DeTraS/TempusFugit: Herramientas para la investigación en la actividad de los desarrolladores*

Carlos García Campos, Juan José Amor y Gregorio Robles
{carlosgc, jjamor, grex}@gsyc.escet.urjc.es

Libre Software Engineering Lab
Grupo de Sistemas y Comunicaciones
Universidad Rey Juan Carlos (Móstoles, Madrid)

Resumen La medición de la actividad de los desarrolladores es útil por varios motivos. Los jefes de proyecto utilizan técnicas y modelos para poder gestionar el proyecto en todas sus fases (desde la especificación de requisitos hasta todas las fases de pruebas). Uno de los campos más importantes en la gestión de un proyecto es la estimación del esfuerzo que nos va a llevar realizar todo ese trabajo. Nuestro grupo de investigación, formado por investigadores en ingeniería del software y desarrolladores de software libre, está interesado entre otras cosas, en las estimaciones de esfuerzo para el software libre.
En este trabajo se presentará un sistema que nuestro grupo está desarrollando, usando tecnología GNOME, destinado a poder realizar mediciones de actividad de los desarrolladores con el fin de ayudar a la estimación de costes en el software libre. Este sistema está inspirado en una herramienta no finalizada y disponible en el CVS de GNOME (timeline), aunque incluye numerosas mejoras.
En este artículo presentaremos el sistema así como un resumen de las motivaciones que nos llevan a su implementación y a su divulgación entre los desarrolladores de GNOME. Ya que, más que nunca, será necesaria la colaboración de la comunidad de desarrolladores para conseguir que el sistema dé resultados útiles.

1. Introducción

La estimación de costes en el software es uno de los desafíos permanentes en la ingeniería. Estimar costes significa conocer cuántas personas tienen que involucrarse en un proyecto y durante cuánto tiempo. Por tanto, también nos sirve, no solo para saber cuánto cuesta el proyecto, sino también cuánto tiempo se tardará en desarrollarlo.

Resulta obvio que la estimación es un desafío, ya que estamos hablando de predecir algo que no conocemos y sometido a muchas desviaciones. En la ingeniería del software se han construido modelos que permiten, por ejemplo, a partir

* Este trabajo ha sido financiado en parte por la Comisión Europea, dentro del la acción coordinada CALIBRE dentro del programa IST, número de contrato 004337.

de la experiencia previa en otros proyectos, estimar el tamaño de los proyectos futuros (como sucede con el modelo de puntos de función [1]) o el coste a partir de una estimación del tamaño (como sucede con los modelos COCOMO [4] y COCOMO II [5]).

Sin embargo, estos modelos suelen adolecer de, al menos, el siguiente problema: no suelen tener en cuenta más que un parámetro del proyecto (el tamaño del mismo).

En el software privativo no suele ser fácil medir más parámetros aparte del tamaño. Sin embargo, el modelo de desarrollo de software libre, al ser distribuido (lo que suele llamarse *desarrollo global* [6]), implica el uso de herramientas electrónicas que en modelos privativos no suelen utilizarse o cuyos datos no suelen ser fácilmente accesibles. Por ejemplo, se suelen utilizar sistemas de seguimiento de erratas o los desarrolladores se suelen coordinar mediante listas de correo electrónico.

Esto nos da la idea de en qué podemos mejorar la estimación si utilizamos además de las medidas de tamaño del proyecto, otras inferidas a partir de los datos de un Bugzilla o de las listas de correo de los desarrolladores, pues corresponde también a otras actividades en las que los desarrolladores invierten su tiempo, aparte de producir líneas de código fuente.

Sin embargo, esta información sigue sin ser suficiente: gracias a los commits que los desarrolladores hacen a un CVS podemos medir datos relacionados con la productividad en los mismos [10], o gracias a las fechas de los correos electrónicos podemos saber qué participación tiene cada desarrollador en la discusión. Sin embargo, no conocemos realmente cuánto tiempo ha invertido el desarrollador en realizar ese trabajo de codificación que se ha traducido en un envío al CVS o cuánto tiempo real ha necesitado para escribir los correos que hayan quedado registrados en una lista de correo.

Por otro lado, en el modelo de desarrollo tradicional (predominante en el software privativo), los desarrolladores suelen tener, como una tarea adicional, la de rellenar informes diarios acerca de su actividad, estimando de esta forma en qué han invertido su tiempo para el que han sido contratados. Por tanto, al final se sabe con bastante precisión el coste del proyecto, pues se conoce cuánta gente ha trabajado y durante cuánto tiempo.

