La gestión de la información en una comunidad OS: el germen de las redes sociales

Carlos Acuña (*), Fernando Usero (**), Josetxo Vicedo (***), Hugo Alexer Parada (***), José María Polvorosa (*)

(*) Telefónica I+D, carlosa@tid.es, jmpa@tid.es; (**) Telvent, fernando.usero@telvent.com; (***) UPM, hparada@dit.upm.es

***Abstract* — Las redes de expertos constituyen la génesis de lo que hoy se conoce como redes sociales. Las comunidades de desarrollo colaborativo heterogéneo, habitualmente de FLOSS (Free Libre Open Source SW), que están nutridas por contribuyentes individuales, grupales o corporativos, y que son la vanguardia de estas redes sociales, ya existían desde hace años. Actualmente, uno de los núcleos de su acción (el otro es el desarrollo) es la gestión de la información, básicamente los procesos de integración y evaluación de calidad de las contribuciones de SW hechas por diferentes desarrolladores. Esto es especialmente relevante para la continuidad del negocio en el sector primario del SW, pero lo es aún más para el sector secundario (aquellas compañías cuyo negocio no es el SW pero se apoya fuertemente en él). Se trata de procesos complejos para lo cual existen algunas herramientas que pueden ayudar a que no lo sean tanto, y cuyo uso empieza a extenderse. Esta ponencia refleja un conjunto de ideas sobre integración y evaluación de la calidad de los módulos FLOSS y muestra el resultado de la evaluación de las herramientas que pueden ser de utilidad. Todo esto para ayudar a reducir el impacto de un axioma clásico de la ingeniería del SW *"reusar lo máximo, usar lo mínimo"* . Es un resumen de resultados obtenidos en el marco del proyecto PROFIT-ITEA-COSI (http://www.itea-cosi.org).**

I. Introducción

Actualmente el FLOSS está escrito por individuos en su 2/3 partes y la recuperación de arquitectura (Figura 1) es especialmente útil para aplicar a módulos FLOSS no sólo para integrarlos sino también adaptarlos.

La integración asume que componentes desarrollados de forma separada puedan trabajar juntos, especialmente en el contexto FLOSS donde la integración es una de las tareas más críticas a pesar de que son de igual o mejor calidad que los componentes COTS (**C**ommercial-**O**ff-**T**he-**S**helf). Este proceso cubre la selección de componentes, la integración, el mantenimiento, la adaptación, etc., y es de gran utilidad dado que el coste de integración de los componentes FLOSS dista de ser gratis.

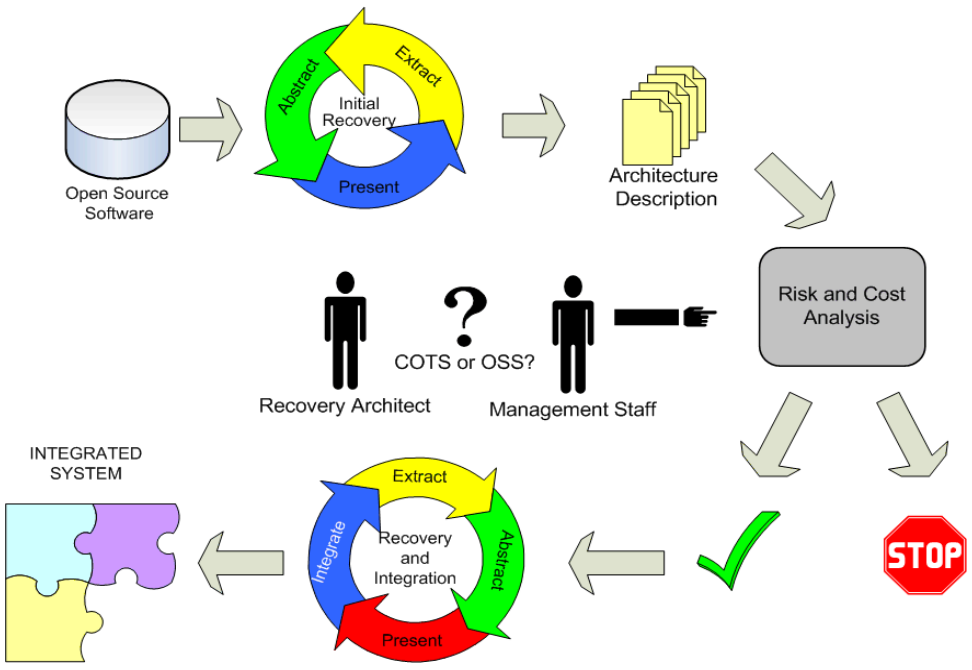

Fig. 1: El proceso de recuperación de arquitectura e integración

II. LA ARQUITECTURA

En este proceso existen dificultades diversas aunque las más relevantes suelen ser la falta de documentación y la naturaleza del cambio. Todos los componentes de SW necesitan una parte de *glueware* para ser integrados.

