Universidad Rey Juan Carlos

Escuela Técnica Superior de Ingeniería de Telecomunicación

Ingeniería de Telecomunicación - Licenciatura en Administración y Dirección de Empresas

Estimación de Esfuerzo de Desarrollo en Proyectos Free/Open Source Software (FOSS) Mediante Minería de Repositorios Software

Proyecto Fin de Carrera

Autor: Carlos Cervigón Ávila

Tutor: Gregorio Robles Martínez

Curso Académico 2013/2014

Proyecto Fin de Carrera

Estimación de Esfuerzo de Desarrollo en Proyectos Free/Open Source Software (FOSS) Mediante Menería de Repositorios Software

Autor
Carlos Cervigón Ávila

Tutor
Gregorio Robles Martínez

La defensa del presente Proyecto Fin de Carrera se realizó el día de de , siendo calificada por el siguiente tribunal:

PRESIDENTE:

SECRETARIO:

VOCAL:

y habiendo obtenido la siguiente calificación:

CALIFICACIÓN:

Fuenlabrada, a de de .

Agradecimientos

En primer lugar quisiera dar las gracias por todo a mi familia. A mis padres, pues sin su apoyo y educación nunca hubiera llegado hasta donde estoy ahora mismo. A mis hermanos, aunque les gustase sacarme de quicio (sobre todo mi hermana en época de exámenes). A todos mis tíos y primos, sobretodo a mi tía Eli, que siempre ha sido un apoyo, y mi tío Carlos, con quien me encanta pasar un buen rato por el monte con la bici desconectando de todo. Y a mis abuelos, que aunque ya no estén entre nosotros nunca les olvidaré por todo lo que han sido para mí.

Me gustaría dar las gracias a todos mis compañeros y amigos, pues durante todos los años de la carrera me han hecho pasar muy buenos momentos, al igual que he podido trabajar codo con codo con ellos para lograr nuestras metas tanto académicas como personales. Algunos como Borja, que desde que le conocí en la ESO y posteriormente decidió hacer la misma carrera que yo ha sido un compañero y amigo inseparable. Juancar un amigo fiel, sensato y que dispone de la mayor biblioteca de enlaces a paginas web raras :D. Helen nuestra exótica teleco, Fran el cabeza loca, Lore nuestra internacional, Noelia pues sin sus apuntes LADE hubiese sido un infierno, David, Maria, Angela, Alberto, Sara, Jose, Miguel... gracias a todos por estar ahí.

También me gustaría dar las gracias a Gregorio por darme la oportunidad de participar en este interesante proyecto. Por empujarme a defender mi trabajo en comunidad y en inglés durante el BENEVOL 2013 y así darme cuenta lo importante que es este idioma en nuestro mundo. Y por guiarme en un entorno completamente desconocido para mi como es el campo de la investigación. Así como a todos los profesores que gracias a sus conocimientos y aptitudes me han brindado la oportunidad de llegar hasta aquí.

Por último y muy importante, quiero agradecer a mi amor Ángela todo el tiempo que ha estado apoyándome. Pues desde que nos conocimos hace 5 años las clases de LADE han sido mucho más amenas, gracias a ella he aprendido muchísimas cosas nuevas, ha sido un apoyo constante hasta en los momentos más difíciles para seguir dando el 100 % y estar aquí finalizando la carrera, y por supuesto espero que continue a mi lado dándome su apoyo para seguir creciendo tanto personal como profesionalmente.

Resumen

Este Proyecto de investigación se ha enfocado en la búsqueda de un modelo de estimación de esfuerzo en proyectos Free/Open Source Software (FOSS). Con este nuevo modelo se pretende proporcionar una herramienta de valoración del trabajo realizado por los desarrolladores más fiel a la realidad, pues los modelos actuales están más orientados a proyectos de índole industrial y no a proyectos FOSS.

En proyectos FOSS, desarrolladores voluntarios pueden trabajar conjuntamente con desarrolladores pagados por empresas [29]. En estos casos la cantidad de esfuerzo total invertido en el proyecto es normalmente desconocido incluso después de su finalización.

Si bien durante sus inicios los proyectos FOSS eran desarrollados principalmente por voluntarios, hoy en día la colaboración entre desarrolladores es muy variada, pudiendo encontrar proyectos que van desde participación casi exclusiva por voluntarios, proyectos basados en la colaboración de empresas y voluntarios (por ejemplo, GNOME o Linux), hasta proyectos claramente industriales en los que el desarrollo principal está a cargo de empresas (por ejemplo, WebKit o OpenStack) [28, 30].

En el caso de la participación de una empresa, la obtención general del esfuerzo realizado en el desarrollo durante un período de tiempo, que suele ser una medida importante para las compañías, no es sencillo. Pues las empresas individualmente son conscientes del esfuerzo que han invertido en dicho proyecto, pero no es fácil que tengan una idea clara del esfuerzo invertido por el resto de la comunidad, incluyendo a las demás empresas que colaboran en el proyecto.

Además, no sólo empresas colaboradoras en proyectos FOSS están interesadas en la estimación de esfuerzo en estos proyectos, también fundaciones que actúan como paraguas de proyectos FOSS están interesadas en estos datos, pues pueden ofrecer pistas sobre el proyecto y así atraer el interés industrial [31].

Para ello, el primer paso dado ha sido el estudio de los resultados obtenidos en investigaciones previas en el mismo campo [22, 26]. Con ello se obtiene una referencia de los posibles caminos a seguir en la búsqueda de dicho modelo.

Así, se ha procedido a analizan diferentes vías de valoración de esfuerzo, cada una de ella considerando aspectos diferentes con los que solventar las carencias y errores cometidos en las anteriores.

Una vez obtenidos varios métodos se ha observado la necesidad de encontrar alguna manera de validar los resultados obtenidos, por lo que se elabora una encuesta para obtener información de esfuerzo directamente de los desarrolladores de los proyectos.

Finalmente se comparan y validan los resultados obtenidos mediante los diferentes métodos considerados con los obtenidos a partir de la encuesta.

Índice general

Índice de figuras

Capítulo 1

Introducción

En este proyecto se ha investigado, elaborado y validado un modelo de estimación de esfuerzo para proyectos FOSS alternativo a los actuales, que están más enfocados a la industria. Por ello en esta introducción se va a explicar el contexto del proyecto, las motivaciones para realizarlo, como se ha desarrollado el trabajo y una breve explicación de cómo se estructura la memoria.

1.1. Contexto

El presente proyecto fin de carrera se corresponde con el trabajo realizado dentro del grupo de investigación Libresoft. Dicho trabajo ha sido realizado dentro de una beca de 20 horas semanales desde septiembre del 2013 hasta junio del 2014 y bajo la supervisión de Gregorio Robles Martínez. En el proyecto también han colaborado Andrea Capiluppi de la Universidad de Brunel, Jesús González Barahona de la Universidad Rey Juan Carlos y Daniel Izquierdo Cortazar de Bitergia S.L.

El proyecto está enfocado en la búsqueda de un modelo de fiable y sencillo con el que poder estimar, mediante el análisis de repositorios, el esfuerzo realizado por cada uno de los desarrolladores o empresas, en un proyecto FOSS concreto. Esto es debido a que en la actualidad no existe ningún modelo orientado a dicho tipo de proyectos que emplee toda la información que es posible extraer de los sistemas de control de versiones, sistemas de control de errores y listas de correo electrónico.

El modelo planteado en este proyecto fin de carrera busca ser una herramienta sencilla, fiable y aceptada por la comunidad como sustituta de los modelos actuales, no orientados a proyectos FOSS, complejos como puede ser COCOMO.

1.2. Motivación personal

La principal motivación para elegir este proyecto fin de carrera fue el proceso que había que seguir para lograr el fin buscado, es decir, al ser un proyecto de investigación no sólo iba a ser necesario crear un programa que realizase una tarea concreta, sino que había que analizar estudios anteriores, pensar en un sistema que realice la tarea mejor que las actuales soluciones, y una vez encontrada la mejor solución, elaborar un programa que permita emplear la solución propuesta con cualquier proyecto.

Otra motivación importante fue la posibilidad de experimentar el estilo de trabajo en la universidad, compartiendo en ciertos momentos ideas y formas de trabajar con algunos de los profesores que hemos tenido a lo largo de la carrera, así como con otros investigadores que han participado en la investigación.

Por tanto, la suma de investigación, búsqueda de soluciones posibles, elaboración de los diversos métodos y herramientas para lograr los objetivos y la experiencia de trabajar en la universidad, fueron la causa de elección del proyecto.

1.3. Desarrollo del proyecto

El proyecto se ha dividido en diversas fases, con el fin de obtener un modelo válido, fiable y aceptado por la comunidad en la estimación de esfuerzo en desarrollo software.

En la primera fase se ha llevado a cabo un proceso de investigación, recopilación y estudio de los modelos de estimación de esfuerzo existentes hasta la fecha. Con ellos es posible realizar una captura de las diversas vías de investigación que han sido llevadas a cabo, con sus fortalezas y debilidades. Y así poder ir orientando la dirección que debe tomar el proyecto.

En una segunda fase se ha procedido a analizar en diversos proyectos la estructura de trabajo realizada por sus desarrolladores. Con los datos extraídos se han ido elaborando métodos sucesivos de estimación de esfuerzo, los cuales mejoraban las carencias de los predecesores, buscando encontrar el método definitivo que fuese lo más fiel posible a la realidad.

Debido a la dificultad de encontrar datos que nos permitan validar los resultados obtenidos con los diversos métodos, en la tercera fase se ha procedido a realizar una encuesta a los desarrolladores de proyectos de diverso índole y tamaño.

En la última fase, con los datos de esfuerzo recogidos directamente de los propio desarrolladores, se han comparado con los generados por los métodos. Llegando a las conclusiones sobre la validez de nuestro estudio.

1.4. Estructura de la memoria

Con el fin de facilitar la lectura de este documento, se cree necesario explicar brevemente su estructura, indicando la finalidad de cada uno de los capítulos que lo componen:

1. **Introducción**. Cuyo propósito es enmarcar el contexto del proyecto, para que el lector pueda entender su finalidad, así cómo indicar la motivación de realizarlo y como se ha desarrollado.

2. **Objetivos**. En los que se indica la meta que se persigue, así cómo los diferentes objetivos parciales seguidos para lograr dicha meta y un diagrama GANTT con los tiempos de cada una de las tareas.

3. **Estado de investigación**. Presentación de los diferentes conceptos, herramientas y estudios existentes en este campo de investigación.

4. **Estado tecnológico**. Presentación de las principales tecnologías empleadas en el proyecto.

5. **Métodos de valoración**. Listado de los diferentes métodos de estimación de esfuerzo elaborados a lo largo del proyecto, así cómo las causas de la progresiva evolución entre ellos hasta llegar al método definitivo.

6. **Resultados y validación**. Muestra los resultados obtenidos de aplicar cada unos de los métodos, la necesidad de crear una encuesta para validar dichos resultados y las conclusiones obtenidas sobre el método de estimación de esfuerzo analizado en el proyecto según los datos obtenidos de las encuestas.

7. **Conclusiones y trabajos futuros**. Capítulo final destinado a la evolución global del proyecto. En esta evaluación se analiza se se han alcanzado los objetivos, las lecciones empleadas de la carrera, las obtenidas gracias a la realización del proyecto y las futuras líneas de trabajo del proyecto.

Seguidamente, se encuentra un apartado con los apéndices que dan soporte a esta memoria y la bibliografía empleada en su elaboración. En el apéndice se puede encontrar los pasos necesarios para la instalación de las herramientas empleadas, un análisis de los beneficios de emplear una función de smoothing a la hora de analizar el trabajo de los desarrolladores, así como las causas de elegir un tipo y tamaño de ventana específico, el abstract enviado al seminario BENEVOL 2013 para su participación, la presentación empleada en dicho seminario, el paper enviado a las conferencias sobre minería de repositorios software MSR 2014 para su participación, el resumen elaborado a partir de los

resultados obtenidos de la encuesta a OpenStack y el diagrama GANTT explicado en el capítulo segundo sobre los objetivos.

Capítulo 2

Objetivos

2.1. Objetivo principal

El objetivo principal que se aborda en el proyecto es el de encontrar un modelo válido, fiable, apoyado por la comunidad y lo más preciso posible que sirva para valorar el esfuerzo realizado por los diferentes desarrolladores participantes en el proyecto FOSS a analizar.

Como ya se ha comentado en la introducción, es una labor de investigación complicada, por ello para abordar dicha labor se ha ido marcando diversos objetivos parciales con el fin de ir cumpliendo metas en la obtención del modelo deseado.

2.2. Objetivos parciales para la consecución del objetivo principal

Por orden cronológico los objetivos parciales marcados han sido:

- Instalar y empezar a utilizar las herramientas de Metrics Grimoire y Viz Grimoire para coger práctica de su uso y del método de empleo con repositorios Git, analizar la información que proporciona, la relación entre las diversas tablas de la base de datos que genera y pensar en la manera de utilizar toda esta información para obtener las primeras estimaciones de esfuerzo.

- Obtener los primeros valores de esfuerzo en diversos proyectos FOSS mediante el análisis de sus repositorios. Para la obtención de este esfuerzo se emplearía un método de valoración básico como el del Método 1 (Subsección 5.1.1).

- Una vez se tienen los primeros valores de esfuerzo analizar las carencias y errores cometidos en cada uno de los métodos sucesivos empleados y pensar en formas más eficientes de estimación.