Sin embargo, en el software libre es muy frecuente que grandes proyectos sean desarrollados, en su mayoría, por voluntarios que emplean su tiempo libre en producir código. En consecuencia, en todos estos proyectos es muy difícil conocer, para cada desarrollador, cuánto tiempo han estado realmente frente al ordenador desde el principio hasta el final de su colaboración en el proyecto (o la liberación del producto que se está desarrollando). Es obvio, por tanto, que con los datos que tenemos es mucho más complicado estimar el coste real del proyecto.

Algo de lo que carecen actualmente muchos proyectos de software libre es, precisamente, de esos informes semanales o mensuales que en muchas empresas se producen. Si bien, en el mundo del software libre usar estos informes serían mucho más imprecisos ya que, mientras que en las empresas los empleados suelen

tener un horario fijo, en el caso del software libre no tiene por qué ser así (especialmente cuando consideramos la aportación de desarrolladores que colaboran con su tiempo libre), dificultándose la estimación final de la actividad desarrollada y el tiempo empleado en ella.

Por todo esto estamos desarrollando un "framework" de traza del trabajo que hacen los desarrolladores en su escritorio. Este conjunto de herramientas almacena únicamente información de interés para la traza del esfuerzo significativo para el proyecto (por ejemplo, saber cuándo se está editando código fuente o cuándo se está participando en una lista de correo) pero no pretende trazar cualquier otro tiempo invertido en tareas ajenas al proyecto (por ejemplo, navegación por páginas web o edición de correos no relacionados con el proyecto).

A modo de resumen, el conjunto de herramientas se denomina "DeTraS" y consta de una parte instalada en cliente, que realiza la captura de eventos relacionados con la actividad del desarrollador y su posterior filtrado para eliminar información que no es de interés, al tiempo que preserva la privacidad del desarrollador; y una parte instalada en servidor. En el cliente, "TempusFugit" permitirá conocer el tiempo dedicado a diferentes tareas por parte del usuario y de manera no intrusiva.

En la siguiente sección se describirá la arquitectura del *framework* DeTraS. A continuación, se discutirá la utilidad que este tipo de herramientas puede presentar a la investigación. Por último se ofrecerán las conclusiones de este artículo.

2. Arquitectura de DeTraS

El sistema DeTraS está compuesto por varias herramientas divididas entre una parte cliente y una servidora de la siguiente forma:

- Cliente: Es la parte del sistema que se ejecuta en el ordenador de cada desarrollador. Las herramientas en esta parte son:
  - Observer (TempusFugit): Es el demonio que se ejecuta en segundo plano y permanece a la escucha de distintos tipos de eventos, que va registrando en un fichero XML.
  - Notifier: Es el script encargado de parsear los datos generados por el observer, realizar los filtrados necesarios que garanticen la privacidad y anonimato del desarrollador y finalmente enviarlo en formato XML a través de un servicio web en el lado del servidor.
- Servidor: Es la parte que recibe los datos de todos los desarrolladores para su posterior procesamiento y análisis.
  - Aplicación Web: es una aplicación web que espera peticiones de los distintos clientes y almacena los datos recibidos en una base de datos.
  - Informes: programas que sean capaces de, a partir de la información almacenada en la base de datos, producir informes de texto y/o gráficos.

