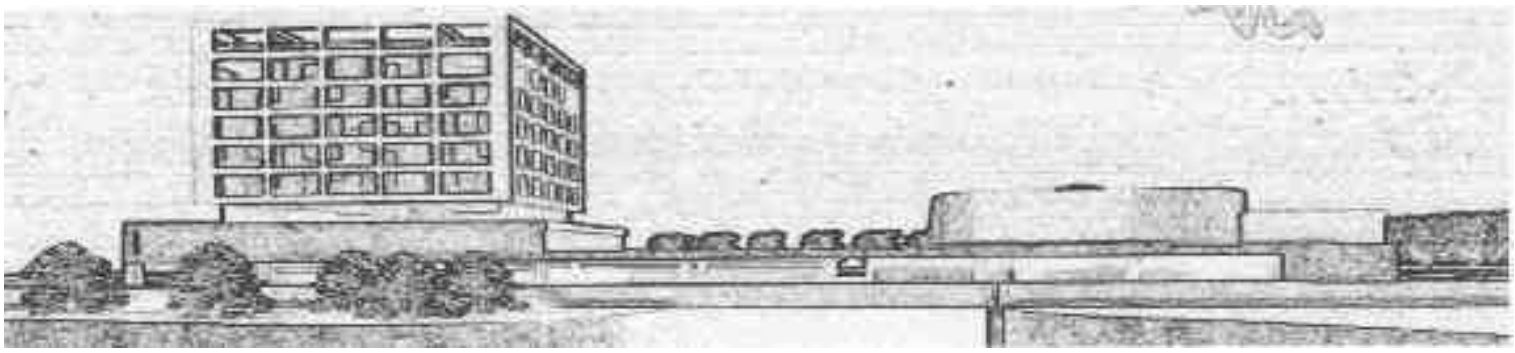
VOLUME VI, NUMBER 3



REPORTS ON SYSTEMS AND COMMUNICATIONS

CARACTERIZACIÓN DE LA ACTIVIDAD DE DESARROLLO EN CÓDIGO FUENTE. CASO DE ESTUDIO.

JUAN JOSÉ AMOR IGLESIAS

Móstoles (Madrid), Julio 2006
Depósito Legal: M-50653-2004
ISSN: 1698-7489

Caracterización de la actividad de desarrollo en código fuente. Caso de estudio.*

Juan José Amor Iglesias
GSyC, Universidad Rey Juan Carlos (Madrid, Spain)
jjamor_at_gsyc.escet.urjc.es
Junio de 2006

Resumen

En la caracterización de la actividad en la producción de código fuente de un proyecto software, es bastante habitual estudiar las transacciones del sistema de control de versiones utilizado. En este trabajo presentaremos una metodología de caracterización dividida en dos partes: por un lado, un análisis cuantitativo del repositorio de versiones de los proyectos, centrada principalmente en el estudio de las transacciones; y por otro lado, una metodología para clasificar los tipos de actividad en estos repositorios, basada en el análisis de las descripciones (comentarios) de la transacción. La metodología ha sido probada en un caso concreto, el cual sirve como caso de estudio (el repositorio CVS de FreeBSD), así como algunas ideas acerca de cómo el uso de esta técnica puede ayudar a la una mejor caracterización de la actividad de los desarrolladores de software y también, en la estimación de esfuerzos, entre otras líneas interesantes.

keywords: cvs, caracterización, bayes, clasificación aprendizaje

1. Introducción

El análisis de la actividad de los desarrolladores en los proyectos de software puede ser interesante por varios motivos, siendo la estimación de esfuerzos uno de los más claros casos de estudio. Medir la actividad puede ser difícil al no disponer de informes personales de actividad y dedicación u otra información similar, que en modelos de desarrollo cerrado están casi siempre disponibles. En el software libre, los proyectos suelen adolecer de no disponer de esos informes de dedicación pero, por suerte, se tienen otras fuentes de información que pueden utilizarse para inferir la actividad. Los sistemas de control de versiones, las listas de correo y los sistemas de seguimiento de fallos, por ejemplo, proporcionan gran cantidad de información acerca de la actividad de los desarrolladores en diferentes campos [1].

Estos datos están normalmente accesibles de manera pública y pueden ser recopilados y analizados con el fin de obtener una visión detallada de la actividad de los desarrolladores. Esto ha despertado interés en la comunidad investigadora. Por ejemplo, en el campo de la estimación de esfuerzo, en [5] se intentó aproximar el esfuerzo requerido en los cambios a partir del estudio de los datos de un repositorio de control de versiones.

Y es que, en el caso concreto de los sistemas de control de versiones, disponemos de la historia completa de modificaciones en cada fichero almacenado, incluyendo información acerca de líneas cambiadas, autor del cambio, fecha y hora y otras informaciones. Además, se almacena una descripción del cambio que el propio desarrollador redacta. Por ello, desde hace algunos años, algunos grupos de investigación en diversos campos han utilizado esta información, aunque pocas veces se ha utilizado la descripción del cambio debido, probablemente, a la complejidad del análisis del lenguaje natural.

Sin embargo, algunos autores han utilizado esta información para caracterizar sobre todo transacciones de código[1]. Así pues, encontramos que Mockus et al. [8] propusieron una metodología de clasificación de transacciones de código basada en el análisis de las descripciones. Estos autores realizaron una etapa de normalización del texto (esto es, convertirlo a minúsculas, eliminando signos de puntuación y además, las palabras comunes que no tienen valor semántico y obteniendo del resto de palabras sus raíces).

Por ejemplo, si consideramos el texto sin normalizar,

> Fixed: Remove obsoleted floppy partition from device name.

tras la etapa de normalización nos quedaría el siguiente,

> fix remove obsolete floppy partition device name

Posteriormente, estos autores realizaron análisis de frecuencia de palabras y clasificación de grupos de palabras con el fin de obtener una serie de reglas que permitan clasificar cada modificación en uno de los tres tipos de cambios propuestos en [11]: cambios correctivos (destinados a la corrección de fallos), adaptativos (destinados a la implementación de nuevas funcionalidades) y perfectivos (destinados a la mejora de la implementación actual, como puede ser la refactorización de código).

Por su parte, German [4] utilizó las descripciones de texto para analizar los cambios de corrección de fallos y de modificaciones en comentarios.

Aunque estas aproximaciones son buenos puntos de partida, no son suficientes para cubrir nuestro objetivo de realizar una clasificación de granularidad más fina, es decir, realizar una clasificación de la actividad de mantenimiento, no solo en las tres clases de transaccioens principales vistas más arriba, sino en otras subclases de las anteriores, deducibles a partir de la simple observación de unos cuantos proyectos de software libre.