Si están hechos como bloques monolíticos los problemas que surgen habitualmente son:

- Reusabilidad baja
- No todas las características del componente serán necesarias
- Son difíciles de adaptar

Estas dificultades se pueden abordar promoviendo la modificabilidad (versiones pequeñas y frecuentes), la reusabilidad y la configurabilidad de la arquitectura de integración, con técnicas repetitivas para la integración de componentes y aprovechando los beneficios que brinda una comunidad de FLOSS en términos de integrabilidad y usabilidad.

Habitualmente los componentes FLOSS se implementan como "caja negra" debido a que su modificación es poco rentable. Allí reside la utilidad del proceso de recuperación de arquitectura que evalúa costes y riesgos. Tiene dos etapas. La primera de descripción de la arquitectura, y la segunda de integración, posterior a la decisión positiva.

Mediante el uso de recuperación de arquitectura (un subconjunto de la ingeniería inversa) se puede evaluar la posible adopción de un producto FLOSS. El conocimiento de la arquitectura del producto en cuestión permite, modificarlo para conseguir integrarlo al sistema de modo que el coste de integración sea el menor posible, mejorar la calidad y facilita futuras integraciones. En cualquier caso esta estrategia implica una inversión inicial antes del esfuerzo de integración.

III. LAS HERRAMIENTAS

Existe un espectro significativo de herramientas de este tipo aunque la mayoría adolece de ciertos grados de incapacidad de interoperar con herramientas de otros entornos (fundamentalmente en la exportación de datos). El proceso más novedoso en este sentido se apoya en el concepto de Visualización Continua que agrega la capacidad compleja de Extraer (datos de la DB)-Abstraer (datos de las consultas)-Presentar (diagramas). La idea es entonces extraer los diagramas del componente a evaluar y abstraer sobre dichos diagramas. Esto puede hacerse casi como única opción para un staff con poca experiencia en recuperación de arquitectura, usando UML[1] para crear un modelo abstracto del componente.

La interoperabilidad tiene otras dimensiones. El proyecto COSI, apunta entre otras cosas a las necesidades crecientes de optimizar el esfuerzo dedicado al desarrollo del software. El uso de sus resultados permite evitar dedicar esfuerzos al desarrollo del software llamado "*commodity*" (ya se trate de FLOSS o de COTS) y dedicarlo entonces a desarrollar el SW diferenciador. Sin embargo no se pierde de vista que sólo el 5% del SW que soporta a un producto o servicio (en promedio) es diferenciador[2], por lo cual, el proyecto COSI también aborda la integración del resto con atención a la interoperabilidad como otra de las vertientes para incrementar la eficiencia y la productividad. Una de los resultados de COSI expresa que el uso del llamado *Agile Integration Service: AIS* y la aplicación del concepto *Service Oriented Architecture: SOA*, son fundamentales para afrontar el desafío de la interoperabilidad entre *Building Blocks* y el proceso de evolución/mantenimiento.

En el entorno JAVA las herramientas analizadas en el proyecto, que se pueden destacar son:

- Omondo [3] : analiza el código y genera diagramas estadísticamente
- Jude [4] : incluye propósitos y procedimientos de Omondo y otras funcionalidades de integración
- Test and Performance Tool Platform (TPTP)[5]: monitoriza el rendimiento de aplicaciones durante el proceso de desarrollo (dinámicamente). Es un complemento de las otras dos herramientas y se aborda en la sección IV

A propósito de esta última referencia, la arquitectura Eclipse ha migrado al esquema OSGi[6] justamente para resolver dificultades de integración. Es un hecho relevante si se tiene en cuenta que OSGi ha acompañado las tendencias de Eclipse. Eclipse es entre otras cosas un *test-bed* para compañías de SW y puede ser adecuado dependiendo de las posibilidades de adaptación al modelo al negocio de la compañía que se trate (no todas tienen modelos de negocio compatibles).