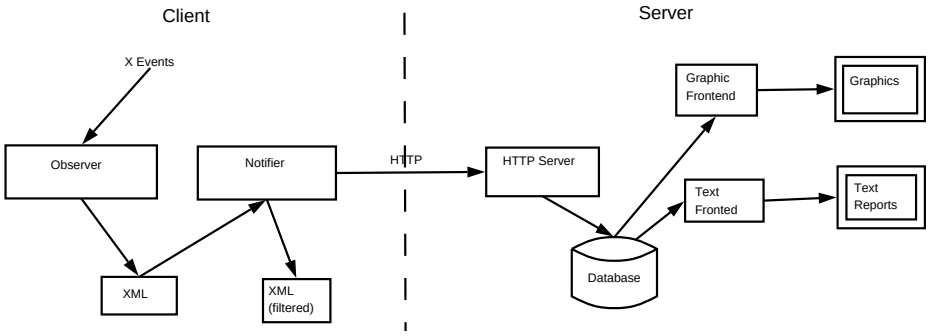

Figura 1. Arquitectura del sistema DeTraS

2.1. TempusFugit

El TempusFugit es la parte del cliente que se encarga de observar la actividad del desarrollador, registrándola para su posterior procesamiento. Es la única parte del sistema desarrollada en este momento, aunque no en su totalidad. Puesto que es la parte que se ejecuta en el cliente y, por tanto, en el ordenador de cada uno de los desarrolladores, esta herramienta tiene una importancia vital. Por este motivo se ha desarrollado siguiendo los siguientes criterios:

- Independiente del escritorio: tan solo depende del sistema de ventanas X Window, que es el éstandar actual en todos los sistemas UNIX. De todas las formas el diseño permite adapatar el sistema a cualquier otro éstandar si fuese necesario.
- Diseño orientado a objetos: además del sistema de ventanas, TempusFugit depende de GLib, la librería de GNOME que nos proporciona un sistema de objetos en C. La naturaleza y cantidad de eventos relacionados con el desarrollo puede llegar a ser muy amplia, por lo que TempusFugit debe ser extensible y tolerante a cambios en el futuro. Gracias al sistema de objetos GObject, TempusFugit está diseñado siguiendo los principios de la programación orientada a objetos, haciendo un uso adecuado de los patrones de diseño que nos aseguran un diseño solido, extensible y tolerante a cambios.
- Silencioso: debido al hecho de que se ejecuta en el ordenador de cada desarrollador, TempusFugit debe resultar lo menos molesto posible, de manera que pase desapercibido. Por ello, permanecerá ejecutándose en segundo plano, consumiendo la mínima cantidad de recursos del sistema y realizando sus funciones de la forma más eficiente posible. Esto nos asegura que no interfiera en la ejecución del resto de los procesos del sistema y pase totalmente desapercibido para el desarrollador.

Detalles de implementación TempusFugit se ejecuta en segundo plano y permanece a la escucha de distintos tipos de eventos que va registrando. El

registro de los eventos se realiza en un fichero XML. Los distintos tipos de evento, así que como el XML producido, son los siguientes:

- Cambios de ventana: cuando se produce un cambio de ventana en el escritorio, se registra el nombre de la aplicación y el título de la ventana actual.

```
<event time=''1145630542'' type=''WindowChange''>
        <app>Evince</app>
        <title>Visor de documentos Evince</title>
</event>
```

- Inicio y fin de sesión: es importante registrar el momento en el que inicia una sesión así como el momento de su fin, ya que éste último marca el fin del evento anterior.

```
<event time=''1145627579'' type=''Session'' event=''Init'' />
<event time=''1145627664'' type=''Session'' event=''End'' />
```

- Períodos de inactividad: cuando no se está realizando ningún tipo de actividad en el ordenador (no hay movimiento de ratón, ni tecleo, etc.) durante un periodo de tiempo, se registra un evento de inactividad. El fin de este periodo viene marcado por el siguiente evento.

```
<event time=''1145819441'' type=''InactiveTime'' />
```

Estos son los eventos que TempusFugit es capaz de detectar y registrar a día de hoy. Está previsto seguir incluyendo nuevos tipos de eventos que puedan estar relacionados con la actividad del desarrollador.

2.2. Notifier: ideas generales

El siguiente módulo que se ejecuta en el ordenador del desarrollador es el Notifier. Este módulo es el que interpreta y filtra los datos generados por TempusFugit y los prepara para enviarlos al servidor.

El Notifier tiene un papel vital, ya que va a ser el encargado de asegurar el nivel de privacidad de los datos a enviar. En la actualidad no se encuentra implementado, pero en su diseño se deberán cumplir los siguientes requisitos:

- La privacidad será configurable por el usuario. Es decir, el cliente será quien decida qué información se envía.
- Las posibilidades de configuración de la privacidad serán muy flexibles. Así pues, la configuración más restringida sería filtrarlo absolutamente todo, mientras que la menos restringida sería enviar toda la información, incluyendo la indentificación del usuario.

- Será posible enviar la información de traza de manera anónima, aunque en cualquier caso el cliente tendrá un identificador único anónimo (tipo hash) que permitirá siempre seguir la actividad en el tiempo del desarrollador aunque no sepamos quién es.