En este trabajo, presentaremos una metodología que utiliza varias técnicas basadas en el análisis de las transacciones de los sistemas de control de versiones (como los clasificadores bayesianos) para obtener así una caracterización de la actividad en el repositorio de versiones que consta, entre otros, de un análisis del repositorio a nivel de grano grueso y otro de grano fino mediante una metodología de clasificación de transacciones de código a partir de las descripciones redactadas por los desarrolladores.

Las secciones de las que consta este artículo son las siguientes: en la siguiente sección se comienza introduciendo los atributos elegidos para clasificar las transacciones de código, como paso previo a la metodología que se describe en la siguiente sección. A continuación, se muestra en detalle la metodología aplicada a un caso de estudio, el sistema operativo FreeBSD, dividida en dos partes: la dedicada al estudio del repositorio y la dedicada a la clasificación asistida de las transacciones de código. La última sección servirá para emitir las conclusiones de este trabajo.

[1] Llamamos transacciones de código en un sistema de control de versiones, a un *commit* atómico, que cambia al menos un fichero de código fuente [4]. También se suelen denominar en la literatura registro de modificación (en respuesta a una petición de modificación o *modification request*).

2.. Atributos para caracterización de transacciones

Las transacciones del repositorio de control de versiones vienen caracterizadas por un conjunto de atributos, que en este trabajo se examinan de acuerdo con una metodología.

Estos atributos destacables son:

- Número de ficheros. Corresponde con el número de ficheros que se modifican en la transacción.
- Tamaño (número de líneas de código fuente o SLOC). Corresponde con el número de líneas de código que produce la transacción, para los ficheros modificados.
- Líneas añadidas. Corresponde con el número de líneas de código añadidas durante la transacción, en todos los ficheros modificados.
- Líneas borradas. Corresponde con el número de líneas de código eliminadas durante la transacción, en todos los ficheros modificados.
- Tipo predominante de fichero. Cada fichero se puede clasificar por un tipo (código, ficheros de traducción, etc). El tipo predominante corresponde con el tipo de fichero que más frecuentemente aparece en la transacción.
- Tipo de transacción. Corresponde con la clasificación que se describe a continuación. Esta clasificación se ha realizado buscando una significación relacionada con el esfuerzo que implica la transacción, es decir, diferentes tipos de transacciones se orientan hacia esfuerzos diferentes.

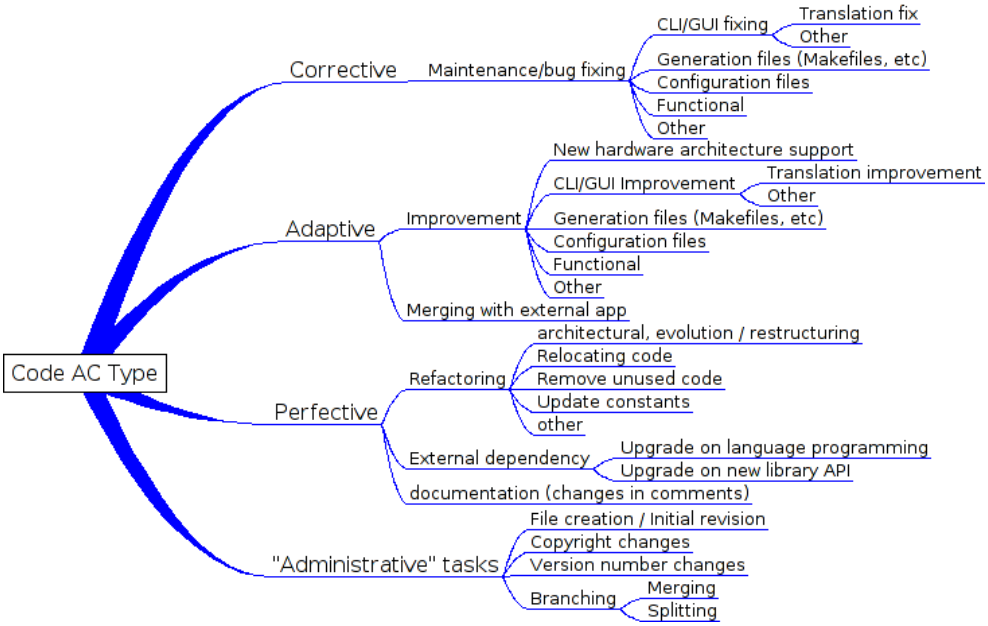


Figura 1. Esquema de clasificación de las transacciones de código

Como hemos indicado, parte de nuestro interés se centra en clasificar las transacciones, de acuerdo a una tipología. Para ello, debemos comenzar identificando las diferentes clases de transacciones con las que

vamos a trabajar. Para esta identificación, cuyo esquema se encuentra en la figura 1, hemos partido de los tipos de transacción ya mencionados como 'clásicos' (correspondientes a las actividades de mantenimiento correctivas, adaptativas y perfectivas) [11] a los que hemos añadido una cuarta clase, llamada administrativa. Esta clase corresponde con transacciones no relacionadas con las anteriores y no siempre realizadas por desarrolladores, sino por administradores del sistema de control de versiones. Puede verse a qué tipo de actividades nos referimos al mirar la figura anterior: nos referimos a cambios relacionados con la creación de ficheros, los cambios en el copyright o las operaciones con ramas.

Con esta clasificación en mente, hemos examinado sistemas de control de versiones (CVS[2]) de varios proyectos de software libre y hemos identificado así las diferentes subclases para cada una de las clases principales. En esta clasificación, además, se ha mantenido en mente que se buscaban tipos que tuvieran, al menos intuitivamente, efecto en el coste final de realizar los cambios de la transacción por parte del desarrollador, debido a que el objetivo final de nuestros estudios es la estimación de esfuerzo a partir de la caracterización de actividades. Por ejemplo, podemos presuponer que transacciones correctivas relacionadas con las cadenas de texto de las interfaces de usuario ('CLI/GUI fixing') son claramente menos costosas que cualquiera otra. Y que los cambios funcionales, usualmente serán más costosas que el resto. Por otro lado, en la clasificación elegida se ha buscado que los tipos sean lo suficientemente genéricos como para estar en cualquier tipo de proyecto de software y no solo en algunos específicos como pudieran ser sistemas operativos.

En definitiva, hemos identificado, dentro de las cuatro clases principales, las subclases que se enumeran a continuación:

- Las transacciones correctivas han sido clasificadas según la clase de corrección identificada: CLI/GUI[3] (correcciones que afectan a las cadenas de texto de la interfaz de usuario y no a la funcionalidad de la aplicación), ficheros de generación (correcciones en *makefiles* y otros archivos usados durante la compilación y generación de ejecutables), ficheros de configuración, funcional (aquellos que corresponden con corrección de fallos de funcionalidad) y otros (por ejemplo, una corrección ortográfica en un comentario).

- Las transacciones adaptativas se clasifican en los tipos *improvement* (mejoras) y *merging* (mejoras producidas por la incorporación de código externo). A su vez, dentro del primer tipo se encuentran subtipos parecidos a los vistos en las correctivas.