IV. LA CALIDAD DEL SW

La calidad del SW tiene tres pilares: fiabilidad, seguridad y privacidad, atributos que además no son independientes entre sí. Sigue siendo poco usual (aunque cada vez lo es más) encontrar información madura sobre gestión de S&P (seguridad y privacidad) a nivel de componente, dado que, por un lado, no se percibe como una característica del producto y, por otro lado los contribuyentes se acercan muy lentamente al interés por aspectos de seguridad y privacidad de sus contribuciones.

Tanto es así que el proyecto Sardonix que nació con vocación de auditoría de Linux, languideció por un par de años hasta la inactividad total, una señal de la falta de interés de los potenciales miembros de esa comunidad en ese momento.

Algo que puede ser de gran ayuda para medir la importancia de los atributos S&P, es estudiar la relación coste/beneficio de la inversión en este área. Un ejemplo claro es el indicador ROSI (Return on Security Investement) utilizado por Microsoft[7]. Dependiendo del resultado de la aplicación del ROSI es posible que la S&P de un determinado componente sea o no una barrera para reutilizarlo o tener garantía de implementación en productos derivados.

Otra dimensión de la evaluación apunta a observar una de las vertientes de la fiabilidad, llamada *Trustworthiness*: la fiablidad de un componente para funcionar correctamente y sin interrupciones. Esto se hace a nivel de componente, arquitectura y sistema. Para estos propósitos, el proyecto COSI ha señalado básicamente cuatro herramientas y/o procesos:

1. RAP[8,] : evalúa las posibilidades de operar correctamente a nivel de componente:

   - calcula la probabilidad de fallo
   - automatiza la simulación del sistema a nivel de arquitectura
   - calcula la probabilidad de fallo del sistema completo
   - se comunica naturalmente con UML actualizando el modelo de acuerdo con los resultados del análisis

2. RT tool, recientemente rebautizada como ComponentBee tool[9]: ayuda a los desarrolladores a seleccionar e integrar los componentes más adecuados. Es capaz de describir el comportamiento (dinámicamente) extrayendo patrones y calculando los valores de fiabilidad usando dichos patrones

3. TPTP (ya mencionado en la sección III) : aborda el ciclo completo de pruebas de rendimiento, desde las pruebas iniciales hasta la monitorización de la aplicación en funcionamiento, incluyendo la edición y ejecución de textos, el trazado, el perfilado y el análisis de la capacidad de conexión.

Se evalúa la posibilidad de que RAP y ComponentBee puedan ser complementarios y se puedan usar juntos. También se está en proceso de integrar ambos como sub-proyectos de TPTP.

V. LA CERTIFICACIÓN DE CALIDAD

Se trata del cuarto proceso recomendado. Por lo general los indicadores de calidad son difíciles de evaluar y el fallo mayor se produce cuando el resultado de la evaluación depende de la opinión subjetiva del evaluador. Incluso algunos indicadores son relativos entre sí y pueden provocar inconsistencias.

La dificultad aumenta cuando se trata de poner el foco en la garantía de calidad más que en la evaluación pura. El proyecto COSI apunta a evitar subjetividades y reducir dependencias por medio de su OSS Quality Measurement Model & OSS Quality Certfication Process [10] (Figura 2).

