.

Los productos de código abierto se desarrollan mediante trabajo colaborativo en el que el éxito se logra porque los miembros deben mostrar buena voluntad y responsabilidad. El carácter colaborativo y esencialmente social es necesario para hacer frente a los grandes proyectos de código abierto. Una aplicación de este tipo crea un ambiente en el que las personas trabajan mejor juntas, pueden compartir información sin limitaciones de tiempo y espacio y que se caracteriza por aspectos como actividades conjuntas, medio ambiente compartido y la forma de interacción. El desarrollo de código abierto no se organiza caóticamente sino que sigue una dirección clara: incrementar el rendimiento para el procesamiento de grandes intereses.

El ritmo de desarrollo del código abierto está permanentemente en ascenso y el número de componentes distribuidos también es amplio; por todo esto es conveniente pensar en trabajos futuros como:

- Identificar los componentes de código abierto que los usuarios pueden utilizar en sus procesos.
- Crear, publicar y recomendar una jerarquía de componentes de código abierto de acuerdo con su calidad y comportamiento en uso.
- Integrar los componentes lo más rápidamente posible teniendo en cuenta las demandas del usuario. Cuando los componentes estén terminados es necesario disponer de motores que los encuentren e integren en una estructura unitaria en la que sean operativos. Todo instrumento debe estar destinado a crear una gestión avanzada en el campo del código abierto, porque un componente existe no necesariamente cuando se utiliza, sino cuando es escrito y publicado en una librería y además debe ser usable, accesible y tener un procedimiento de fácil acceso. Si las listas de parámetros son demasiado largas en los datos de entrada de la prueba, se debe respetar unos requisitos estrictos, porque si los resultados no tienen una estructura flexible la aplicación se vuelve difícil de usar y los usuarios se desaniman y la abandonan.

Se considera que el nivel alcanzado actualmente por la comunidad de código abierto es muy alto y que la información es suficiente para caracterizarla completamente. De esta forma las entidades que la rigen y las técnicas y los métodos para gestionarla, muy pronto darán pie para el desarrollo de una nueva idea que cumpla con todos los requisitos necesarios para discutir acerca de los beneficios del código abierto.

7. Lista de referencias

Ciurea, C. (2009). Collaborative Open Source Applications. *Open Source Scientific Journal*, 1(1), 116-125.

Cristescu, M-P. (2009). Open Source Software Reliability: Features and Tendence. *Open Source Scientific Journal*, *1*(1), 163-178.

Crosman, P. (2012). *Did Weak Risk Management Controls Worsen Knight Capital Loss?* USA: Wall Street & Technology.

Hammond, J. S. (2009). *Open Source Software Goes Mainstream*. USA: Forrester Research, Inc.

Henley, M. & Kemp, R. (2008). Open Source Software: An introduction. *Computer Law & Security Report*, 24(1), 77-85.

Ivan, I. & Ciurea, C. (2009). Quality characteristics of collaborative systems. *Proceedings The Second International Conference on Advances in Computer-Human Interactions*, Cancun, Mexico, 1, 164-168.

Ivan, I. (2009). Open source – State of the Art. *Open Source Scientific Journal,* 1(1), 5-21.

Ivan, I., Boja, C. & Ciurea, C. (2007). *Collaborative Systems Metrics*. Bucharest: ASE Publishing House.

Ivan, I., Pirvulescu, A., Pocatilu, P. & Nitescu, I. (2007). Software Quality Verification through Empirical Testing. *Journal of Applied Quantitative Methods*, 2(1), 38-60.

Korel, B. (1990). Automated Software Test Data Generation. *IEEE Transactions on Software Engineering*, 16(8), 870-879.

Pocatilu, P. (2002). Automated Software Testing Process. *Economy Informatics*, 2(1), 97-99.

Pocatilu, P. (2006). Software Testing Costs. *Economy Informatics,* 6(1), 90-93.

The Economist. (2009). *Born free: Open-source software firms are flourishing, but are also becoming less distinctive*. Recuperado 17 de mayo de 2012: http://www.economist.com/node/13743278.

Vail, C. (2005). *Stress, load, volume, performance, benchmark and base line testing tool evaluation and comparison*. Recuperado 10 de mayo de 2012: http://www.vcaa.com/tools/loadtesttoolevaluationchart-023.pdf.