- Las perfectivas han sido divididas en refactorizaciones, adaptación a dependencia externa (por ejemplo, por una nueva versión de una biblioteca de la que depende el código) y cambios perfectivos en los comentarios del código. Las refactorizaciones, a su vez, se dividen en varios subtipos correspondientes a varias clases de refactorizaciones que se pueden identificar.

- Las administrativas corresponden con la creación de ficheros (en ocasiones el comentario del texto es simplemente del tipo 'initial version'), cambios en el copyright o la licencia, ajustes en el número de versiones y operaciones con ramas (por ejemplo, ante la operación de introducir en la rama principal los cambios de una rama de desarrollo).

[2] Aunque CVS empieza a ser sustituido por otros sistemas, todavía es el sistema de gestión de versiones más usado en los proyectos de software libre.

[3] CLI corresponde con Command Line Interface, es decir, interfaz de usuario en modo comando; y GUI, Graphic User Interface o interfaz gráfica

Es importante observar que la clasificación realizada no tiene por qué ser completa y puede ser mejorada mediante el estudio de otros proyectos. Sin embargo, para los propósitos de este trabajo sí la consideramos suficiente.

3. Metodología

La metodología de análisis tiene dos partes principales:

1. En primer lugar, utilizando la herramienta CVSAnaly [10, 6], se obtiene información sobre todo el repositorio CVS a analizar. Entre otros resultados, esta herramienta nos proporciona todas las transacciones del repositorio, con toda la información relacionada, incluyendo las descripciones que el desarrollador redactó para cada transacción.

2. En segundo lugar, se realiza una clasificación de la base datos de transacciones, a partir del texto de las descripciones, tomando como base el clasificador *Naive Bayes*. El resultado de la clasificación automática consiste en asignar a cada transacción, una de los tipos de transacción con una cierta probabilidad.

CVSAnaly funciona analizando el *log* de un repositorio CVS, agrupando los *commits* en transacciones (mediante la utilización del algoritmo descrito en [4]). Para cada transacción, se guarda información relevante como el autor de la misma (commiter) o la descripción realizada por el desarrollador. Asímismo, para cada *commit* de fichero individual se guarda información relevante, como el número de versión, las líneas añadidas o borradas o el instante del envío. El resultado es una base de datos que describe exhaustivamente el repositorio de CVS, en un formato estructurado.

Sobre esta base de datos, se realizan una serie de análisis que ayudan a la caracterización e interpretación de los datos del repositorio en cuestión.

En esta primera parte, se trata bastante con la discriminación de tipos de ficheros. Aunque los repositorios de control de versiones se diseñaron en un principio para el mantenimiento de código fuente, lo cierto es que cada vez se utilizan más para almacenar otros tipos de ficheros. Los tipos elegidos corresponden con documentación, imágenes, traducción (i18n), interfaces de usuario (UI), multimedia y por último, código fuente. Para estos últimos, se distinguen algunos subtipos: ficheros de código que son parte de la aplicación (código), ficheros que sirven para ejecutar el proceso de construcción de ejecutables (generación) y ficheros de documentación que están ligados tanto al código como al proceso de construcción (build-doc).

La detección de los tipos la realiza la herramienta CVSAnaly, mediante unas heurísticas basadas en algunos casos en la extensión del nombre de los ficheros y en otros en el nombre completo. En el cuadro 1 se detallan algunos patrones de nombre y extensión de fichero utilizados en la detección. No siempre son conocidos los tipos de fichero, por lo que hay que considerar en los estudios, las proporciones de ficheros que no hayan podido clasificarse.

Por otro lado, el análisis se hace en su mayor parte tomando como unidad la transacción del repositorio. Desafortunadamente, los sistemas de control de versiones como CVS o Subversion no almacenan información acerca de las transacciones, por lo que es necesario reconstruirlas. La reconstrucción consiste en conseguir agrupar los commits que fueron hechos casi con seguridad en una operación simultánea. El algoritmo utilizado es el descrito por German [4], por el que dos commits consecutivos se agruparán en la misma transacción si:

- Ambos commits tienen el mismo autor y el mismo comentario en texto.

Tipo de fichero	Patrón
Documentación	*.html *.txt *.ps *.tex *.sgml
Imágenes	*.png *.jpg *.jpeg *.bmp *.gif
i18n	*.po *.pot *.mo *.charset
Interfaz gráfica	*.desktop *.ui *.xpm *.theme
Código	*.c *.h *.cc *.pl *.java *.s *.ada
Generación	configure.* makefile.* *.make
Devel-doc	readme* changelog* todo* hacking*

Cuadro 1. Algunos patrones usados en la detección del tipo de fichero.

- Ambos commits están separados temporalmente en un valor máximo dado.

El algoritmo va recorriendo la línea de tiempo de commits y agrupándolos en una transacción lo más grande posible, a modo de ventana deslizante, pero el tamaño máximo vendrá limitado por otro parámetro temporal, correspondiente a la diferencia máxima en tiempo entre el primer y el último commit de la transacción.

El algoritmo busca agrupar commits enviados simultáneamente por el autor pero considerando los retrasos entre ellos propios del retardo del envío de cada fichero por una línea de baja velocidad o cualquier incidencia típica relacionada con la red utilizada, que hace que una transacción en teoría instantánea, se demore con frecuencia en varios segundos.

Una vez agrupadas las transacciones, obtener los diferentes datos de la transacción es una mera cuestión de números (tamaño en líneas de código, añadidas, borradas, etc) en casi todos los casos. El único parámetro un poco especial es el tipo predominante: como hemos dicho anteriormente, cada fichero tiene un tipo (normalmente elegido a partir de su nombre o su extensión). El tipo predominante ayuda a caracterizar la transacción y el criterio para definirlo es, simplemente, que se corresponde con el tipo de fichero que más veces aparece (en número de ficheros) en la transacción. Y en caso de que varios tipos tengan el mismo número de ficheros, se decidirá el tipo al azar entre los que comparten el máximo número de apariciones.

Para la clasificación de las transacciones de código, utilizamos el algoritmo *naive Bayes* [3], el método de clasificación automática de textos mediante aprendizaje más conocido. Este método se basa en el teorema de Bayes, y aunque requiere que el conjunto de atributos sea condicionalmente independiente para cada clase, normalmente funciona muy bien en casos reales, aunque no se cumplan del todo estas condiciones. De hecho, se utiliza ampliamente para clasificar automáticamente correo no solicitado (*spam* [2]). Nosotros utilizaremos una herramienta que lo implementa, llamada “conjunto de herramientas Bow” (Bow toolkit [7]), libremente disponible. Aunque esta herramienta implementa varios algoritmos de clasificación de textos por aprendizaje, nosotros utilizaremos únicamente el método bayesiano.