- Mostrar los avances realizados sobre la investigación de este nuevo modelo de estimación de esfuerzo a la comunidad. Para ello se presentan dichos avances en el seminario de evolución de software BENEVOL 2013 [27]. El abstract puede verse en el Apéndice C y la presentación en el Apéndice D.

- Creación de una encuesta con Django. Esta irá dirigida a los desarrolladores de Openstack con preguntas sobre el trabajo realizado en dicho proyecto. Con esto logramos una retroalimentación que nos permita validar y afinar nuestros métodos de estimación de esfuerzo.

- Análisis y elaboración de un artículo para su envío y aceptación en las conferencias de minería de repositorios software MSR 2014 (Apéndice E). En este artículo se comparará los resultados obtenidos en la encuesta con los obtenidos empleando el último método analizado hasta la fecha.

- Realizar la encuesta para otros proyectos FOSS (Linux kernel, Moodle, WebKit, MediaWiki y Ceph).

- Analizar los datos obtenidos con la encuesta y la validez del modelo con estos proyectos.

2.3. Diagrama GANTT del desarrollo del proyecto

Para una visión más sencilla y esquemática de las tareas realizadas, se ha elaborado un diagrama GANTT con el tiempo empleado en cada uno de los objetivos parciales seguidos, la elaboración y mantenimiento de los diferentes programas y el tiempo dedicado a la investigación y validación de resultados a lo largo del proyecto. Este diagrama GANTT puede observarse en el Apéndice G.

Capítulo 3

Estado de investigación

En este capítulo se va a proceder a dar una visión general tanto de las herramientas como de las investigaciones sobre estimación de esfuerzos realizadas hasta la fecha. Así como una breve introducción al tipo de proyectos de los cuales se desea obtener una estimación de esfuerzo.

3.1. Free and Open Source Software (FOSS)

El software libre y de código abierto (FOSS) es el software que está licenciado para que todos los usuarios puedan estudiar, modificar y mejorar su diseño mediante la disponibilidad de su código fuente.

Para que un software pueda catalogarse como software libre o de código abierto debe cumplir los siguientes requisitos:

Requisitos del software libre:

- Libertad de usar el programa con cualquier propósito.
- Libertad de estudiar como funciona un programa y modificarlo adaptándolo a tus necesidades.
- Libertad de distribuir copias del programa pudiendo así poder ayudar al prójimo.
- Libertad de mejorar el programa y hacer públicas esas mejoras para los demás.

Requisitos del código abierto:

- Libre redistribución.
- Acceso libre al código fuente.
- Permiso de realización de trabajos derivados.

- Integridad del código fuente del autor.
- Sin discriminación de personas o grupos.
- Sin discriminación de áreas de iniciativa.
- Distribución de la licencia.
- La licencia no debe ser específica de un producto.
- La licencia no debe restringir otro software.

Como puede apreciarse ambos requisitos son muy similares, pues están íntimamente relacionados.

Detrás de cada iniciativa existen organizaciones de apoyo y promoción. Por la parte de software libre existe la Free Software Foundation [5], mientras que por la parte del código abierto existe la Open Source Initiative [12].

3.2. COCOMO

El modelo COCOMO (**CO**nstructive **CO**st **MO**del) es un algoritmo de estimación del coste de desarrollo software publicado en 1981 por Barry W. Boehm en su libro Software Engineering Economics [24]. El modelo emplea una formula de regresión básica con parámetros que se derivan de datos históricos y actuales, así como de futuras características del proyecto.

COCOMO fue publicado como un modelo de estimación de costes, esfuerzo y programa para proyectos software. Se basó en el estudio de 63 proyectos de ATW Aerospace donde era directos de investigación software y tecnología. El estudio examinó proyectos con tamaños desde las 2.000 hasta las 100.000 líneas de código. Estos proyectos estaban basados en el modelo de trabajo Waterfall, típico de la época.

En 1995 COCOMO II fue desarrollado y finalmente publicado en el año 2000 en el libro Software Cost Estimation with COCOMO II [25]. COCOMO II fue el sucesor de COCOMO 81 (la primera versión), el cual es un mejor modelo para estimar proyectos de desarrollo software modernos. Este proporciona un mayor soporte a procesos de desarrollo software modernos y una base de datos de proyectos actualizada.

COCOMO consiste en una jerarquía de tres formas cada vez más detalladas y precisas.

- El primer nivel, el COCOMO Básico es bueno para una estimación de costes rápida, temprana y burda de proyectos pequeños y medianos, pues su precisión es limitada debido a la falta de factores a tener en cuenta para diferenciar los diversos atributos

del proyecto.

Considera tres modos de desarrollo: orgánico, semiencajado y empotrado. En el modo orgánico nos encontramos con un pequeño grupo de programadores experimentados que desarrollan software en un entorno familiar. En el empotrado el proyecto tiene unas fuertes restricciones, que pueden estar relacionadas con el procesador y el interfaz hardware. Y el modo semiencajado es un punto intermedio entre los anteriores.

- El segundo nivel es el COCOMO Intermedio, en el que se introducen 15 atributos de coste para tener en cuenta el entorno de trabajo. Estos atributos se utilizan para ajustar el coste nominal del proyecto al entorno real, incrementando la precisión de la estimación.

- Y el tercer nivel es el COCOMO Detallado, el cual puede procesar todas las características del proyecto para construir una estimación. Se introducen dos características principales:

  - Multiplicadores de esfuerzo sensitivos a la fase del proyecto.
  - Jerarquía del producto a tres niveles. Se definen tres niveles de producto: módulo, subsistema y sistema. Se consideran de nivel de módulo los aspectos que representan gran variación a bajo nivel, los que representan pocas variaciones a nivel subsistema, y el resto a nivel sistema.

Este modelo de estimación de esfuerzo es el más documentado e implementado en la actualidad, aunque tiene una serie de inconvenientes, a destacar:

- Los resultados no son proporcionales a las tareas de gestión, ya que no tienen en cuenta los recursos necesarios para realizarlas.

- Se puede desviar de la realidad si se indica mal el porcentaje de líneas de comentarios en el código fuente.

- Es un tanto subjetivo, puesto que está basado en estimaciones y parámetros que pueden ser interpretados de distinta manera por distintos analistas que usen el método.

- Se miden los costes del producto, de acuerdo a su tamaño u otras características, pero no la productividad.

- La medición por líneas de código no es válida para orientación a objetos.

- Utilizar este modelo correctamente resultar muy complicado debido a los numerosos parámetros que hay que introducir para obtener un estimación válida.
- No es posible diferenciar el esfuerzo realizado por cada desarrollador, ni en diferentes períodos.

3.3. Effort Estimation by Characterizing Developer Activity

En el artículo [22] se habla del incesante incremento en el desarrollo de FOSS software que ha estado viviendo la industria durante los últimos tiempos. Esto provoca que los métodos de estimación de esfuerzo que existían se pongan en entredicho, pues como bien indican de COCOMO, se basa en el número de líneas de código para realizar la estimación de esfuerzo. Esto en desarrollo FOSS no es válido, pues la participación en dichos proyectos es muy dispar.

Por ello, los autores proponen una nueva visión en el estudio de estimación de esfuerzo, en el que no sólo se emplee las líneas de código, sino que se aproveche la información existente en los repositorios, listas de correo y bug trackers. Así se lograría valorar la actividad de los propios desarrolladores por separado y no sólo la cantidad total de código existente en el proyecto.

Para analizar estas nuevas fuentes de valiosa información a la hora de realizar estimaciones de esfuerzo mejores y más fiables, presentan las herramientas creadas por ellos para dicho objetivo (CVSAnalY, MailingListStats, GlueTheos, DrJones y CODD), siendo cada una de ellas creada para analizar una fuente de información concreta.

De estas ideas de emplear la información que puede extraerse de los repositorios y las herramientas creadas para tan efecto, de las que se nutre este proyecto fin de carrera.

3.4. Effort Estimation for FLOSS Projects: A Study of the Linux Kernel

En este artículo [26], Andrea Capiluppi y Daniel Izquierdo Cortázar analizan los períodos de actividad de los desarrolladores participantes en el Kernel de Linux para proponer un nuevo método de estimación de esfuerzo para proyectos FLOSS. En su estudio comprueban cómo los días de mayor actividad en el proyecto son los correspondientes al período de lunes a viernes, las horas en las que mayor número de commits se realizan son las comprendidas entre las 9 y las 17 horas y que la mejor calidad del código se encuentra en dicho período.

Para comprobar la calidad de la actividad realizada en los diferentes períodos analizan tanto las líneas añadidas, como modificadas y eliminadas. Teniendo en cuenta también los períodos pre y post lanzamiento de una *major release*, pues en estos períodos se producen incrementos y descensos de actividad significativos con respecto al nivel de actividad habitual.

Por ello proponen para dicho nuevo método de estimación de esfuerzo, una división de las horas del día en 3 períodos: Horario de Oficina (OH) de 9 a 17 horas, Después de Oficina (AO) de 17 a 1 horas y Nocturno (LN) de 1 a 9 horas. Para posteriormente estimar el esfuerzo realizado en el proyecto por los desarrolladores, analizando su actividad y ponderándola con unos pesos estimados diferentes según: el período de tiempo en el que se encuentren y si la actividad se está desarrollando antes o después de una *major release*.

3.5. What Does It Take to Develop a Million Lines of Open Source Code?

Paper de Juan Fernández Ramil, Daniel Izquierdo Cortázar y Tom Mens en el que estudian la relación entre el tamaño en líneas de código y el esfuerzo, duración y tamaño del equipo de desarrollo empleado en diversos proyectos FOSS. En el paper analizan 11 proyectos con un tamaño entre 0,6 y 5,3 millones de líneas de código. Midiendo el esfuerzo según el número de desarrolladores activos al mes. Observan la necesidad de un método de estimación de esfuerzo para proyectos FOSS, pues métodos como COCOMO no son válidos para este tipo de proyectos. Por lo que evalúan el uso de 16 modelos de regresión lineal diferentes, empleando en cada uno de ellos parejas de atributos diferentes. Llegando a la conclusión de que el mejor modelo de todos los que estudian, podría ser empleado como sustituto de COCOMO para el análisis de esfuerzo en proyectos FOSS debido a su mayor precisión.

Capítulo 4

Estado tecnológico

En este capítulo se va a proceder a dar una visión general de las herramientas tecnológicas empleadas en el proyecto.

Dichas herramientas van desde las empleadas para la elaboración de los programas de análisis, hasta las tecnologías de control de versiones donde están alojados los proyectos FOSS que se busca analizar y cuantificar.

4.1. Python

Python [13] es un lenguaje de programación interpretado con un nivel de abstracción alto, usa tipado dinámico, multiplataforma y orientado a una sintaxis clara para disponer de un código legible. Además admite varios paradigmas de programación como programación orientada a objetos, programación imperativa y programación funcional.

Fue creado a finales de los ochenta por Guido van Rossum en los Países Bajos.
Python es un ejemplo de proyecto FOSS distribuido bajo una licencia de código abierto (*Python Software Foundation License*) compatible con la licencia GPL [6] a partir de la versión 2.1.1, e incompatible en ciertas versiones anteriores.

Otras de sus características más importantes son: la disponibilidad de múltiples librerías o APIs, su tratamiento de excepciones, el uso de clases y módulos, sensible a mayúsculas y minúsculas, su gestión automática de memoria transparente para el programador (reservándola cuando es necesaria y liberándola cuando deja de emplearse) y el empleo de estructuras de datos sencillas y flexibles como diccionarios, listas o strings como parte integral del lenguaje.

Actualmente existen 2 versiones la 2.7.6 y la 3.4. Las versiones 3.x son el futuro de este lenguaje de programación, pero debido a las numerosas APIs y frameworks que emplean las versiones 2.x de Python se sigue manteniendo el soporte para la 2.x, aunque no va a seguir evolucionando. En este proyecto se ha empleado Python 2.7 debido a la incompatibilidad por ahora de algunos módulos, como el de acceso a las bases de datos MySQL.

4.1.1. Django

Django [4] es un framework de desarrollo web de código abierto escrito en Python. Django se centra en la reutilización, la conectividad y extensibilidad de los componentes que forman un sitio web.
Sigue el paradigma de Modelo-Vista-Controlador (MVC). Este paradigma es un modelo de abstracción de desarrollo software que separa los datos de una aplicación (Modelo), la interfaz de usuario (Vista) y la lógica de negocio (Controlador) en tres componentes distintos.

Sus funcionalidades más importantes a destacar son:

- Herramienta para la gestión de la aplicación.
- Framework de capa de presentación.
- Manipulación de bases de datos mediante el mapeo de objetos de la aplicación a entradas relacionales.
- Seguridad (XSS, SQL Injection, CSRF, etc.).
- Sistema de serialización para general y procesar instancias del modelo de la aplicación Django representadas en formatos como JSON o XML, caché, internacionalización, plantillas, etc.

Pero lo más importante es que permite desarrollar sitios web complejos y dinámicos de una manera rápida y sencilla.

Actualmente la versión más reciente del framework es la 1.6.2, que se corresponde con la empleada en la encuesta del proyecto.

4.1.2. SciPy

SciPy [17] es una biblioteca de herramientas y algoritmos matemáticos open source. Está compuesta por las herramientas NumPy, Matplotlib, SciPy library, IPython, Sympy y pandas. De todas ellas nos centraremos en las 2 primeras, pues son las empleadas para el análisis de la actividad de los desarrolladores y la creación de las representaciones de su actividad en el proyecto.