Un requisito más que debe cumplir el Notifier es la *inteligencia* necesaria para configurar el filtrado. No se trata solamente de filtrar la actividad de ciertas aplicaciones que no se consideren parte de la actividad del desarrollador en el proyecto, sino también de filtrar parte de la actividad en algunas aplicaciones.

Por ejemplo, puede permitirse trazar la actividad realizada con el lector de correo siempre y cuando esté relacionada con el seguimiento y participación en una lista de correo (la del proyecto), eliminando la actividad en correos de índole personal. De la misma forma, en ocasiones se querrá facilitar la traza de la actividad en el navegador web, siempre que por ejemplo se trate de navegación o edición en páginas del gestor de fallos via web que se utilice en el proyecto (por ejemplo, Bugzilla).

3. Posibilidades

El software libre poco a poco va ganando interés, no solo entre los usuarios en general sino también en las empresas. Naturalmente, la comunidad investigadora también está muy interesada en los modelos de desarrollo que han producido tanta cantidad de software libre hoy día, prestando mucho interés en la comunidad GNOME, por su tamaño y los resultados ofrecidos hasta el momento. Hasta ahora se han encontrado interesantes resultados, como aquellos relacionados con unos patrones de crecimiento elevados y diferentes a los clasicos [7] o la baja densidad de fallos registrada en muchos casos [9].

Como hemos indicado, las empresas (y la administración pública) están muy interesadas en el fenómeno. Aunque su comportamiento en general no es uniforme: en ocasiones contratan gente para que continúen su ya iniciado trabajo en el desarrollo de ciertas aplicaciones, otras liberan como software libre productos que antes eran privativos, etc. Grandes empresas como Novell, Sun Microsystems o IBM están experimentando nuevos modelos, sin contar con las otras muchas Pymes que se están abriendo paso en el mercado gracias también al software libre.

A pesar del interés existente, el área de estimación de esfuerzo en el software libre no está muy estudiado. Y todos sabemos que las empresas necesitan buenos modelos de estimación en su toma de decisiones. Los modelos clásicos pueden sernos útiles pero, los datos que nos proporciona el proceso de desarrollo del software libre puede ayudarnos a hacerlos más precisos.

Dado que el trabajo voluntario es muy importante, creemos en la necesidad de poder conocer en muestras de desarrolladores, cómo invierten su tiempo en cada una de las tareas del proyecto, con el fin de compararlo con la productividad real (commits enviados a un CVS, mensajes enviados a las listas de correo de desarrollo, “bugs” gestionados en un Bugzilla, etc).

3.1. Fuentes de información medibles y su aplicación en las estimaciones de esfuerzo

Como hemos adelantado ya, en el software libre podemos contar con más fuentes de información que puede utilizarse para las mediciones relacionadas con el coste del proyecto. Tomemos, por ejemplo, las siguientes tres fuentes:

1. Datos desde un sistema de control de versiones. El código fuente, como sabemos, suele estar mantenido concurrentemente por los desarrolladores a través de un sistema de control de versiones, como CVS. Ello facilita muchísimo el trabajo de los desarrolladores pero, además, podemos analizar en profundidad la información que proporciona para cada fichero o módulo, por un lado, y cada desarrollador por otro, produciendo interesantes resultados [3] acerca de las transacciones que el CVS va dejando registradas.
2. Listas de correo. En el software libre, con frecuencia los desarrolladores trabajan en lugares geográficamente dispersos. Por tanto, muchas veces las reuniones de coordinación son sustituidas por discusiones en listas de correo. Los archivos de estas listas pueden ser analizados obteniendo también resultados interesantes acerca de la actividad en éstas [8].
3. Sistemas de gestión de erratas. Estos sistemas son usados ampliamente para almacenar los informes de error y sugerencias de los usuarios, pero en ellos los desarrolladores también aportan información de interés al indicar cuándo quedan resueltos los fallos, cómo han sido resueltos, qué parches se han utilizado, etc. Los propios usuarios del sistema pueden añadir comentarios, lo que con frecuencia es utilizado para ayudar al desarrollador a resolver el problema, incluso publicando *parches de código* a través del sistema.