Los métodos con aprendizaje requieren que, en una primera fase, se prepare una colección de registros clasificados por un experto. Posteriormente, esta colección se *alimenta* al clasificador con el fin de que *aprenda* y pueda clasificar el resto de los registros automáticamente. En realidad, la metodología no es tan simple: después de un primer intento de clasificación automática, el experto debe revisar los resultados y verificar la corrección de los mismos, cambiando o mejorando la tabla de aprendizaje con el fin de mejorar los resultados obtenidos automáticamente.

En las siguientes secciones describiremos el caso de estudio elegido y cómo se ha aplicado la metodología descrita aquí, al sistema elegido.

4. Caso de estudio: FreeBSD

4.1. Introducción a FreeBSD

FreeBSD es un sistema operativo libre que nació para ordenadores personales basados en arquitectura x86, aunque actualmente se encuentra optimizado para las versiones más modernas de la arquitectura así como portado a nueve arquitecturas más.

FreeBSD procede de 4.4BSD-Lite, creado en la Universidad de California (Berkeley). Esta versión de BSD también ha sido base de otros sistemas operativos libres: OpenBSD y NetBSD, que actualmente se desarrollan de manera independiente.

El trabajo de desarrollo en FreeBSD se divide en dos áreas principales: el sistema operativo propiamente dicho y los denominados *transportes* o *ports*. Estos últimos corresponden al trabajo de la comunidad de desarrolladores para preparar el código fuente de otros productos externos al proyecto, con el fin de poderlos compilar y empaquetar fácilmente en FreeBSD. Este trabajo, sin embargo, se centra en el estudio del sistema operativo FreeBSD.

El sistema operativo, a su vez se encuentra dividido en dos partes principales: el núcleo y las herramientas del espacio de usuario, conocidas como *userland*. Este conjunto tiene en la actualidad de aproximadamente 6 millones de SLOCs[4], de los que unos 2 millones corresponden al núcleo del sistema.

Al tratarse de un sistema operativo descendiente de Unix, se encuentra desarrollado en lenguaje C, principalmente, suponiendo el 90 % del código. El resto se reparte principalmente en guiones de *shell* y la presencia testimonial de más de 15 lenguajes de programación.

Aunque el tamaño actual es, como hemos dicho, de unos 6 millones de líneas, el análisis realizado con la herramienta CVSAnaly reporta un trabajo entre líneas cambiadas, añadidas y borradas mucho mayor, correspondiente a las más de 650000 versiones almacenadas.

Es interesante reseñar cómo se organizan las versiones en FreeBSD. Aunque el número de etiquetas y ramas es mucho mayor, esbozaremos aquí las ramas principales que permiten identificar una versión de desarrollo o una versión estable.

En todo momento, existe una rama llamada FreeBSD-CURRENT (HEAD). Realizando un checkout de esta rama obtendremos la última versión de desarrollo, con cambios pendientes, funcionalidades aun experimentales y otros que, finalmente, pueden aparecer o no publicados en una futura versión estable. Normalmente HEAD se asocia con los números de versión más actuales (altos) de los ficheros.

En la rama FreeBSD-STABLE encontramos código revisado que normalmente ha pasado un tiempo en CURRENT.

Desde las versiones 4.x, se introdujeron las llamadas ramas de publicación (“release branches”), que se utilizan para varias cosas, tales como crear versiones estables (“release engineering”), así como agrupar parches de seguridad y otros tipos de fallos. De manera resumida, el esquema del árbol de ramas que estamos comentando es el de la figura 2.

De este modo, periódicamente, el equipo de desarrollo de FreeBSD decide crear una rama de publicación (por ejemplo, *RELENG_6*), en la que se congela la funcionalidad actual de HEAD. A partir de ese momento, en esa rama se crean sub-ramas destinadas a revisiones periódicas, que contienen únicamente fallos críticos. De este modo, las versiones *RELENG_6_0*, *RELENG_6_1*, etcétera; se crean para publicar una versión que no añade funcionalidad pero corrige un conjunto de fallos de seguridad y otros fallos críticos encontrados hasta ese momento.

[4]SLOCs son líneas *físicas* de código fuente, es decir, las líneas de programa sin contar líneas en blanco o que solo contengan comentarios

```
HEAD              CVS HEAD
  RELENG_6        6-STABLE devel branch
    RELENG_6_0    6.0 eng branch
  RELENG_5        5-STABLE devel branch
    RELENG_5_0    5.0 eng branch
    RELENG_5_1    5.1 eng branch
    ....
    RELENG_5_5    5.5 eng branch
  RELENG_4        4-STABLE devel branch
    RELENG_4_11   4.11 eng branch
    ...
```

Figura 2. Ramas más importantes en el CVS de FreeBSD.

Estas ramas de mantenimiento siguen activas hoy en día incluso en las versiones antiguas. Así, es de esperar que pronto se publique la versión *RELENG_5_6*, para aquellos usuarios que prefieran mantenerse en las funcionalidades y requisitos de la versión 5, mientras que siguen apareciendo revisiones de la versión 6 y, en HEAD, se sigue desarrollando lo que en algún momento se congelará creando por primera vez la versión 7 (rama *RELENG_7*).

4.2. Análisis general del repositorio

Volviendo a los números del repositorio, en el cuadro 2 se muestra un resumen de las características del repositorio analizado. En el análisis se observa que se ha analizado solo el módulo *src*. Este módulo es precisamente el que corresponde con la elección de analizar el núcleo y el *userland*. Otros módulos se encuentran dedicados a los *transportes* y a otras herramientas que no forman parte esencial del sistema operativo.

En esta tabla, además de los datos generales como el número de desarrolladores involucrados en las modificaciones (commiters), las modificaciones en sí (commits), etc; se han incluido también las transacciones o commits atómicos.

En el cuadro 3 mostramos las distribuciones de los ficheros, tipos de cambio y tipos de transacción, de acuerdo a como los proporciona la herramienta CVSAnaly. El tipo de transacción ha sido determinado eligiendo el tipo más frecuente en todos los cambios (commits) que forman parte de la transacción. En caso de que hubiera varios tipos con la máxima frecuencia, se elige uno de ellos al azar. De esta table podemos tener una idea de las características del proyecto estudiado.

Vemos, por ejemplo, que la mayor parte de las transacciones son de código, lo que en buena medida justifica un posterior estudio de estas transacciones, uno de los objetivos de este trabajo. En un proyecto de desarrollo típico, es normal que las transacciones de código sean las que tengan mayor presencia. Véase, por ejemplo, el caso de estudio presentado en [9]. Sin embargo, en nuestro caso nos encontramos que la proporción de código es mayor aun, lo que enriquece las posibilidades de un posterior análisis específico de este tipo de transacciones.