NumPy

NumPy [11] es una librería de Python que agrega un mayor soporte para vectores y matrices, constituyendo una biblioteca de funciones matemáticas de alto nivel para operar con esos vectores o matrices. Para este proyecto la función más interesante de la librería es la capacidad de procesar vectores de actividad mediante funciones de suavizado. Al igual que la capacidad de generar la estructura de datos necesaria para generar las gráficas de actividad con Matplotlib.

Matplotlib

Matplotlib [8] es una librería de Python capaz de generar figuras en 2 ó más dimensiones a partir de las listas o arrays elaboradas con NumPy. Estas figuras son fáciles de crear, con soporte para LaTeX en leyendas y textos, en diversos formatos de alta calidad y con la posibilidad de tratarlas en un entorno interactivo.

4.2. Bases de datos

En el proyecto se ha empleado varios tipos de bases de datos para almacenar toda la información.

Las herramientas de Metrics Grimoire y Viz Grimoire almacenan gran cantidad de información para analizar los repositorios de los diversos proyectos estudiados. También es necesario un sistema de almacenamiento para recoger las respuestas de los numerosos desarrolladores que han aceptado muy amablemente a ayudar completando la encuesta

realizada y explicada en detalle más adelante.
Por tanto ha sido necesario implementar dos tipos de bases de datos a la hora de trabajar en el proyecto. Una base de datos MySQL para la información obtenida de los repositorios y una base de datos SQLite para el resultado de las encuestas.

4.2.1. MySQL

MySQL [10] es un sistema de gestión de bases de datos relacionales, multihilo y multiusuario creada por MySQL AB. Está disponible en 2 tipos de licencias diferentes según su uso, una GPL y una comercial. Se emplea en el proyecto este tipo de base de datos por ser el sistema por defecto que emplea las herramientas de Metrics Grimoire. Aún así es posible destacar las siguientes características de MySQL:

- Es multiplataforma.
- Usa GNU Automake, Autoconf y Libtool para portabilidad.
- APIs disponibles en múltiples lenguajes de programación.
- Empleo de multihilos para mejorar la velocidad.
- Soporta gran cantidad de tipos de datos para las columnas.
- Gestión de usuarios y passwords para mejorar la seguridad.
- Gran cantidad de utilidades de administración.
- Proporciona sistemas de almacenamiento tanto transaccionales como no transaccionales.

4.2.2. SQLite

SQLite [18] fue creado en el año 2000 por D. Richard Hipp. Es un completo sistema de gestión de bases de datos relacional de dominio público que soporta múltiples tablas,

índices, triggers y vistas. El empleo de este tipo de base de datos en el proyecto se debe no sólo a ser la base de datos por defecto de Django, sino a las múltiples características que posee:

- Más eficiente y rápido que MySQL o PostgreSQL.
- No necesita un proceso separado funcionando como servidor, sino que lee y escribe sobre archivos que se encuentran en el disco duro.
- Es multiplataforma.
- Emplea registros de tamaño variable de forma que se utiliza el espacio en disco que realmente sea necesario en cada momento.
- Pequeña librería en C de como máximo 500KiB completamente configurada.
- Implementa un sistema ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad).
- No necesita de configuración previa.
- Implementa la mayor parte del estándar SQL-92.

4.3. Metrics Grimoire y Viz Grimoire

Metrics Grimoire [9] es un conjunto de herramientas creadas por el grupo de investigación Libresoft [7] de la Universidad Rey Juan Carlos [19]. Actualmente, junto con las herramientas de Viz Grimoire [20], son mantenidas por la empresa Bitergia [2].
Estas herramientas obtienen datos de los repositorios software (información de los commits, control de tickets, comunicaciones el listas de correos, bugs, etc.) para posteriormente ser procesados y generar una base de datos con toda la información relevante sobre el proyecto analizado. Son estos datos almacenados en la base de datos los que se emplean en este proyecto para estudiar el esfuerzo realizado por los desarrolladores a lo largo del tiempo en el proyecto.

Viz Grimoire por su parte es un framework que analiza la base de datos y genera un dashboard en HTML con la toda la información obtenida.

4.3.1. CVSAnalY

Dentro de Metrics Grimoire encontramos CVSAnalY [3]. Esta herramienta recupera y organiza la información del sistema de control de versiones del código fuente. Posteriormente almacena dicha información estructuradamente en diversas tablas en una base de datos.

4.3.2. RepositoryHandler

RepositoryHandler [16] contiene las librerías necesarias para que el resto de herramientas puedan manejar el repositorio del código fuente.

4.3.3. Bicho

Bicho [1] se encarga de recuperar y organizar la información extraída de los sistemas de control de errores. Como CVSAnalY almacena el resultado en una base de datos.

4.3.4. VizGrimoreUtils

Dentro de Viz Grimoire encontramos VizGrimoireUtils [21], una librería con herramientas para analizar identidades, analizar posibles repeticiones de desarrolladores con diversos alias a lo largo del proyecto y generar tablas con relaciones de desarrolladores con las empresas participantes en el proyecto.

4.4. R

Para analizar los resultados obtenidos de las diferentes encuestas, así como para elaborar las gráficas de dichos resultados y de la evolución de los commits y autores de los proyectos estudiados se ha empleado este lenguaje de programación.

R [14] es un lenguaje y entorno de programación creado en 1993 para análisis estadístico y gráfico. Se trata de un un proyecto multiplataforma de software libre distribuido bajo la licencia GNU GPL.

R proporciona un amplio abanico de herramientas estadísticas (modelos lineales y no lineales, test estadísticos, análisis de series temporales, algoritmos de clasificación y agrupamiento, etc.) y gráficas. También permite conectarse a bases de datos para realizar consultas y análisis de sus datos.

Además, al formar parte de un proyecto colaborativo y abierto, los usuarios pueden realizar y publicar paquetes que extiendan su configuración básica. Muchos de ellos podemos encontrarlos en su repositorio oficial CRAN [15], el cual está organizado en vistas que permiten agruparlos por naturaleza o función debido al alto número de paquetes (actualmente más de 5000 paquetes).

4.5. Control de Versiones

La función de un sistema de control de versiones es registrar los cambios realizados sobre uno o varios archivos a lo largo del tiempo, permitiendo recuperar versiones específicas de los archivos más adelante.

Para las empresas y desarrolladores esta herramienta es muy útil, debido a que puedes volver fácilmente un archivo o un proyecto completo a un estado anterior, así como realizar comparaciones a lo largo del tiempo o comprobar quien fue el último en realizar un cambio en un archivo en concreto.

Para llevar un control de versiones podemos emplear un método manual (creación de carpetas con las diferentes versiones) aunque lo recomendado es hacerlo mediante un sistema automático. Estos sistemas facilitan la tarea, evitan errores de sobre-escritura errónea de archivos y dotan al desarrollador de muchos beneficios extra. Existen numerosos de estos sistemas, como: Git, CVS, Subversion, Bazaar, Mercurial, etc.

Las características más destacadas son:

- Mecanismo de almacenamiento de los elementos a gestionar
- Posibilidad de realizar cambios sobre los elementos almacenados.
- Registro histórico de las acciones realizadas en cada elemento o conjunto de elementos.
- Posibilidad de volver a versiones antiguas de los archivos o proyectos.
- Varias arquitecturas de almacenamiento (local, centralizado o distribuido)

Los sistemas **locales** guardan información de los cambios en una simple base de datos localizada en el propio equipo.

Los sistemas **centralizados** se basan en la colaboración entre desarrolladores, por lo que un único servidor dispone de todos los archivos versionados y los desarrolladores descargan sólo la version de los archivos requerida a su equipo local desde ese lugar central. Esta configuración permite que todo el mundo conozca en que están trabajando el resto de desarrolladores. Teniendo entonces la administración un mayor control de lo que se está realizando. El mayor problema que supone este sistema es en el caso de que el servidor se caiga, pues nadie podrá colaborar o guardar sus cambios durante ese tiempo. Al igual que si no se lleva un proceso de copias de seguridad adecuado puede perderse todos los cambios ante un problema en los discos duros que almacenan el proyecto.

Los sistemas **distribuidos** al igual que los centralizados se basan en la colaboración. Pero la mayor diferencia se basa en que en este caso los desarrolladores en lugar de descargarse una version de los archivos del proyecto se descargan el sistema de versiones al completo. Por lo que en caso de perderse la información en el servidor central se podría recuperar con cualquiera de las copias realizadas por los desarrolladores. Manteniendo por lo tanto los repositorios centrales más limpios y requiriendo menos recursos al realizarse la mayor parte del trabajo en las copias locales. Quedando por tanto el servidor central como punto de sincronización de los distintos repositorios locales.

4.6. Cuadro resumen

En la siguiente figura se puede observar la relación entre las diferentes tecnologías explicadas en el capítulo y empleadas a lo largo del proyecto.

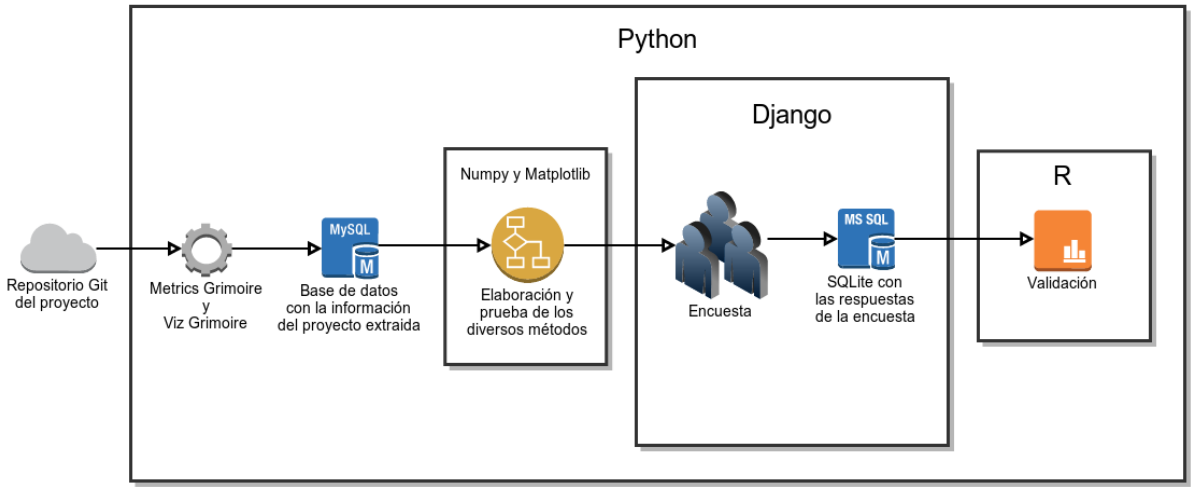


Figura 4.1: Cuadro resumen del uso de las diferentes tecnologías.

Capítulo 5

Métodos de valoración

En este capítulo se va a proceder a explicar detalladamente cada uno de los métodos creados y empleados. Estos métodos según orden de creación y análisis reflejan el proceso de búsqueda de la mejor vía de valoración de los proyectos FOSS en el que se ha visto involucrado el proyecto. Cada uno ha ido elaborándose según las bondades y carencias que podíamos observar de los métodos anteriores.

5.1. Métodos estudiados

5.1.1. Método 1

Este primer método básico se ha creado debido a la falta de información y resultados de estimaciones de esfuerzo basados en el análisis de repositorios. En él se ha empleando la actividad de los desarrolladores en forma de commits.

Estructura

Su estructura es muy simple, se observa la actividad de cada desarrollador en cada uno de los meses de vida del proyecto, valorando con 1 Persona-Mes cada uno de ellos en los que haya realizado al menos 1 commit. Por lo que el esfuerzo realizado en el proyecto en cuestión sería la suma del esfuerzo de cada uno de los desarrolladores.

Implementación

Para implementarlo se ha elaborado un script que mediante consultas a la base de datos elaborada por CVSAnalY, obtiene todos los commits realizados por cada desarrollador. Una vez con ellos se elabora un vector para cada desarrollador, de longitud el número de meses de vida del proyecto y que contiene el número de commits realizados en cada mes. Por último tan sólo tenemos que sumar el número de posiciones distintas de cero

para obtener la contribución en Personas-Mes realizada por el desarrollador y agregar la de todos los desarrolladores para obtener el esfuerzo total del proyecto.

Esquema

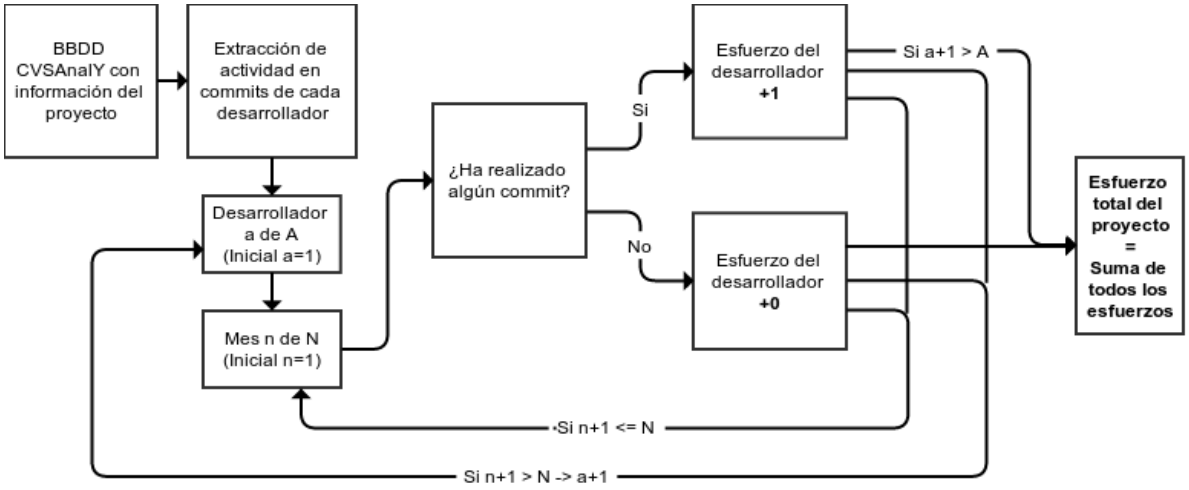


Figura 5.1: Esquema de análisis del método 1.