Resulta evidente que los desarrolladores invierten tiempo en cada una de estas actividades. Idealmente, si el desarrollador no invirtiese más tiempo para el proyecto que en las actividades enumeradas anteriormente, podríamos tener el siguiente modelo de esfuerzo [2]:

$$esfuerzo = g(actividad) = a \cdot h(CVS) + b \cdot j(ML) + c \cdot k(BTS) + d$$

donde g, h, j y k son funciones a definir, así como los factores a, b, c y d. Por supuesto, ninguno de ellos está definido aun, pero precisamente la medición de actividades y la traza de tiempos dedicado a éstas nos puede ayudar a inferir estas incógnitas.

Por establecer una comparación, muchos modelos de estimación de esfuerzo clásicos sólo tienen en cuenta la actividad de producción de código, lo que podría representarse así:

$$esfuerzo = a \cdot h(CVS)$$

Resulta obvio, sin embargo, que en todos los proyectos se invierte tiempo en otras actividades que no son producción de código. Por tanto, todas aquellas que podamos medir deberían poder ayudarnos a mejorar esas estimaciones para proyectos futuros.

4. Conclusiones

En este artículo hemos presentado el *framework* DeTraS como conjunto de herramientas útiles para la traza de la actividad de desarrolladores. En primer lugar hemos justificado la necesidad de este tipo de trazado, para luego adentrarnos un poco más en la descripción de la herramienta y su arquitectura, describiendo técnicamente la parte desarrollada y en líneas generales el sistema completo. Finalmente hemos justificado desde un punto de vista de investigación los posibles resultados de la misma.

Es muy interesante apreciar que la disponibilidad de datos de productividad en el software libre y, con la ayuda de las herramientas presentadas aquí, la disponiblidad de datos de tiempo invertido en cada actividad, es posible mejorar la precisión con la que se calcula el coste de un proyecto. Esta mejora en la estimación de esfuerzos puede ser extremadamente útil para las empresas que desarrollan profesionalmente software libre y también para la comunidad investigadora, que actualmente carece de modelos adecuados de estimación para estas nuevas formas de desarrollar software.

Por todo ello, creemos que la realimentación de desarrolladores y responsables de proyecto de software libre, en el desarrollo de DeTraS, es muy importante. Esta herramienta aspira a ser útil en la investigación sobre el funcionamiento de las comunidades de desarrollo, lo cual se facilitará enormemente con el apoyo de estas comunidades.

Referencias

1. A. Albrecht. Measuring application development productivity. In I. B. M. Press, editor, *IBM Application Development Symp.*, pages 83–92, October 1979.
2. J. J. Amor, G. Robles, and J. M. Gonzalez-Barahona. Effort estimation by characterizing developer activity. In *8th Intl. Workshop on Software Engineering Economics*, pages 3–6, Shanghai, China, May 2006.
3. J. J. Amor, G. Robles, and J. M. González-Barahona. GNOME como caso de estudio de ingeniería de software libre. In *Actas de la II GUADEC Hispana*, pages 1–11, La Coruña, Spain, May 2005.
4. B. W. Boehm. *Software Engineering Economics*. Prentice Hall PTR, October 1981.
5. B. W. Boehm, E. Horowitz, R. Madachy, D. Reifer, B. K. Clark, B. Steece, W. A. Brown, S. Chulani, and C. Abts. *Software Cost Estimation with Cocomo II*. Prentice Hall PTR, January 2000.
6. D. M. Germán. The GNOME project: a case study of open source, global software development. *Journal of Software Process: Improvement and Practice*, 8(4):201–215, 2004.
7. M. W. Godfrey and Q. Tu. Evolution in Open Source software: A case study. In *Proceedings of the International Conference on Software Maintenance*, pages 131–142, San Jose, California, 2000.
8. I. Herraiz, J. J. Amor, and A.Ñavarro. Análisis de listas de correo en el software libre: un caso de estudio. In *Actas de las Jornadas Andaluzas de Software Libre*, pages 39–44, Córdoba, Spain, Oct. 2005.

9. A. Mockus, R. T. Fielding, and J. D. Herbsleb. Two case studies of Open Source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology*, 11(3):309–346, 2002.
10. G. Robles, S. Koch, and J. M. González-Barahona. Remote analysis and measurement of libre software systems by means of the CVSAnalY tool. In *Proceedings of the 2nd ICSE Workshop on Remote Analysis and Measurement of Software Systems (RAMSS)*, pages 51–56, Edinburg, Scotland, UK, 2004.