Otra observación interesante es que, a diferencia de otros proyectos, apenas hay trabajo de internacionalización o traducción a otros idiomas (i18n). Del mismo modo, hay muy poca actividad con ficheros de

Número de módulos	1 (src)
Número de commiters	440
Número de commits	626228
Número de ficheros	61663
Ficheros de código	36569
Fecha primer commit	1/1/1995
Fecha último commit	17/6/2006
Número de transacciones	153395
Transacciones de código	103326
Líneas añadidas	21770000
Líneas borradas	10353000

Cuadro 2. Resumen de características del repositorio CVS de FreeBSD.

Tipo	Ficheros	%	Commits	%	Transacciones	%
Total	61663	100	626228	100	153395	100
Código	36569	59.3	407222	65	96406	62.8
Desconocido	13866	22.4	123743	19.7	29603	19.3
Generación	6482	10.5	62092	9.9	18346	11.9
Devel-doc	1997	3.2	13796	2.2	2696	1.7
Documentación	1950	3.1	15145	2.4	5330	3.4
Interfaz gráfica	694	1.1	3928	0.6	1004	0.6
Imágenes	94	0.1	238	0	10	0
i18n	11	0	64	0	0	0

Cuadro 3. Tipos de ficheros, commits y transacciones en FreeBSD.

imágenes o interfaz gráfica. Todo ello es bastante propio de un sistema operativo, aunque el trabajo de traducción a otros idiomas pueda ir ganando en importancia en el futuro.

Por último, vemos la nada despreciable proporción de ficheros de tipo desconocido. Tras un examen visual de los datos observamos que en buena proporción corresponden con las *páginas de manual* de FreeBSD (ficheros con extensión numérica, que CVSAnaly no identifica correctamente como documentación), apareciendo también muchos ficheros sin extensión o con extensiones difícilmente identificables, donde se mezclan anotaciones (ficheros con documentos breves), imágenes de firmware, etc.

Podemos mirar un poco más las transacciones analizadas en el proyecto FreeBSD. Observando el cuadro 4 vemos que, en primer lugar, la proporción de transacciones de un solo fichero es bastante elevada, en torno al 59 %. Esto muestra que FreeBSD no es un proyecto en el que, por cada transacción, se documente el cambio en un fichero *ChangeLog* como sí sucede en otros proyectos[4].

Es interesante comprobar que las transacciones con un alto número de commits no es despreciable. Es decir, aunque el número de transacciones con más de, digamos, 500 ficheros es muy bajo, sí vemos que algo más de uno de cada diez commits corresponde a una transacción de esas características. Un examen de transacciones de este tipo escogidas al azar nos muestra que este tipo de transacciones suele tener un significado administrativo: recuperación de versiones anteriores decididas por un coordinador, revisiones de texto de copyright, operaciones con ramas (mezclas). También se observan transacciones de este tipo cuando se incorpora código de otro proyecto, como puede ser una nueva revisión del compilador estándar

# ficheros	transacciones	%	commits	%
1	90242	58.8	90242	14.4
2	28334	18.5	56668	9
3	9998	6.5	29994	4.8
4	6040	3.9	24160	3.9
5	3335	2.2	16675	2.7
>10	7388	4.8	376055	60.1
>25	2970	1.9	306775	49
>50	1503	1	255134	40.7
>100	734	0.4	200592	32
>500	94	0.1	71131	11.4
>1000	16	0	20600	3.3

Cuadro 4. Distribución del número de ficheros por cada transacción

gcc (que mantiene el proyecto GNU) o algunas partes mantenidas en otro proyecto BSD, como OpenSSH (que mantiene OpenBSD).

Predominancia	transacciones	%
Todas	153395	100.0
=100 %	133141	86.8
<90 %	18974	12.4
<80 %	16130	10.5
<60 %	9623	6.3
<50 %	1517	1

Cuadro 5. Predominancia de los tipos de ficheros en las transacciones.

Como indicábamos antes, cada transacción se ha clasificado según el tipo de fichero más frecuente que aparece entre los ficheros de dicha transacción. Con el fin de ver si esta clasificación es suficientemente útil, revisamos la proporción de ficheros coincidentes con el tipo predominante, en cada transacción. El resultado del análisis se resume en el cuadro 5. En ella podemos ver que la coincidencia es total en casi un 87 % de los casos, y que es muy buena en un 90 % de los casos (si consideramos que una coincidencia superior en el 80 % de los ficheros de la transacción es una coincidencia *muy buena*).

En cualquier caso, la tabla anterior incluye las transacciones con pocos ficheros, que como hemos visto unos párrafos más arriba, son bastante abundantes. Por eso es necesario considerar este análisis pero omitiendo las transacciones de pocos ficheros. En el cuadro 6 vemos los resultados considerando, en varias columnas, las transacciones de más de 5 ficheros y las de más de 10. Vemos que, en estos casos, la coincidencia del tipo predominante con la totalidad de los ficheros de la transacción desciende hasta un 51 y un 43 % respectivamente, tan solo el 11 % de las transacciones tengan una coincidencia de tipos con el predominante inferior al 60 %. La conclusión que podemos extraer de todo esto es que el uso del tipo de fichero predominante para clasificar las transacciones debe hacerse con cuidado.

Predominancia	transacciones (>5)	% (>5)	transac. (>10)	% (>10)
Todas	15446	100.0	7388	100.0
=100 %	7951	51.5	3195	43.2
<90 %	6215	40.2	3015	40.8
<80 %	4068	26.3	2055	27.8
<60 %	1725	11.2	860	11.6
<50 %	724	4.7	410	5.5

Cuadro 6. Predominancia de los tipos de ficheros en las transacciones con más de 5 o 50 ficheros.

4.3. Clasificación automática de las transacciones de código

4.3.1. Preparación del conjunto de aprendizaje

Para realizar el conjunto de transacciones para el aprendizaje, se ha realizado una extracción de todas las transacciones de código de CVS en el proyecto objeto del caso de estudio. Para desarrollar la metodología hemos utilizado el repositorio CVS completo del proyecto FreeBSD. En el cuadro 2 se muestra un resumen de las características del repositorio analizado.

Para realizar el conjunto de aprendizaje, se han buscado casos correspondientes a cada uno de los subtipos de la figura 1. La forma de buscarlos ha sido la siguiente, para cada tipo:

1. Se obtienen palabras clave que el experto considere que pueden ser habituales en las descripciones para esos tipos de transacción.
2. Se hacen búsquedas al azar en la base de datos de transacciones que contengan esas palabras.
   - En primer lugar, que contengan *todas* las palabras.
   - En segundo lugar, y si se considera necesario, que contengan solo *algunas* de las palabras clave.
3. Mediante un editor web de base de datos, se examinan aleatoriamente las transacciones encontradas y se verifica si se pueden clasificar en el subtipo buscado.
4. Si es así, la clasificamos en ese subtipo. Si no, la clasificamos en otro de los subtipos si el texto no nos resulta ambiguo.
5. Del examen de estas transacciones pueden descubrirse otras palabras que pueden ser habituales en el subtipo buscado, de manera que las anotamos y realizamos nuevas búsquedas.