5.1.2. Método 2

El método 1 es un método muy básico en el que se valora por igual a todos los desarrolladores y al trabajo realizado en cada mes por cada desarrollador. Esto es un error debido a que no se puede valorar todo el trabajo por igual y por tanto a todos los desarrolladores por igual. Por ello en este segundo método se busca diferenciar a los desarrolladores según su trabajo en 3 categorías diferentes, profesionales, semi-profesionales y amateurs.

Los desarrolladores profesionales se identifican como aquellos que se dedican plenamente a dicho proyecto, invirtiendo una media de 40 horas/semana. Los semi-profesionales son los desarrolladores que dedican la mitad de su tiempo de trabajo al proyecto.

Y los amateurs son identificables como aquellos desarrolladores esporádicos que dedican según el momento o interés una pequeña parte de su tiempo al proyecto, pero que no tienen una relación duradera con el proyecto.

Estructura

Para clasificar a los desarrolladores en las diferentes categorías se emplea el número de commits realizado por cada desarrollador en el proyecto. Por tanto para clasificar a un desarrollador como profesional este debe ser responsable de al menos un 10 % del total de commits del proyecto, para clasificarlo como semi-profesional al menos un 5 % de los commits y el resto serán clasificados como amateurs.

Una vez clasificados los desarrolladores en cada una de las categorías se procede a valorar cada mes trabajado por la ponderación establecida para su categoría. En el caso de los profesionales es de 1 Persona-Mes por mes trabajado, de 5/22 Personas-Mes para los semi-profesionales y de 3/30 Personas-Mes para los amateurs.

Implementación

Para implementar esta valoración primero obtenemos el total de commits realizados por desarrollador sumando la actividad registrada en la base de datos. Una vez obtenido este valor se compara con el total de commits del proyecto para clasificarlo en la categoría que corresponda. Y por último se valora los meses trabajados por la ponderación de la categoría en la que ha sido clasificado.

Una vez se tiene todo el trabajo en Persona-Mes de cada desarrollador se suman para obtener el esfuerzo realizado en el proyecto.

Esquema

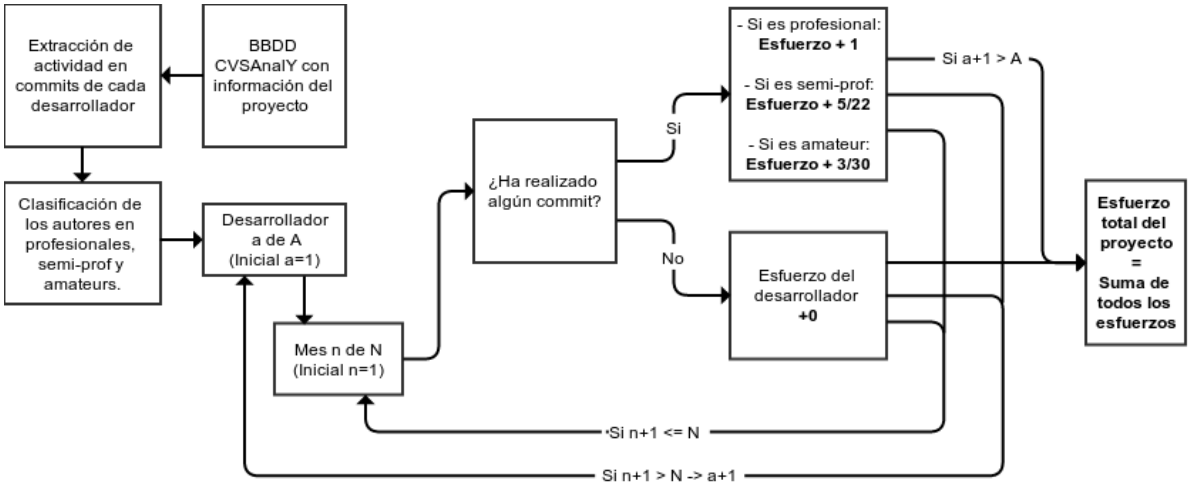


Figura 5.2: Esquema de análisis del método 2.

5.1.3. Método 3

Con el método anterior se observa que es un buen camino a seguir la división de los desarrolladores en diferentes categorías, pues es cierto que no todos los desarrolladores en un mismo proyecto trabajan lo mismo ni el esfuerzo realizado por ellos es idéntico.

Pero basándonos en la literatura, otro punto de vista interesante a la hora de valorar el esfuerzo de los desarrolladores sería tener en cuenta la relación 80/20 en el desarrollo de software. Esta relación nos dice que el 80 % del trabajo en el desarrollo de código es

hecho por el 20 % de los desarrolladores del proyecto, mientras que el restante 20 % del código es creado por el restante 80 % de los desarrolladores participantes.

Por ello en este tercer método vamos a poner el práctica dicha relación pero con alguna pequeña variación.

Estructura

Aunque en la literatura se habla de una relación 80/20, para este método vamos a emplear una relación 90/10 siguiendo los mismos principios. Pues valoraremos con 1 Persona-Mes a los desarrolladores que hayan realizado el 90 % de los commits y con 0.11 Personas-Mes al resto del trabajo realizado en el proyecto.

Implementación

Por ello, para implementar este método lo primero ha sido obtener el número de commits hechos en el proyecto por cada desarrollador. Segundo se ha procedido a ordenar los desarrolladores de mayor a menor número de commits. Y por último se ha obtenido el numero de desarrolladores que han realizado el 90 % de los commits, multiplicando dicho valor por 1.11 para obtener el número de Personas-Mes total empleado en el proyecto.

Esquema

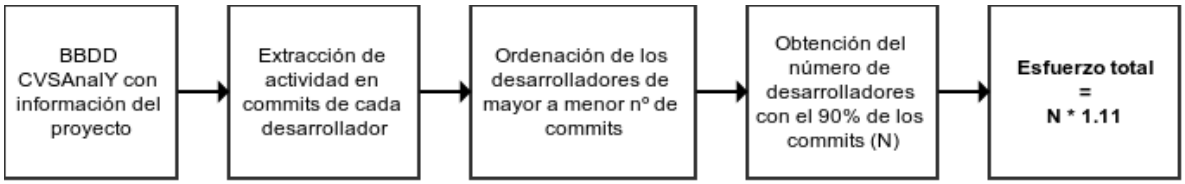


Figura 5.3: Esquema de análisis del método 3.

5.1.4. Método 4

Llegados a este punto, se observa que en todos los métodos anteriores se está cometiendo el error de valorar el trabajo de los desarrolladores de forma total e idéntica en todos los meses en los que ha trabajado. Este hecho raramente es cierto, pues un mismo desarrollador suele tener diferentes periodos de trabajo a lo largo de la vida del proyecto, por ejemplo, un desarrollador puede haber empezado a trabajar de manera parcial en un proyecto y pasando al cabo de unos meses a trabajar a jornada completa.

Por ello en este nuevo método se pasa de analizar el trabajo global de cada desarrollador, a analizar y valorar el trabajo realizado mes a mes.

Estructura

Aunque en todos los métodos anteriores se ha estado analizando el número de commits, en este nuevo método vamos a analizar la actividad mensual del desarrollador, es decir, el número de días al mes que ha estado realizando commits. También vamos a realizar una división de los desarrolladores en profesionales y el resto. Cuando se habla de profesionales hablamos de la misma distinción que en el método 2, pero a diferencia del método 2 en el que distinguíamos entre semi-profesionales y amateurs, en este nuevo método no se realiza dicha distinción, clasificando a ambos grupos como el resto de desarrolladores. Esto es así porque analizando la actividad de diversos desarrolladores clasificados en ambas categorías, se observaba que las diferencias entre ambos son prácticamente inexistentes, por lo que no es necesario dividirlos en diferentes categorías.

Para distinguir entre profesionales y el resto, se ha elegido un *threshold* de 15 días de actividad, es decir, en los meses en los que un desarrollador ha trabajado un mínimo de 15 días se le ha considerado como profesional, mientras que en los meses que su trabajo haya sido menor a 15 días se le ha considerado como no-profesional (resto).

Para valorar el trabajo realizado se analiza mes a mes a los desarrolladores, a los clasificados como profesionales se les valora con 1 Persona-Mes su trabajo, mientras que para valorar al resto se realiza una función lineal entre el primer desarrollador no-profesional con mayor actividad (PM = días trabajados / 15) y el desarrollador con menor actividad (no nula). Siendo el esfuerzo global en el proyecto como en los métodos anteriores, la suma de todos los esfuerzos.

Implementación

Para implementar este método se ha analizado el número de commits que ha realizado cada desarrollador en cada mes. Una vez tenemos el vector de actividad en cada mes de todos los desarrolladores, se ha recorrido dicha actividad, ordenada de mayor a menor actividad, comparándola con el *threshold* para obtener el número de profesionales que existen en dicho mes y el último profesional. Un vez tenemos el número de profesionales y el último profesional, se realiza una función lineal entre el primer no profesional de la lista y el ultimo desarrollador con actividad en dicho mes. Por último sólo hay que sumar el número de profesionales en el mes, con la lista de pesos obtenida de aplicar la función lineal para obtener el esfuerzo total realizado en el mes.

Esquema

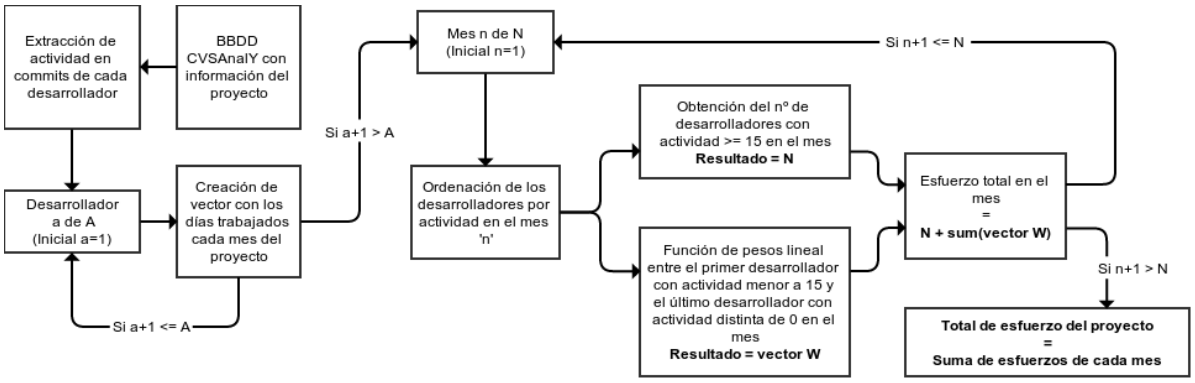


Figura 5.4: Esquema de análisis del método 4.

5.1.5. Método 5

Con método cuarto se ha observado que es un buen enfoque a la hora de buscar un método de estimación de esfuerzo, pues la valoración mes a mes es la mejor manera de no cometer errores sobre o sub-estimando esfuerzo de los desarrolladores a lo largo del proyecto. La división en profesionales y el resto es un enfoque bastante fiel de la realidad en un proyecto en el que podemos encontrarnos desarrolladores a tiempo completo en ellos y desarrolladores que dedican sólo una parte de su tiempo en él. Pero cuando se valora el esfuerzo de los desarrolladores no-profesionales mediante el empleo de una función lineal tan simple se cometen errores de cuantificación de esfuerzo, pues no se emplea el trabajo realizado como tal, sino que se emplea la posición que ocupa en la lista de desarrolladores no-profesionales según sus días trabajados. Por ello en este nuevo método se ha decidido cambiar la función lineal por una valoración basada en el número de commits realizados por dichos desarrolladores.

Estructura

La estructura empleada es idéntica a la del método anterior salvo por la valoración de los desarrolladores no-profesionales. En este caso se obtiene el porcentaje de commits realizados por los desarrolladores no-profesionales.

Implementación

La implementación como su estructura es idéntica a la del método anterior, salvo que cuando valoramos el trabajo de los no-profesionales se calcula el total de commits del mes y el número de commits de los no-profesionales. Con estos valores se obtiene el porcentaje de commits realizados por los no-profesionales en ese mes. Después se multiplica el número

de desarrolladores profesionales por 1.X, donde X es el porcentaje de commits de los no-profesionales calculado.