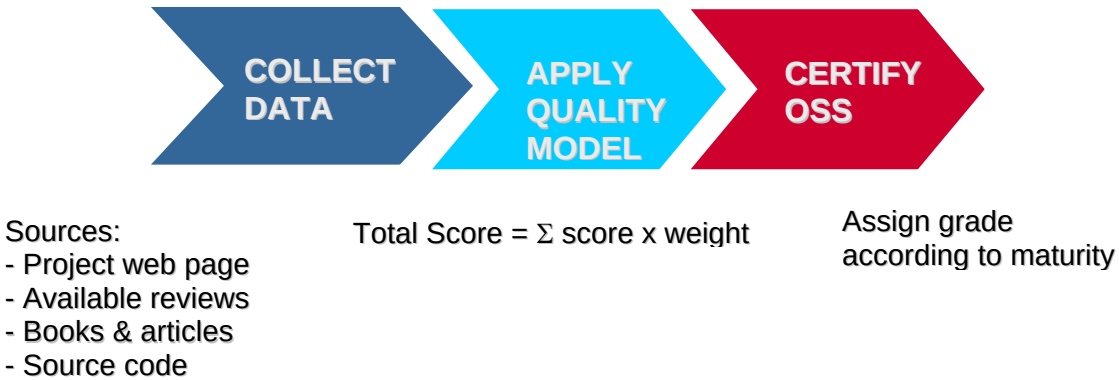

Fig.2: El Proceso de Medida y Certificación de la Calidad FLOSS

Dos de las característica distintivas del mundo FLOSS son, el libre acceso a su código y el hecho de que el desarrollo está impulsado por la comunidad (bottom-up). Justamente esas características son las que permiten potencialmente una multitud de métodos de selección de componentes. Es importante destacar que la posibilidad de disponer de procesos de certificación

de calidad basados en modelos de calidad, es un punto diferenciador muy fuerte del mundo FLOSS. Sobre todo en el modelo que se trata, dado que mide la calidad de atributos que son completamente específicos de FLOSS:

- Documentación y requisitos de *training*: disponibilidad y grado de actualización de la documentación de desarrollador/usuario, actividad de la comunidad, tiempo de respuesta, etc.
- Factor de Reputación: compañías comerciales que hacen soporte, grado de compromiso de dichas compañía, aceptación del mercado, volumen de negocio, etc
- Nivel mínimo establecido de calidad/adecuación del SW; la comunidad es la tarjeta de visita: tamaño, cantidad de proyectos, reputación de los miembros, edad de la tecnología y la comunidad, organización, etc
- Licencias vs Requisitos de Negocio: no es un objetivo puro de calidad pero puede limitar el uso
- Interoperabilidad: numero de descargas, ajuste a standards y formatos de datos, etc

La primera parte del proceso de certificación, la evaluación, se basa en atributos perfectamente medibles. Hay proyectos FLOSS colaborativos (ej.: Flossmole [11]), cuyo objetivo es medirlos explorando todas las fuentes disponibles.

El proceso se completa contrastando el valor medido de los atributos en una tabla, llamada Tabla de Consistencia que tiene en cuenta la interdependencia de muchos de ellos entre sí. Si los valores son adecuados es entonces cuando se realiza la certificación detallada para que el producto, suficientemente maduro pueda ser adoptado con mínimo riesgo.

VI. CONCLUSIONES

Se han descrito las generalidades de una parte fundamental de la gestión de la información en comunidades de FLOSS describiendo algunas particularidades de los procesos necesarios para realizarla. Los procesos abordados fueron los relativos a la Integración de módulos de FLOSS provistos por diferentes desarrolladores, tales como la recuperación de la arquitectura, la evaluación de la calidad y finalmente la certificación de la misma. Se han destacado especialmente herramientas y procesos identificados dentro del proyecto COSI, tales como RAP, RT-ComponentBee y TPTP, y/o creados en el marco del mismo: OSS Quality Measurement Model & Quality Certification Process.

AGRADECIMIENTOS

Se agradece especialmente al Ministerio de Industria Turismo y Comercio de España por la co-financiación y el apoyo al proyecto ITEA-COSI, a través del programa Profit, sin los cuales hubiese sido muy difícil abordar los objetivos descritos.

REFERENCIAS

[1] http://www.omg.org/technology/documents/formal/uml.htm
[2] F.v.d. Linden y otros , “Commodification of industrial software, a case for open and inner source”, OSS07, Junio 14, 2007, Limerick, Ireland. (http://oss2007.dti.unimi.it)
[3] http://www.omondo.com/
[4] http://www.omg.org/technology/documents/
[5] http://www.eclipsecon.com
[6] http://www.osgi.org
[7] http://search.microsoft.com
[8] Carlos Acuña y otros. “Co-desarrollo Usando Código Abierto e Interno en Productos /Servicios con Software Intensivo”, Telecom. I+D 2007, octubre 2007, Valencia, España
[9] http://www.vtt.fi/proj/cosi/cosi_ComponentBEE.jsp
[10] Josetxo Vicedo, y otros. “OSS Quality Measurement Model & OSS Quality Certification”. ITEA-COSI project. Deliverable 3.2.1, slides 128 to 164.
[11] http://ossmole.sourceforge.net