Por ejemplo, para buscar transacciones de los tipos *CLI/GUI fixing*, se tomaron inicialmente las palabras *fix, string*. El examen de varias transacciones nos proporcionó palabras como *user interface* o *comment* y otras que nos sirvieron para encontrar nuevos casos clasificables como *CLI/GUI fixing*. Al mismo tiempo, otros muchos registros examinados en ese momento pudieron ser clasificados ya en otros grupos con mayor presencia, especialmente los cambios perfectivos de refactorización, las correcciones de fallos funcionales y las mejoras también funcionales. Aunque obviamente, en el ejemplo concreto mostrado, también aparecieron casos de *CLI/GUI improvement*, tanto para los cambios de traducciones como los que no lo son.

Tras la aplicación de esta metodología, se ha podido construir una tabla de aprendizaje formada por 568 transacciones clasificadas por el experto.

Hay que tener en cuenta que algunas clases tienen muy pocos ejemplos encontrados en esta primera fase. En algún caso se debe a que los textos son prácticamente idénticos en todas las transacciones de esa clase y por tanto no sirve de nada aumentar el número de casos de aprendizaje. Es el caso de la clase *4.1*, en la que la mayoría de las descripciones son simples textos indicando simplemente *initial version*.

En otros casos no hemos podido encontrar más ejemplos de los que se han incluido en la tabla. Pensamos que en otros casos de estudio podrían aparecer más casos de estos tipos.

4.3.2. Comportamiento y mejora del conjunto de aprendizaje

Una vez realizado el conjunto de aprendizaje, mediante unos scripts preparamos los ficheros de entrada para alimentar al programa principal de la biblioteca Bow, *rainbow*.

Una vez alimentado este programa, realizamos una primera prueba consistente en clasificar automáticamente todas las transacciones de FreeBSD y proceder a analizar los resultados. En primer lugar, nos interesará conocer qué nivel de confianza otorga el propio algoritmo bayesiano que la biblioteca Bow implementa, a la clasificación realizada.

Así pues, observamos la probabilidad de acierto que el clasificador bayesiano otorga a las transacciones analizadas, y vemos que la probabilidad media para todas las transacciones es de alrededor del 52 %. Esta probabilidad es considerada baja, y sugiere realizar un análisis adicional con el fin de mejorar el comportamiento del algoritmo.

La técnica elegida ha sido elegir, en primer lugar, un conjunto adicional aleatorio de 100 transacciones, que nuevamente son clasificadas a mano. El total de transacciones, 568, se detalla en el cuadro 7. Con estas transacciones, la probabilidad de acierto proporcionada por el algoritmo bayesiano sube hasta un 65 %.

Tras este paso, se procede a clasificar automáticamente las transacciones de las ramas detalladas en la figura 2, dado que se pretenden obtener resultados aceptables para la clasificación de las transacciones de las ramas. La clasificación de las ramas vuelve a arrojar un resultado que consideramos pobre: la probabilidad media de acierto en cada rama sigue siendo tan solo ligeramente superior al 50 %.

La técnica elegida en este caso ha sido elegir un conjunto de transacciones con una clasificación de probabilidad baja (inferior al 40 %) y trabajar con él. Tras un examen de estas transacciones se ha comprobado que en muchos casos el problema es la ambigüedad de los comentarios de la transacción, o la difícil correspondencia con una de las clases establecidas, lo que causa para el clasificador bayesiano la misma dificultad que un clasificador *humano*. Concretando, se han localizado los siguientes tipos de problemas de clasificación:

- **Nuevas palabras clave.** De vez en cuando aparecen textos que no han sido bien clasificados por la biblioteca Bow, debido a que contienen palabras clave que no han sido utilizadas en el conjunto de aprendizaje, pero que pueden corresponderle sin duda alguna de las clases de transacciones propuestas.

- **Ambigüedad**. En ocasiones aparecen textos cuyo significado hacen muy difícil su clasificación incluso al experto. Por ejemplo, el texto *120 seconds is not 3 minutes* no contiene ninguna palabra clave que ayude a decidir y puede pertenecer a varias clases: podría tratarse de una corrección de fallo, aunque también de mejora de prestaciones, etc.

- **Agrupación de varios tipos de cambios en una sola transacción**. En ocasiones, encontramos transacciones que correspondían a varios cambios de diferentes tipos, que aparecen de esta forma documentados en el comentario. Por ejemplo, en el siguiente: *Add an #ifdef.d non-blocking version of*

Clase	Transacciones
1.1.1.1	9
1.1.1.2	17
1.1.2	16
1.1.3	11
1.1.4	61
1.1.5	16
2.1.1	14
2.1.2.1	17
2.1.2.2	6
2.1.3	13
2.1.4	29
2.1.5	77
2.1.6	13
2.2	16
3.1.1	28
3.1.2	14
3.1.3	26
3.1.4	10
3.1.5	85
3.2.1	6
3.2.2	5
3.3	16
4.1	2
4.2	10
4.3	15
4.4.1	32
4.4.2	3

Cuadro 7. Estadísticas de la tabla de aprendizaje.

the test. Update copyright encontramos un posible cambio adaptativo junto con cambio administrativo (licencias o derechos de copia).

- **Referencia a clases inexistentes.** En muy contadas ocasiones encontramos transacciones que no corresponden a ninguna de las clases establecidas, sino que sugieren una nueva. Por ejemplo, alguna transacción aparecía documentada con el texto *Change branch name* y correspondería a una nueva clase administrativa correspondiente al cambio de nombre de una rama.

Aunque la ambigüedad y los problemas de clasificación merecen un estudio más profundo, en general vimos que los textos pobremente clasificados corresponden casi siempre a cambios que incluyen múltiples clases (la transacción contiene varios tipos de cambio), y a textos fácilmente clasificables por el experto pero cuyas palabras clave no fueron incluidas en el conjunto de aprendizaje original.

Por ello, resulta interesante volver a clasificar a mano un subconjunto elegido al azar de transacciones pobremente clasificadas, eligiendo textos no ambiguos claramente clasificables en una de las clases propuestas

(y que contendrán, por tanto, nuevas palabras clave).

El siguiente paso ha sido, por tanto, realizar un nuevo conjunto de aprendizaje y unirlo al anterior, para luego volver a comprobar el comportamiento de *rainbow*.

Este conjunto adicional de aprendizaje consta de 96 textos, distribuidos en clases según el cuadro 8.

Clase	Transacciones
1.1.1.2	3
1.1.2	1
1.1.4	22
1.1.5	3
2.1.1	1
2.1.2.1	1
2.1.3	1
2.1.4	3
2.1.5	8
2.2	1
3.1.1	1
3.1.3	5
3.1.4	1
3.1.5	29
3.3	2
4.1	2
4.2	1
4.3	1
4.4.1	10

Cuadro 8. Estadísticas de la tabla adicional de aprendizaje.