Esquema

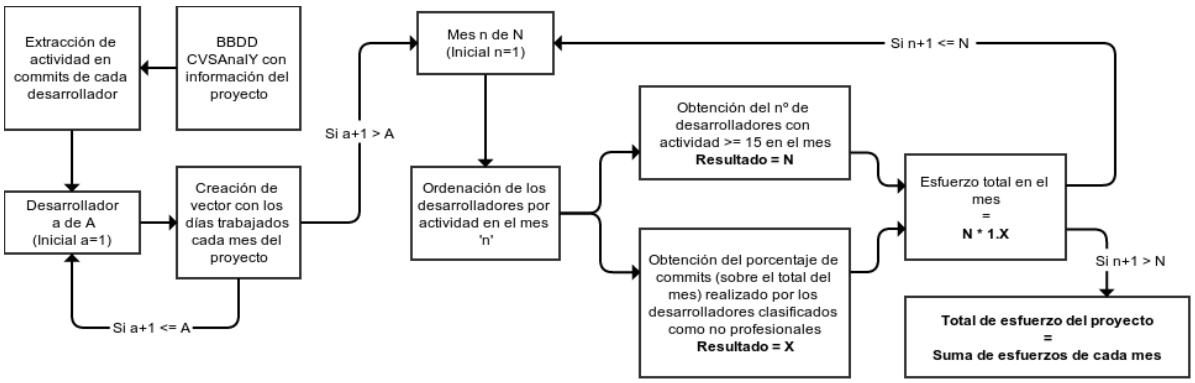


Figura 5.5: Esquema de análisis del método 5.

5.1.6. Método 6

Después de probar todas las anteriores maneras de cuantificar el trabajo realizado por los desarrolladores para poder clasificarlos como profesionales o el resto, se ha observado que la mejor manera es hacerlo de la manera que se ha implementado en los métodos 4 y 5. El problema en el que se estaba incurriendo era que haciendo esto no se valora a los desarrolladores profesionales de una manera completamente real, por ejemplo los meses que habían disfrutado de sus vacaciones muy probablemente no se les clasifique como profesionales debido a que no están haciendo commits el número mínimo de días fijado.

Para solucionar este problema se ha procedido a implementar en lugar de una cuantificación de los días de cada mes por separado, una función de *smoothing* que pondere el trabajo realizado. El análisis de las funciones de *smoothing*, así como la elección del tipo y tamaño se puede encontrar en el Apéndice B.

Consiguiendo de esta manera una función del trabajo realizado por los desarrolladores que muestra de una manera más clara y fiel los diferentes períodos de trabajo que ha tenido cada desarrollador a lo largo de la vida del proyecto.

Estructura

La estructura es bastante similar al método 4, salvo porque en este caso se realiza el proceso de suavizado indicado anteriormente con la función de *smoothing* a la actividad de cada desarrollador antes de analizar si dicha actividad mes a mes supera el *threshold*. Debido a que al realizar el suavizado los valores de actividad se ven reducidos, el *thres-*

hold es ahora situado en los 10 días, es decir, los desarrolladores serán clasificados como profesionales cuando tengan una actividad mensual de al menos 10 días haciendo commits.

Como en el método cuarto se valora a los desarrolladores profesionales con 1 Persona-Mes, mientras que el resto es valorado según la parte proporcional de los días trabajados con respecto al *threshold* que marca el mínimo para ser clasificado como profesional en cada mes.

Implementación

La implementación como se ha indicado en la estructura es similar al método 4, salvo por la adicción del suavizado previo al análisis de los valores de actividad y a la nueva valoración del esfuerzo realizado por los desarrolladores no-profesionales. Esta valoración del esfuerzo de los desarrolladores no-profesionales es calculada dividiendo la actividad realizada entre el *threshold*.

Esquema

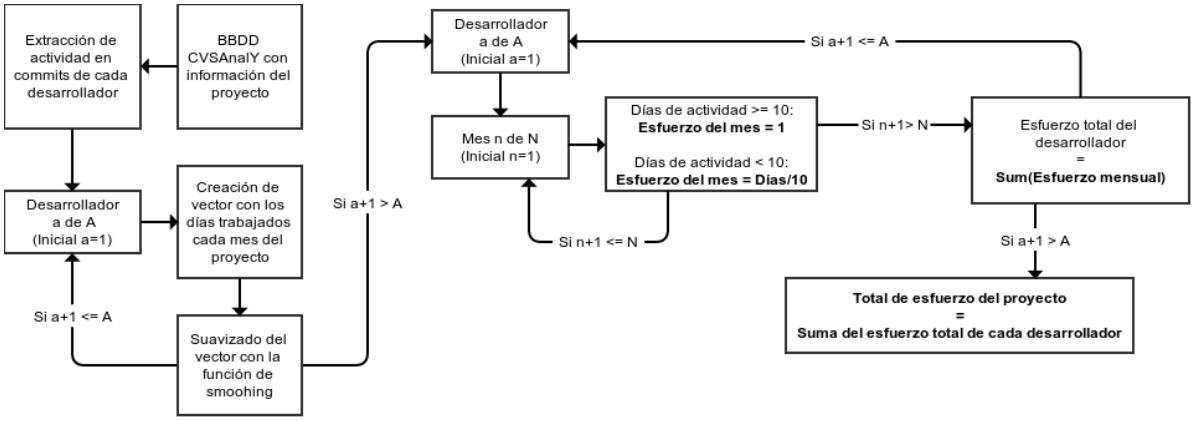


Figura 5.6: Esquema de análisis del método 6.

5.2. Resumen valorativo

Como puede observarse a lo largo del período de búsqueda del método más efectivo para valorar el esfuerzo realizado por los diferentes desarrolladores de cada uno de los proyecto. Se han ido elaborando varios métodos con enfoques diferentes sobre la información y forma de estimar el esfuerzo. Así como el salto de un método a otro se ha realizado analizando los resultados obtenidos, de los cuales se ha sacado las diferentes fortalezas y debilidades con las que realimentar el continuo proceso de búsqueda. Llegando por tanto a un último método basado en la búsqueda de un valor frontera para dividir a los autores

en profesionales o no-profesionales, en el cual se sabe que si se realiza una correcta clasificación de autores en profesionales, el margen de error que se cometería al estimar el esfuerzo del resto de autores se vería muy acotado, además de estar entre los margenes aceptados por la comunidad para este tipo de estimaciones.

De una manera gráfica, el proceso de evolución de los métodos que se ha seguido a lo largo de la investigación puede observarse en la siguiente figura:

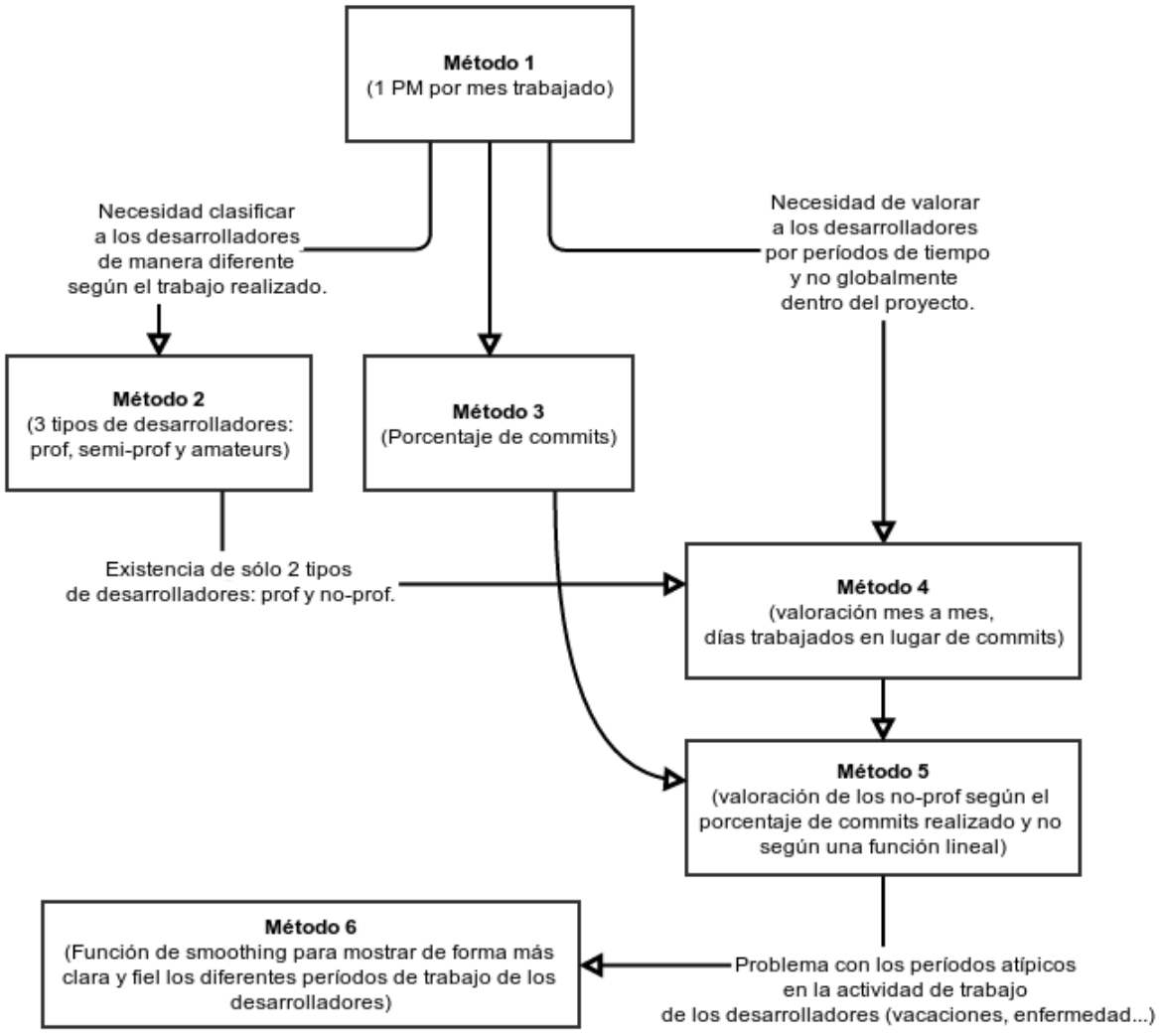


Figura 5.7: Diagrama de evolución de los métodos.

Capítulo 6

Resultados y validación

En este capítulo se muestran los resultados obtenidos al aplicar los diferentes métodos en diversos proyectos FOSS, las encuestas realizadas para validar los resultados anteriores y las conclusiones obtenidas de los datos obtenidos de las encuestas.

6.1. Aplicación y resultados de los métodos en proyectos reales

Para poder comprobar el efecto que tiene emplear cada uno de los métodos, compararlos y ver su evolución, se ha elegido una serie de proyectos FOSS alojados en GitHub en los que aplicar cada uno de estos métodos. Con ello se consigue obtener datos cuantitativos con los que trabajar en el estudio de mejoras y cambios de un método al siguiente.

Los proyectos elegidos en los que aplicar los métodos pueden verse en la Tabla 6.1.

Y los resultados obtenidos de aplicar cada uno de los métodos a estos proyectos podemos observarlos en la Tabla 6.2.

Como puede observarse al emplear el **método 1** lo que hemos logrado ha sido encontrar un techo de esfuerzo en cada uno de los proyectos, además sabemos que con resultados cercanos a ese valor estaríamos cometiendo un error muy alto en nuestra estimación.

Con el **método 2** se reducen los valores de esfuerzo, pues es un paso acertado la realización de una división de los desarrolladores, sin entrar en si el sistema empleado en este método es válido. Pero aún así la valoración y como se ha mencionado antes, la división de los autores sigue sin emplear sistemas que aprovechen toda la información que podemos obtener de los repositorios.

En el **método 3** vemos cómo los resultados de Personas-Mes difieren mucho de los anteriores, pues en el se ha empleado un sistema diferente para dividir los autores y valorar su esfuerzo. La realización de este método se produce paralelo al del **método 2** como manera de buscar una vía correcta de valoración, ya que como se ha ido mencionando

Proyecto	SLOCs	Commits	Descripción
VizGrimoireR	2,848	844	Librería en R para analizar los datos obtenidos por las herramientas de Metrics Grimoire.
Tempest	26,351	2,047	Conjunt de test de integración para ser ejecutados contra un cluster de OpenStack.
Bootstrap	26,997	7,071	Front-end framework para un desarrollo rápido y fácil de webs.
Easytag	43,607	154	Utilidad para ver y editar audio tags.
Gnoduino	52,285	1,074	Implementación de una plataforma de Arduino, para crear objetos electrónicos interactivos.
Libgee	697	192	Librería de interfaces y clases GObject-based comúnmente usadas en estructuras de datos.
Chronojump	98,675	1,792	Conexión para el microcontrolador Chronopic 3 para registrar saltos, carreras, ritmos y tiempos de reacción.
Calendar	13,419	435	Aplicación de calendario para Gnome.
Libgweather	8,905	1,786	Librería de acceso a información meteorológica desde servicios online para numerosas localizaciones.
Kupfer	31,786	3,600	Interfaz para un acceso rápido a aplicaciones y sus documentos.
Gedit	67,964	10,567	Editor de texto.
Tracker	132,374	14,562	Motor de búsqueda, herramienta de búsqueda y sistema de almacenamiento de metadatos.
Packagekit	22,490	4,648	Paquete gratuito de aplicaciones de software diseñado para proporcionar un front-end coherente y de alto nivel para diferentes sistemas de gestión de paquetes.

Tabla 6.1: Tabla de proyectos estudiados.

PM (Persona-Mes)						
Proyecto	**Método 1**	**Método 2**	**Método 3**	**Método 4**	**Método 5**	**Método 6**
VozGrimoireR	78	64,21	6,66	25,46	22,51	39,59
Tempest	551	129,33	82,14	133,19	35,98	123,57
Bootstrap	743	183,40	63,27	242,83	62,33	143,81
Easytag	136	38,96	17,76	28,50	29,72	30,70
Gnoduino	189	98,87	11,10	23,70	71,39	46,76
Libgee	89	50,22	6,66	6,76	54,35	18,90
Chronojump	291	125,65	8,88	48,39	103,38	83,86
Gnome Calendar	75	20,90	7,77	8,33	20,02	14,22
Libgweather	251	119,24	194,25	101,20	74	106,75
Kupfer	224	72,10	3,33	42,92	56,30	57,28
Gedit	3578	669,96	287,49	831,60	203,53	633,91
Tracker	1180	378,37	31,08	417,83	149,94	367,61
Gnome Packagekit	1163	220,20	103,23	216,45	82,00	182,95

Tabla 6.2: PM de cada proyecto según los diferentes métodos.

a lo largo de la memoria, la base de estudio sobre valoración de esfuerzo empleando la minería de repositorios es muy pobre, por lo el camino a seguir en esta etapa es muy difuso.

A partir de este momento en el que tenemos resultados de esfuerzo empleando diferentes técnicas y hemos visto las carencias y fortalezas de ellas, se realiza el **método 4**. En él vemos resultados cercanos a los del **método 2**, esto es debido a que se ha seguido el camino de la división de los desarrolladores, salvo que en este caso se ha refinado analizando los resultados, llegando a la conclusión de que sólo existen 2 tipos: profesionales y no-profesionales. Además se ha procedido a analizar el trabajo mes a mes, logrando así suprimir el error que se introducía cuando se valoraba homogéneamente los diferentes períodos de trabajo de cada desarrollador según su actividad global.

Al centrarnos en el **método 4** en la valoración y clasificación de los desarrolladores profesionales, el método de valoración empleado en los no-profesionales era sub-óptimo, incrementando el error introducido en la estimación de esfuerzo. Por ello en el **método 5** se continua valorando a los profesionales como antes, pero a los no-profesionales se les valora por la cantidad de commits realizado en comparación con el total del mes.

Por último, en el **método 6** se recoge todo lo aprendido hasta ahora con el resto de métodos. Se valora mes a mes, se divide autores en 2 categorías (prof y no-prof), se centra en el trabajo realizado y se solventa el problema de la inclusión de errores de estimación en los períodos vacacionales y atípicos de la actividad de los desarrolladores. Si se observan los resultados de Personas-Mes, puede verse claramente cómo se ha llegado a resultados intermedios entre todos los anteriores, pero lejanos a los del **método 1** en el que sabíamos que el error cometido era muy alto.

El problema llegados a este punto era cómo validar la precisión de estos valores. Pues no podemos compararlos con métodos de estimación anteriores como COCOMO debido a que están más orientados a proyectos industriales. Ni tampoco tenemos otros métodos orientados a proyectos FOSS. Por lo que sólo había una manera de validar estos resultados, realizando una encuesta a los propios desarrolladores sobre el esfuerzo empleado en su actividad en el repositorio. Con estos datos, en caso de ser representativos del proyecto, podríamos hacer comparaciones de resultados y validar el trabajo realizado.

6.2. Encuesta

Debido a la dificultad y falta de información disponible sobre el trabajo realizado por los desarrolladores de proyectos FOSS, se decidió elaborar una encuesta corta y directa, con el fin de poder recabar información de esfuerzo de los desarrolladores de los proyectos elegidos. Así se podría validar, mejorar y optimizar el modelo de estimación de esfuerzo que se está investigando. Como los emails de los desarrolladores se pueden obtener analizando el repositorio de cada proyecto, tan sólo se debía elaborar una encuesta atractiva para los desarrolladores y que no les tomase demasiado tiempo, así se conseguiría maximizar el numero de contribuciones a la causa.

Por tanto, la encuesta consta sólo de 5 preguntas sobre el esfuerzo realizado por los desarrolladores en el proyecto a analizar, 4 de ellas son de selección entre una lista de posibilidades, mientras que la última se trata de un cuadro de texto. Además se incluye un checkbox por si desean los encuestados recibir información sobre el resultado obtenido con la encuesta. Todo ello puede verse en la Figura 6.6.

Aunque la encuesta y por tanto las preguntas realizadas están en inglés, la traducción de estas preguntas y sus posibles respuestas es la siguiente:

- En media, ¿Cuántas horas a la semana dedicó al proyecto en los últimos 6 meses?
  - Posibles respuestas:
    - Más de 40 horas
    - Sobre las 40 horas
    - Sobre las 30 horas
    - Sobre las 20 horas
    - Sobre las 10 horas
    - Sobre las 7 horas
    - Menos de 5 horas

- ¿Cuánto tiempo del empleado en el proyecto es de desarrollo? (Entendiendo "desarrollo" en el proyecto: diseño, discusión, solución de errores, codificación...)
  - Posibles respuestas:
    - Más del 90 %
    - Sobre el 75 %
    - Sobre el 50 %
    - Sobre el 25 %
    - Menos del 10 %
- ¿Haces al menos un commit en el repositorio al día cuando codificas?
  - Posibles respuestas:
    - Cierto
    - Falso
- ¿Cómo te consideras en el proyecto?
  - Posibles respuestas:
    - Contribuidor a tiempo completo
    - Contribuidor a tiempo parcial
    - Contribuidor ocasional
- ¿Siempre trabajas el mismo número de horas en el proyecto?, ¿o has tenido diferentes fases de trabajo? Si has tenido diferentes fases, ¿puedes hablarnos sobre esas diferentes fases? (la gráfica de abajo puede ayudarle, está basada en su actividad grabada en el repositorio).

Las respuestas dadas por los desarrolladores a estas preguntas son almacenadas en la base de datos SQLite de la encuesta.

6.2.1. Estructura

El código de la encuesta consta de 3 partes importantes:

- El código base del proyecto Django de la encuesta.
  Este código incluye todos los archivos de configuración, templates HTML, CSS, imágenes genéricas incluidas en la web, base de datos inicializada pero sin información y estructura de directorios necesarios para poder ejecutar la página web.

- Scripts de inicialización y creación de la encuesta personalizada.
  Estos scripts se encargan de realizar un clon de la página web en el directorio indicado, extraer la información del proyecto deseado para completar la base de datos sin información de la encuesta y la elaboración de las gráficas de esfuerzo de cada uno de los desarrolladores para incluirlas en el directorio de imágenes.

- Script de envío de emails personalizados a los desarrolladores.
  Este script se encarga de enviar emails personalizados a los desarrolladores del proyecto a estudiar, pidiéndoles participar voluntariamente, rellenando la encuesta, en el proceso de validación del modelo. Por lo que el programa rellena una plantilla con la información de cada uno de los desarrolladores del proyecto analizado. Para posteriormente proceder con el envío de los email.

6.2.2. Implementación

Para crear la encuesta se ha empleado el framework de desarrollo web Django. Se eligió Django para elaborar la encuesta debido al conocimiento previo obtenido sobre el uso de dicha herramienta a lo largo de la carrera. Además, debido a la necesidad de disponer del código en un repositorio común en el que poder trabajar conjuntamente los participantes de la investigación, el proyecto se encuentra alojado en GitHub[1].

Para poner en funcionamiento la encuesta se debe ejecutar el script **survey_script.py** pasando una serie de parámetros tanto obligatorios como opcionales:

```
usage: survey_script.py [-h] [-dbhostname DBHOSTNAME]
        -dbuser DBUSER -dbpass DBPASS -dbname DBNAME
        [-survey_dir SURVEY_DIR] [-delete] [-update_survey]
        [-add_project]

Script to make a survey of specific project

optional arguments:
  -h, --help              show this help message and exit
  -dbhostname DBHOSTNAME  MySQL hostname
  -dbuser DBUSER          MySQL user name
  -dbpass DBPASS          MySQL password
  -dbname DBNAME          Name of the database with project
                          cvsanaly information
  -survey_dir SURVEY_DIR  Survey directory
  -delete                 Delete automatically the directory
                          if exist
  -update_survey          Only update survey App
  -add_project            Add project to the survey
```

[1] https://github.com/ccervigon/survey_creator

Como puede observarse es obligatorio en la llamada indicar tanto el usuario como la contraseña de la base de datos MySQL donde está alojada la información extraída del repositorio del proyecto, así como el nombre de la base de datos del proyecto del que se desea hacer la encuesta. Y como parámetros opcionales se debe indicar donde está alojada la base de datos MySQL, por defecto *localhost*, el directorio donde queremos que se cree la encuesta, por defecto */tmp*, y una serie de *flags* según queramos que el programa borre automáticamente sin preguntar todo lo contenido en la carpeta donde se creará la encuesta (**delete**), actualice los archivos de la encuesta sin borrar las gráficas y así evitar tener que volver a crearlas (**update_survey**) o añadir un nuevo proyecto a la encuesta y así poder emplear la misma página web para más de un proyecto (**add_project**).

Por lo que un ejemplo de ejecución del programa empleando los diferentes flags sería:

- Crear encuesta:

```
python survey_script.py -dbuser tthebosss -dbpass XXX
   -dbname project_cvsanaly -survey_dir /tmp/surveys
```

- Crear de nuevo la encuesta:

```
python survey_script.py -dbuser tthebosss -dbpass XXX
   -dbname project_cvsanaly -survey_dir /tmp/surveys
   -delete
```

- Actualizar encuesta:

```
python survey_script.py -dbuser tthebosss -dbpass XXX
   -dbname project_cvsanaly -survey_dir /tmp/surveys
   -update
```

- Añadir proyectos:

```
python survey_script.py -dbuser tthebosss -dbpass XXX
   -dbname project_cvsanaly -survey_dir /tmp/surveys
   -add_project
```

Al ejecutar el script como se ha indicado anteriormente, se realizará una copia de la página web en el directorio indicado. Una vez se ha copiado todo, el programa llama al script **analysis_project_author.py**, este se encarga de generar los gráficos con la evolución del esfuerzo realizado por cada desarrollador durante la vida del proyecto y almacenarlos en el directorio de la encuesta. Cuando ya se tienen todas las gráficas se procede a rellenar la base de datos sqlite de la encuesta con el nombre, email, código hash único (con el cual crear un acceso personalizado a la encuesta sin necesidad de introducir el nombre o email), nombre del proyecto (para el caso de emplear la misma base de datos

para más de un proyecto) y la gráfica asociada a cada desarrollador. Al finalizar el script se dispondrá de una encuesta completamente funcional sobre el proyecto elegido. Tan sólo será necesario enviar los correos a los desarrolladores y alojarla en una dirección web.

Para realizar el envío de los emails se ha elaborado el script **sending_mail_to_developers.py**. Al igual que **survey_script.py** necesita ser llamado de una manera concreta:

```
usage: sending_mail_to_developers.py [-h] -email EMAIL
       -email_pw EMAIL_PW -smtp SMTP -smtp_port SMTP_PORT
       -survey_dir SURVEY_DIR -url URL

Script to send an email to developers of an specific project

optional arguments:
  -h, --help            show this help message and exit
  -email EMAIL          Email
  -email_pw EMAIL_PW    Email password
  -smtp SMTP            SMTP server
  -smtp_port SMTP_PORT  SMTP port
  -survey_dir SURVEY_DIR  Survey directory
  -url URL              URL where the survey is hosted
```

Como puede observarse es necesario llamar al programa con los siguientes parámetros obligatorios: dirección de correo electrónico, contraseña del correo, servidor SMTP de salida, puerto SMTP, directorio donde se encuentra la encuesta, para acceder a la base de datos de la cual obtener los datos de cada uno de los desarrolladores, y la url donde está alojada.

Una vez ejecutado el script los desarrolladores del proyecto recibirán un email personalizado como el siguiente:

Dear DESARROLLADOR,

I am a researcher at a Spanish University and together with another university in the UK we are working on a model for estimating effort in Open Source Software.

To validate our findings we need your input.
Therefore we have built a small survey -it is composed of 6 questions and should take you less than 3 minutes to respond- about your effort and time spent on the PROYECTO project.

The survey can be accessed at
http://ccervigon.libresoft.es/CÓDIGO_HASH_DEL_AUTOR

For more information about ourselves, our research, and the possibility to give more feedback, please visit http://ccervigon.libresoft.es/contact or find us on the IRC (#libresoft on Freenode).

We have already studied OpenStack with our methodology.
Have a look at the preliminary results in http://ccervigon.libresoft.es/result.

Thank you in advance.

Carlos Cervigón
Researcher at Universidad Rey Juan Carlos
GSyC/LibreSoft Libre Software Engineering Research Lab
http://www.libresoft.es

Cuadro 6.3: Ejemplo de email enviado a los desarrolladores.

6.2.3. Diseño

La web está dividida en 6 páginas diferentes.

Todas ellas se basan en una plantilla básica que incluye: el título de la web con una imagen con enlace a la página de Libresoft, un menú a la izquierda para poder acceder a las diversas páginas que componen la web (salvo a la propia encuesta), un pie de página con información sobre el copyright y enlaces, también en forma de imagen, a las paginas web de las universidades involucradas (la Universidad Rey Juan Carlos y la Universidad de Brunel) y en el centro el contenido de cada una de las diferentes páginas.