Tras aplicar esta segunda tabla de aprendizaje, la probabilidad media de acierto sube al 65 % en cada una de las ramas, lo que hacen interesantes unas pruebas de validación.

La validación que realizaremos será simple: consiste en escoger al azar un conjunto de transacciones clasificadas automáticamente y verificar la tasa de acierto, en cada una de las ramas. Esta muestra consta de 30 transacciones por rama. El resultado obtenido ha sido de aciertos en aproximadamente el 70 % de los casos, en cada rama.

4.3.3. Aplicación del clasificador bayesiano al caso de estudio. Observaciones

Una vez elegido el conjunto de aprendizaje definitivo, y comprobada su eficacia sobre nuestro caso de estudio, podemos proceder realizar observaciones sobre su aplicación.

Para una primera visión, clasificamos todo el repositorio. Los resultados se muestran resumidos en el cuadro 9. El resumen consiste en mostrar únicamente la clasificación de segundo nivel de la figura 1, además de una línea con el total de transacciones clasificadas. En la tabla hemos incluido los resultados teniendo en cuenta que la clasificación tenga una probabilidad de acierto superior al 70 % o no. En ambos casos vemos que la actividad de refactorización es la más frecuente, seguida de la actividad de implementación de funcionalidades nuevas y la corrección de fallos. Otras clases tienen una presencia muy baja, destacando sobre todo las operaciones de cambio de notas de copyright y la de gestión de ramas. Observando la tabla

de transacciones con probabilidad de acierto mayor al 70 %, vemos que los porcentajes son parecidos en general, lo cual da una idea de una distribución bastante uniforme por tipo, de las transacciones con mayor probabilidad de acierto.

Clase	Transacciones	Porcentaje (%)	Transacciones (>70 %)	Porcentaje (>70 %)
–	103326	100	42451	100
Corrective/Bug Fixing	20280	19.6	6410	15.1
Adaptive/Improvement	27038	26.2	8660	20.4
Adaptive/Merging	415	0.4	41	0.1
Perfective/Refactoring	51306	49.7	17193	40.5
Perfective/Ext. Depend.	137	0.1	14	0
Perfective/Comments	147	0.1	14	0
Administrative/File Creation	1282	1.2	218	0.5
Administrative/Copyright	1357	1.3	320	0.8
Administrative/Version Numbers	132	0.1	78	0.2
Administrative/Branching	1232	1.2	474	1.1

Cuadro 9. Clasificación automática de las transacciones de FreeBSD.

Abundando en el juego que nos da este tipo de análisis, hemos realizado clasificaciones independientes sobre las ramas más importantes de FreeBSD, es decir, las que vimos en la figura 2, creando para cada una de esas ramas una base de datos diferente. En el cuadro 10 se incluyen los datos generales de las ramas analizadas. Como ya indicamos anteriormente, las ramas elegidas no son las únicas, motivo por el que la suma de transacciones de todas las ramas es mucho menor que las transacciones totales.

Rama	Transacciones de código	Commits	Commiters
–	103326	626228	440
HEAD	9052	36404	290
RELENG 6.1	101	175	37
RELENG 6.0	89	229	34
RELENG 6	2417	7565	142
RELENG 5.5	19	23	5
RELENG 5.4	140	285	36
RELENG 5.3	94	270	19
RELENG 5.2	167	318	38
RELENG 5.1	65	141	22
RELENG 5	231	12899	164
RELENG 4.11	96	249	28
RELENG 4.10	74	198	18
RELENG 4	13742	57358	263
RELENG 3	3530	18813	140

Cuadro 10. Datos generales de las ramas analizadas de FreeBSD.

Aplicando el conjunto de aprendizaje definitivo, se ha procedido a clasificar cada una de las ramas, obteniendo resultados interesantes en general. En el cuadro 11 se muestra el reparto de los tipos de transacción para tres ramas diferentes: la rama HEAD, la rama RELENG_6 y la rama RELENG_6_1, a modo de resumen de todos los datos estudiados. Esta tabla contiene solamente clasificaciones con probabilidad de acierto mayor al 70 %.

	HEAD	RELENG_6	RELENG_6_1
Corrective/Bug fix	25,87	59,04	88,05
Adaptive/Improvement	25,17	16,63	7,46
Adaptive/Merging	0,32	1,07	0
Perfective/Refactoring	47,00	22,96	2,99
Perfective/Ext. Dep.	1,54	0,08	1,49
Perfective/Doc.	0	0	0
Admin/File C.	0,08	0,23	0
Admin/Copyright	0	0	0
Admin/Versions	0	0	0
Admin/Branching	0	0	0

Cuadro 11. Clasificación en tres ramas (con probabilidad de éxito superior a 70 % según el clasificador bayesiano).

Como vemos en esta tabla, alrededor del 25 % de las transacciones en la rama HEAD son del tipo correctivo (corrección de fallos), mientras que el 25 % ha sido clasificado como adaptativo (implementación de nuevas características) y casi un 50 % de actividad perfectiva (refactorización). Tan solo un 2 % cae en otros tipo de clasificaciones. Vemos que, al haber en general un número de transacciones tan bajo de tipo administrativo, la clasificación se vuelve difícil, por lo que en las ramas es muy difícil obtener resultados interesantes.

Como hemos indicado, a modo de comparación hemos incluido los datos arrojados por el análisis de dos ramas más. En la rama RELENG_6, vemos que más del 50 % corresponde a actividad correctiva, bajando en contrapartida, la adaptativa y la perfectiva, a valores respectivos aproximados de un 20 % y un 35 %. En la rama RELENG_6_1 se observa una tendencia mucho más acusada: la actividad correctiva abarca casi el 90 % de las transacciones, existiendo un porcentaje de un 7 % de adaptativa y un 3 % de perfectiva.

Esta tendencia se repite tomando cualquier otra rama estable (tipo RELENG_N_M), lo cual no es sorprendente: en las ramas estables, la mayor parte de la actividad se centra en la resolución de fallos críticos y de seguridad, mientras que la mayor parte de la actividad de producción de nuevas funcionalidades la vamos a encontrar en la rama inestable HEAD. Una inspección realizada sobre el código demuestra que, a pesar de que en teoría no debiera ser así, es cierto que hay pequeñas partes de actividad adaptativa en las ramas estables y en las de liberación; mientras que en HEAD se aplican también correcciones de fallos críticos (que casi con seguridad habrá sido descubierto en una rama estable). Por otro lado, en general se observa que la actividad correctiva tiende a ser siempre inferior (en número de commits) que otros tipos (ver figura 3), tal como se puede esperar.

5. Conclusiones

En este artículo hemos visto algunas posibilidades que el análisis de los repositorios de control de versiones puede brindarnos a la hora de conocer mejor un proyecto, más allá del simple análisis de su código fuente.