Estas diferentes páginas que componen la web son las siguientes:

- Home

  Que incluye el formulario de acceso mediante el nombre o email del desarrollador, así como una corta explicación del motivo de la encuesta y su extensión. Puede ver el aspecto de esta página en la Figura 6.1.

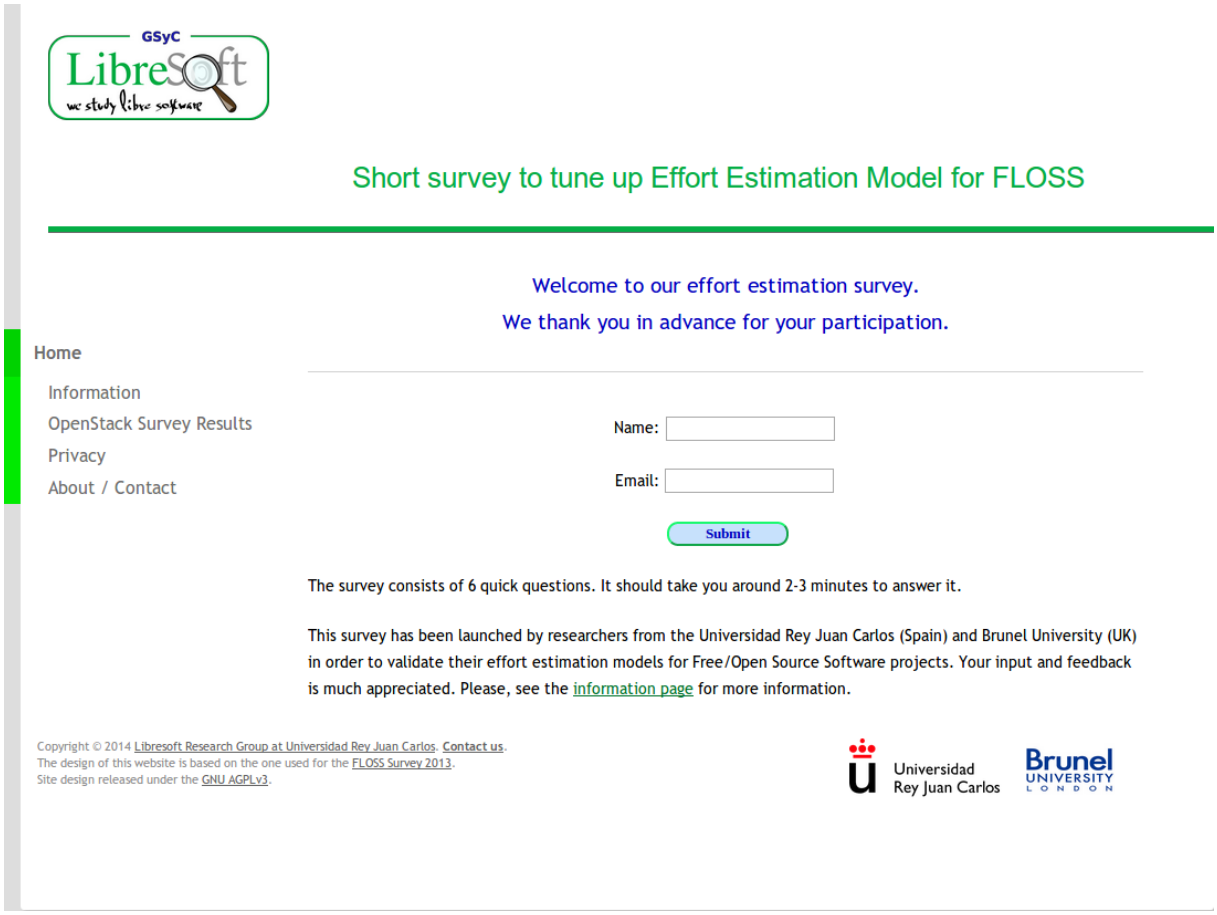


Figura 6.1: Página inicial de la web.

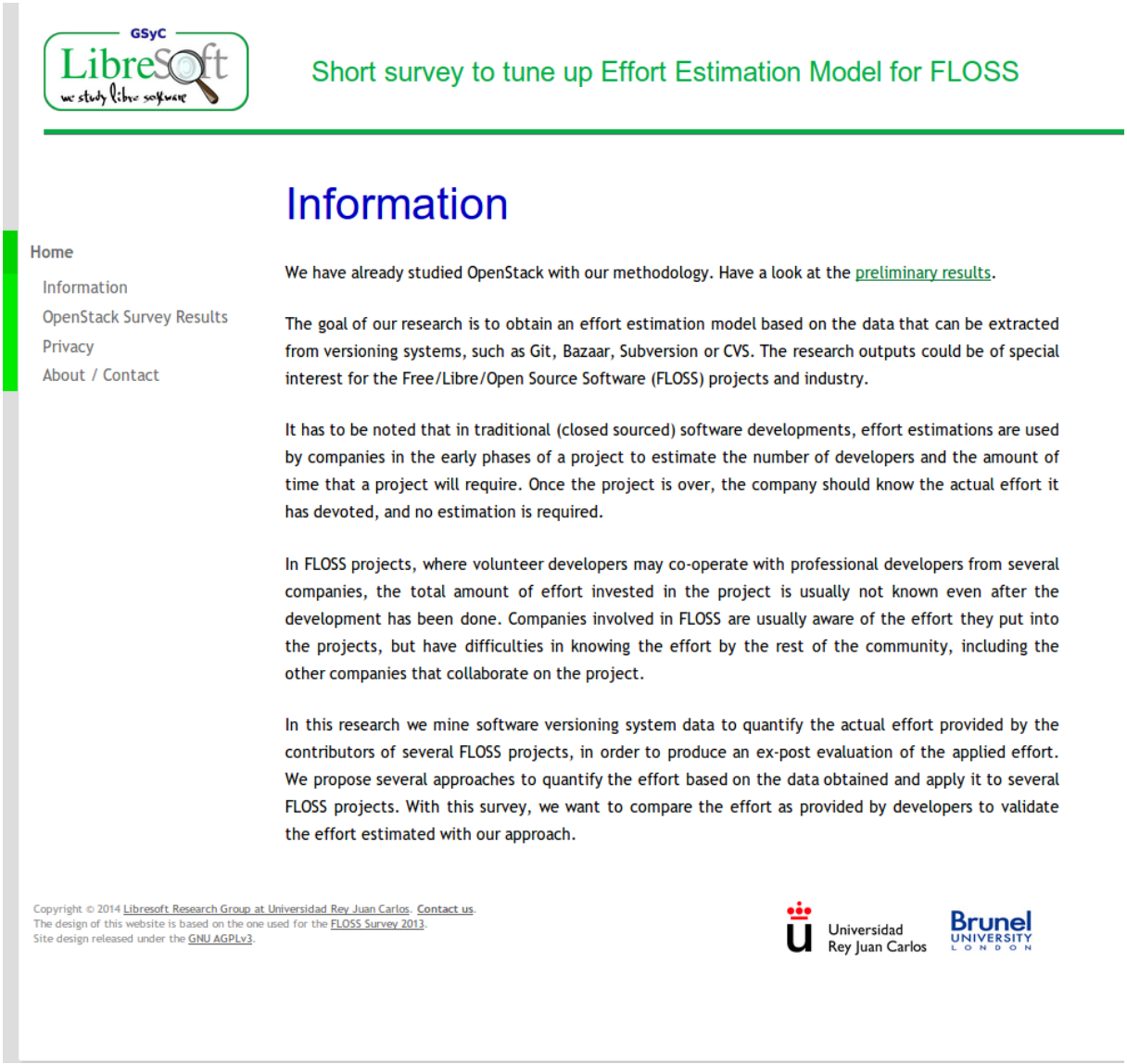


Figura 6.2: Página con información sobre la investigación.

- Information

  En esta página se explica de forma más detallada el objetivo de la encuesta, así como la causa que promueve la investigación del modelo de estimación de esfuerzo que se trata en este proyecto. Puede verse en la Figura 6.2.

- Results

  En esta página se presenta un resumen de los resultados obtenidos del análisis y prueba del modelo de estimación de esfuerzo con los datos obtenidos de la encuesta que se realizó a OpenStack. En ella como puede verse en la Figura 6.3, se ha incrustado una corta presentación con estos resultados empleando la herramienta de SlideShare[2].

[2] `http://www.slideshare.net/`

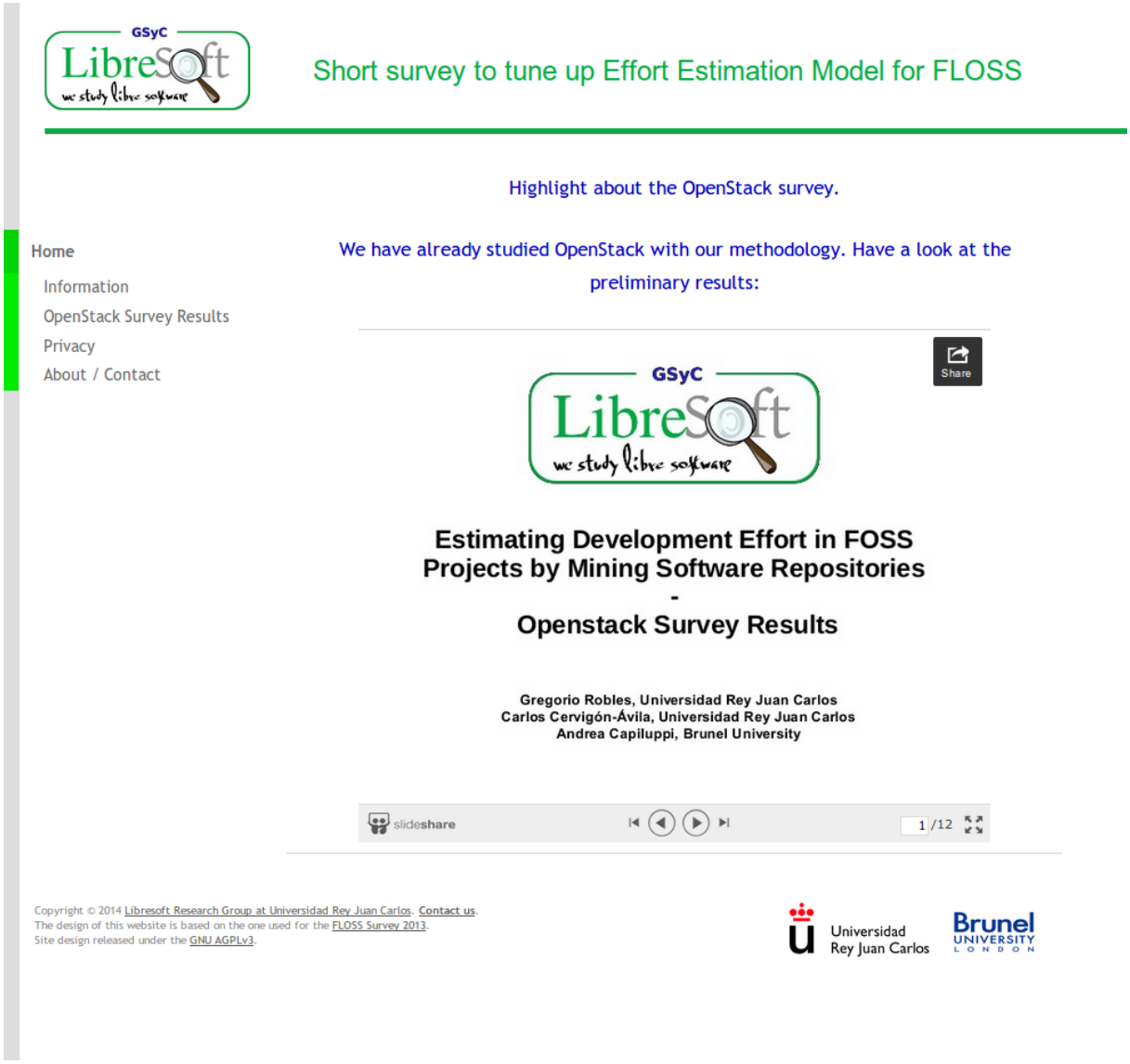


Figura 6.3: Página con los resultados de encuestas anteriores.