Hemos visto que es posible caracterizar la actividad de la producción de código fuente, con un análisis que, enfocado más hacia las transacciones que a los commits individuales, da resultados en principio mejores. Hemos visto que las transacciones pueden quedar caracterizadas por varias propiedades globales, como son el tamaño (en número de ficheros o número de líneas totales), el número de líneas añadidas o borradas;

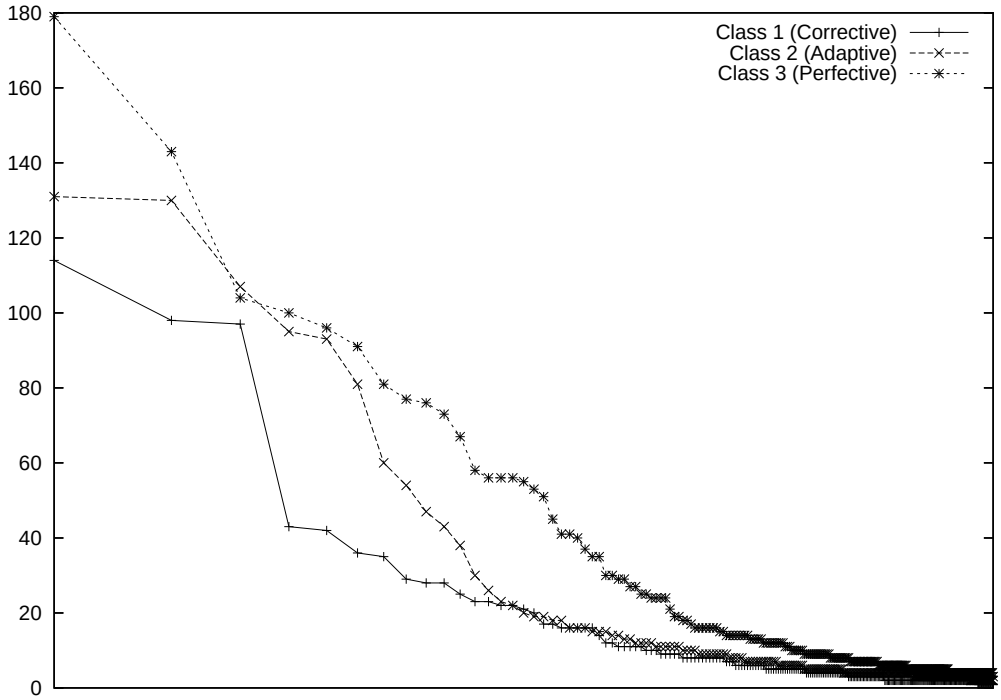


Figura 3. Número de ficheros implicados en cada tipo de transacción (rama HEAD).

o incluso el tipo de fichero predominante en la transacción como esbozo de una clasificación inicial de la actividad en el repositorio.

En este artículo, específicamente, se introduce una clasificación de los tipos de transacciones de código acuerdo, primero, con la clasificación clásica en actividad correctiva, adaptativa y perfectiva; y en segundo lugar con respecto a su posible influencia en el esfuerzo requerido para escribir el código de la transacción. Esta clasificación se complementa, además, con un tipo general de transacciones cuya actividad no se debe normalmente a acciones relacionadas con el desarrollo del código, sino a actividades de carácter administrativo como la gestión de etiquetas o ramas o los cambios de copyright. Creamos así una cuarta clase general, la de los cambios administrativos. En total, la clasificación propuesta consta de 26 clases, aunque en posteriores estudios podría ampliarse en nuevas clases.

Dada la complejidad que supone aplicar metodologías tradicionales a tan alto número de clases, en este artículo también se propone una nueva metodología de clasificación automática, pero inicialmente ayudada por expertos, basada en el método bayesiano de clasificación con aprendizaje. Esta metodología se ha aplicado a nuestro caso de estudio, verificando que los resultados que se obtienen de la clasificación son coherentes con los resultados que se podían esperar, considerando que la metodología es bastante prometedora. Sin embargo, por sí sola presenta limitaciones: al depender exclusivamente de las descripciones de las transacciones, su comportamiento ante textos ambiguos no es bueno. Tampoco presenta buenos resultados cuando se aplican textos que corresponden con palabras de uso infrecuente, al no formar parte probablemente del conjunto de aprendizaje que se utilice.

Como trabajo futuro podemos plantear el mejorar el método de clasificación, ayudándolo con el resto de los atributos que caracterizan la transacción. Otras líneas futuras son aplicar esta misma metodología,

ayudada seguramente de otros atributos, para caracterizar la actividad de los desarrolladores en repositorios de información diferentes al del control de versiones, como puede ser las listas de correo electrónico o los sistemas de gestión de erratas, en los que buena parte de la información sobre la actividad reside en el texto en lenguaje natural que describe el mensaje o el informe de errata y no en el resto de los atributos. Sin duda, esperamos que una metodología similar nos ayude en un trabajo de caracterización de la actividad de los desarrolladores de software libre para una posterior aplicación en las estimaciones de esfuerzo [1].

Referencias

[1] 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.

[2] I. Androutsopoulos, J. Koutsias, K. V. Chandrinos, G. Paliouras, and C. D. Spyropoulos. An evaluation of naive bayesian anti-spam filtering. In *Workshop on Machine Learning in the New Information Age*, pages 9–17, Jun 2000.

[3] N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian network classifiers. *Machine Learning*, 29(2 - 3):131–163, November 1997.

[4] D. M. Germán. An empirical study of fine-grained software modifications. In *Proceedings of the International Conference in Software Maintenance*, Chicago, IL, USA, 2004.

[5] T. L. Graves and A. Mockus. Inferring change effort from configuration management databases. In *5th IEEE International Software Metrics Symposium*, pages 267–, Bethesda, Maryland, USA, 1998.

[6] B. Massey. Longitudinal analysis of long-timescale open source repository data. In *Proceedings of the International Workshop on Predictor Models in Software Engineering (PROMISE 2005)*, St.Louis, Missouri, USA, 2005.

[7] A. K. Mccallum. Bow: A toolkit for statistical language modeling, 1996. http://www.cs.cmu.edu/˜mccallum/bow.

[8] A. Mockus and L. G. Votta. Identifying reasons for software changes using historic databases. In *Proceedings of the International Conference on Software Maintenance*, pages 120–130, October 2000.

[9] G. Robles, J. M. Gonzalez-Barahona, and J. J. Merelo. Beyond source code: The importance of other artifacts in software development (a case study). *Journal of Systems and Software*, 79(9):1233–1248, September 2006.

[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.

[11] E. B. Swanson. The dimensions of maintenance. In *Proceedings of the 2nd International conference on Software Engineering*, pages 492–497. IEEE Computer Society Press, 1976.