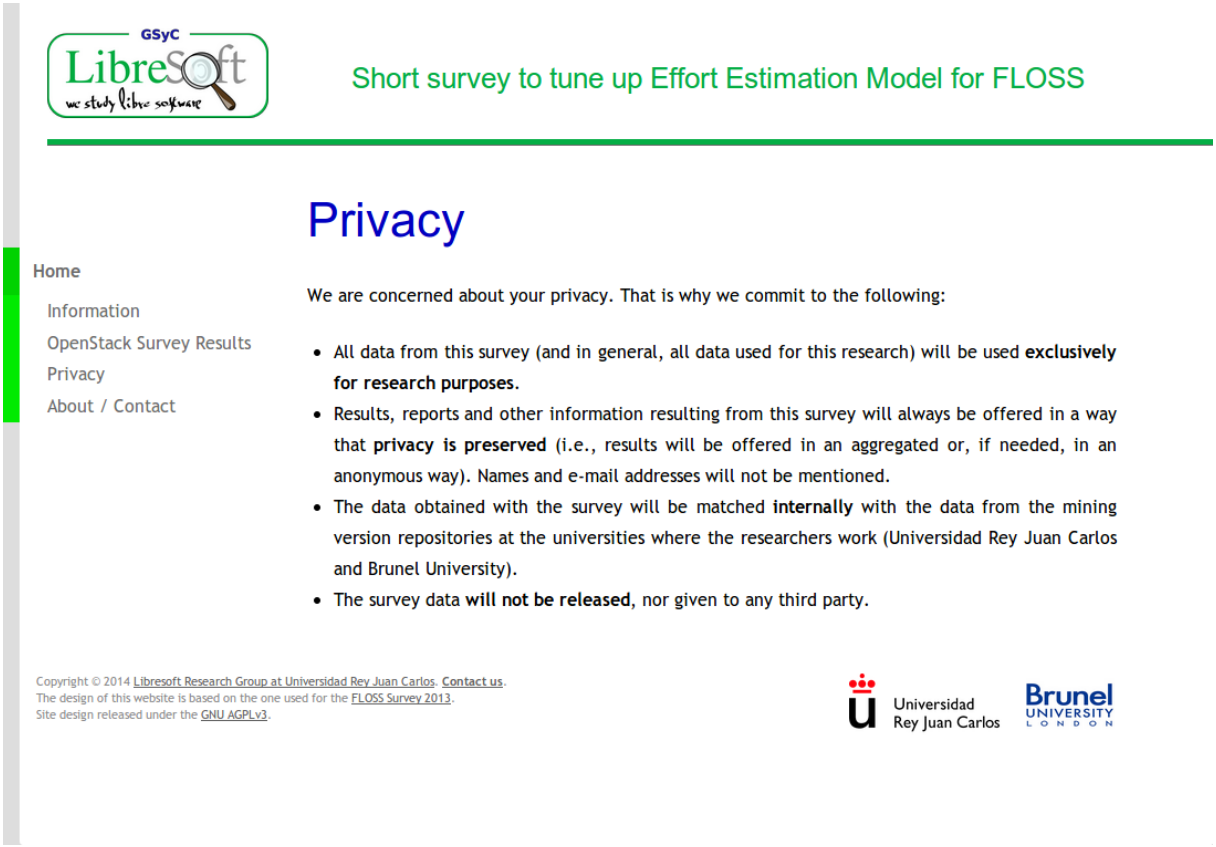


Figura 6.4: Página con información de privacidad de la encuesta.

- Privacy

  En este apartado de la página web, como puede observarse en la Figura 6.4, se explica que los datos obtenidos de la encuesta no serían compartidos con terceros, serían empleados únicamente con el equipo encargado de la investigación del modelo de estimación de esfuerzo, sólo serían empleados en propósitos de investigación y que los resultados obtenidos y publicados de la investigación serían anónimos.

- About/Contact

  En caso de que los desarrolladores quisieran obtener más información, pueden visitar el apartado que vemos en la Figura 6.5. En este apartado pueden obtener diversas vías de contacto con el equipo de investigación, tanto empleando correo electrónico, como mediante un canal IRC.

  Así como una breve explicación de las universidades involucradas en la investigación y enlace a diversas páginas con más información sobre los investigadores participantes en el proyecto.

- Survey

  Una vez que los desarrolladores acceden a la encuesta, tanto si emplea su código hash

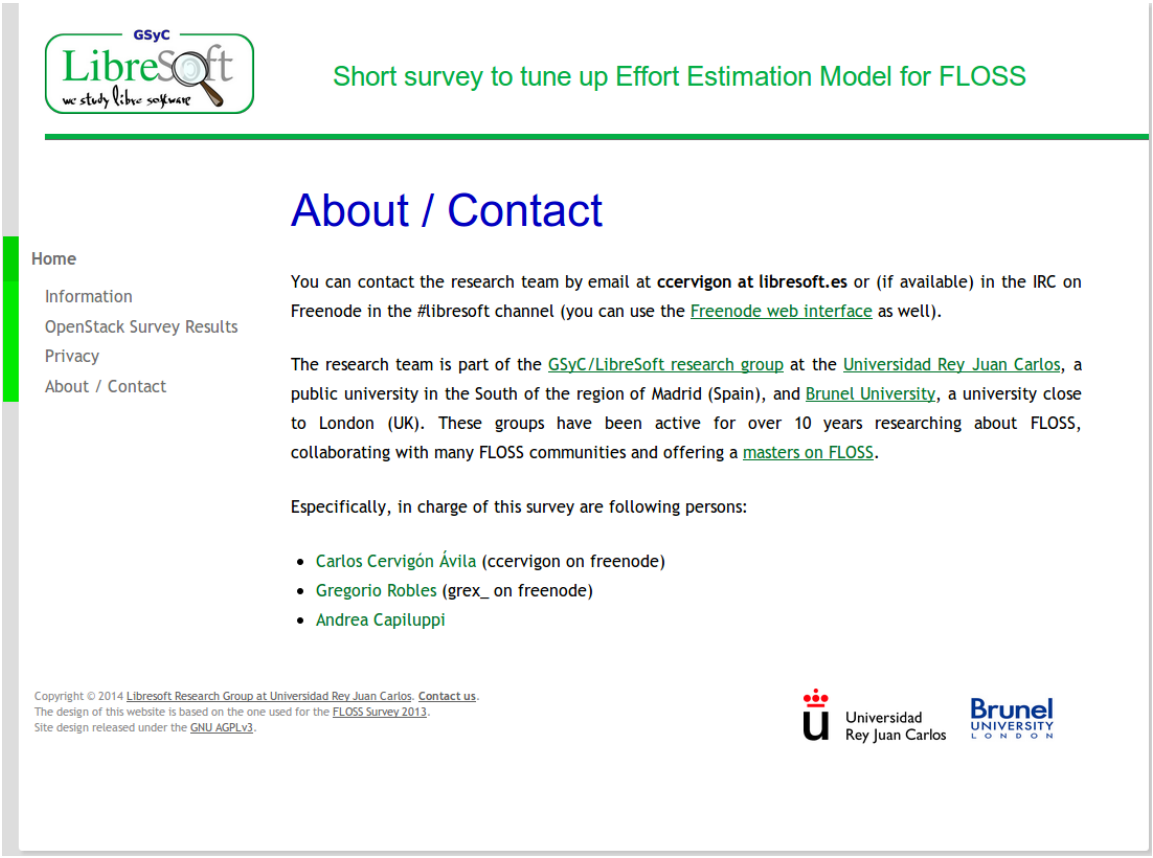


Figura 6.5: Página con información de contacto.

privado como si usa su nombre e email, acceden a la página de la Figura 6.6. En esta página verá las 5 preguntas realizadas sobre su trabajo en el proyecto investigado, una casilla de marcación por si desea que le enviemos un correo con los resultados de la investigación una vez obtenidos y una gráfica en la que está representada su actividad en días a lo largo del proyecto, con el fin de facilitarle el trabajo a la hora de contestar sobre el esfuerzo realizado a lo largo del tiempo en dicho proyecto.

6.2.4. Soporte a desarrolladores

Durante los días siguientes al envío de la encuesta a los desarrolladores se les habilitó varias vías de ayuda para solventar las dudas e inconvenientes que pudieran surgirles, por ejemplo, fallos esporádicos a la hora del envío de la encuesta rellenada (provocados por problemas de saturación de acceso simultáneo en la base de datos), o por necesidad/curiosidad de conocer la manera en la que se había conseguido su nombre e email. Estas vías empleadas fueron mediante la cuenta de email y un canal IRC en Freenode. Ambas podían encontrarlas en el apartado About/Contact de la página web.

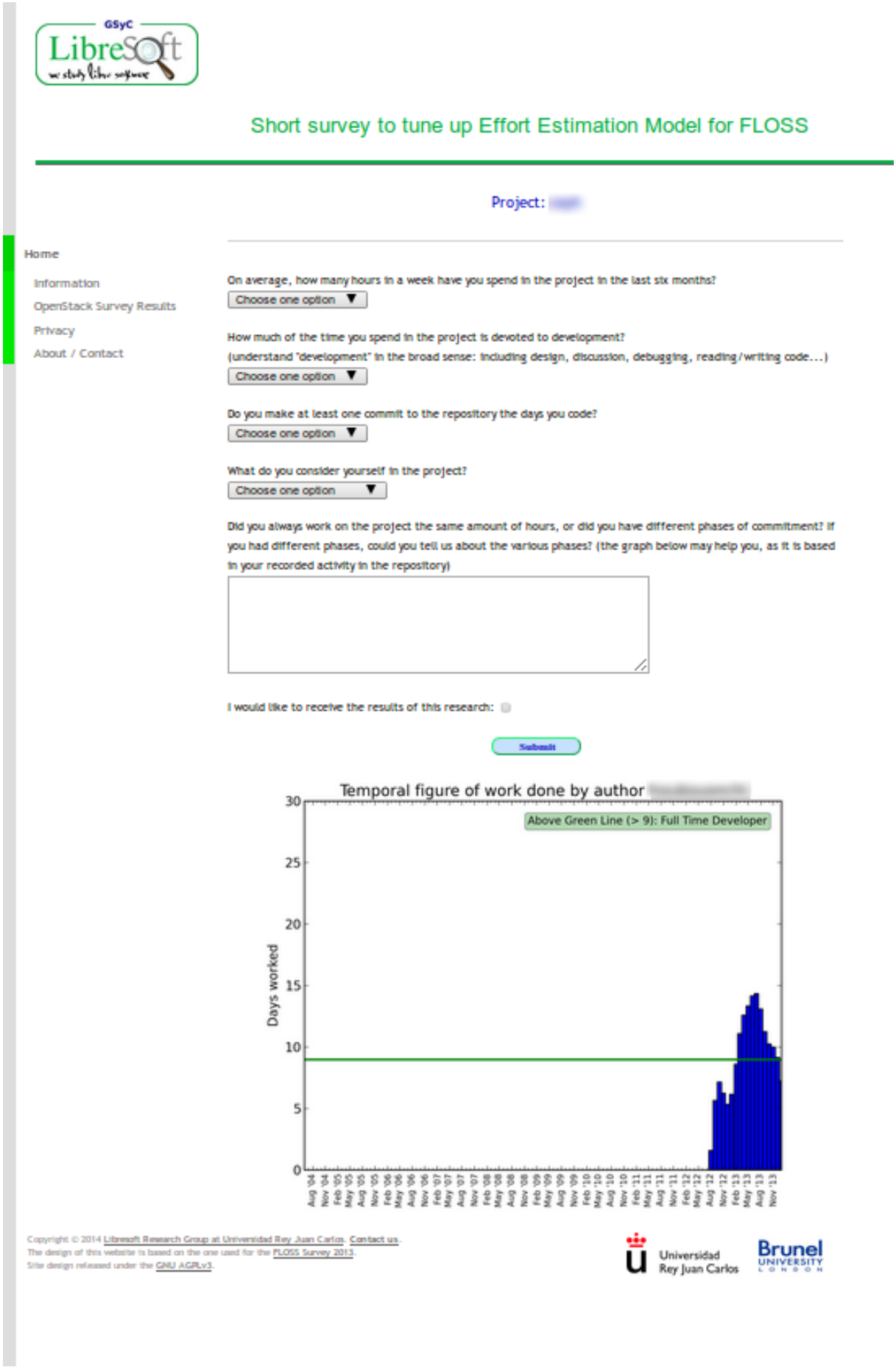


Figura 6.6: Página con las preguntas de la encuesta.

6.2.5. Otras consideraciones

Debido a que la elaboración de la encuesta iba a realizarse por varias personas, se decidió alojarla en un repositorio público en GitHub. Así mediante un sistema de control de versiones como es Git, se ha podido hacer cambios en el código de manera simultánea, lo cual ha facilitado en gran medida un desarrollo rápido de la capa HTML y CSS. En la Figura 6.7 y la Figura 6.8 podemos ver una captura del repositorio y un fragmento del historial de cambios.

6.3. Resultados de las encuestas

6.3.1. Método de análisis

Para analizar las respuestas primero se ha comprobado la representatividad de la muestra obtenida mediante el test de pruebas de rango con signo de Wilcoxon[3]. Siendo la hipótesis nula en el test la no existencia de diferencia entre ambas poblaciones. Por lo que, cuando el valor p obtenido sea mayor que el nivel de significación elegido, concluiremos que se acepta la hipótesis nula y por tanto, no hay diferencia entre poblaciones. Esto es necesario porque no se ha elegido aleatoriamente a los desarrolladores encuestados, sino que se ha realizado la petición de participación en la encuesta al total de desarrolladores y cada uno libremente decidía si deseaba participar o no.

Una vez comprobada la representatividad de los resultado obtenidos se ha procedido a probar sucesivamente diferentes valores frontera de commits para clasificar a los desarrolladores en profesionales o no. Una vez clasificados los desarrolladores, se compara con los resultados de la encuesta para así poder validar la precisión de la clasificación.

Este análisis emplea cuatro indicadores con los que medir los resultados:

- **True positive (tp)**. Son los desarrolladores que se han clasificado correctamente como profesionales.
- **False negative (fn)**. Desarrolladores clasificados erróneamente como no-profesionales.
- **True negative (tn)**. Desarrolladores clasificados correctamente como no-profesionales.
- **False positive(fp)**. Desarrolladores clasificados erróneamente como profesionales.

Para la validación de la bondad de los diferentes valores frontera empleando los indicadores anteriores se han utilizado las siguientes métricas:

[3] `http://es.wikipedia.org/wiki/Prueba_de_los_rangos_con_signo_de_Wilcoxon`

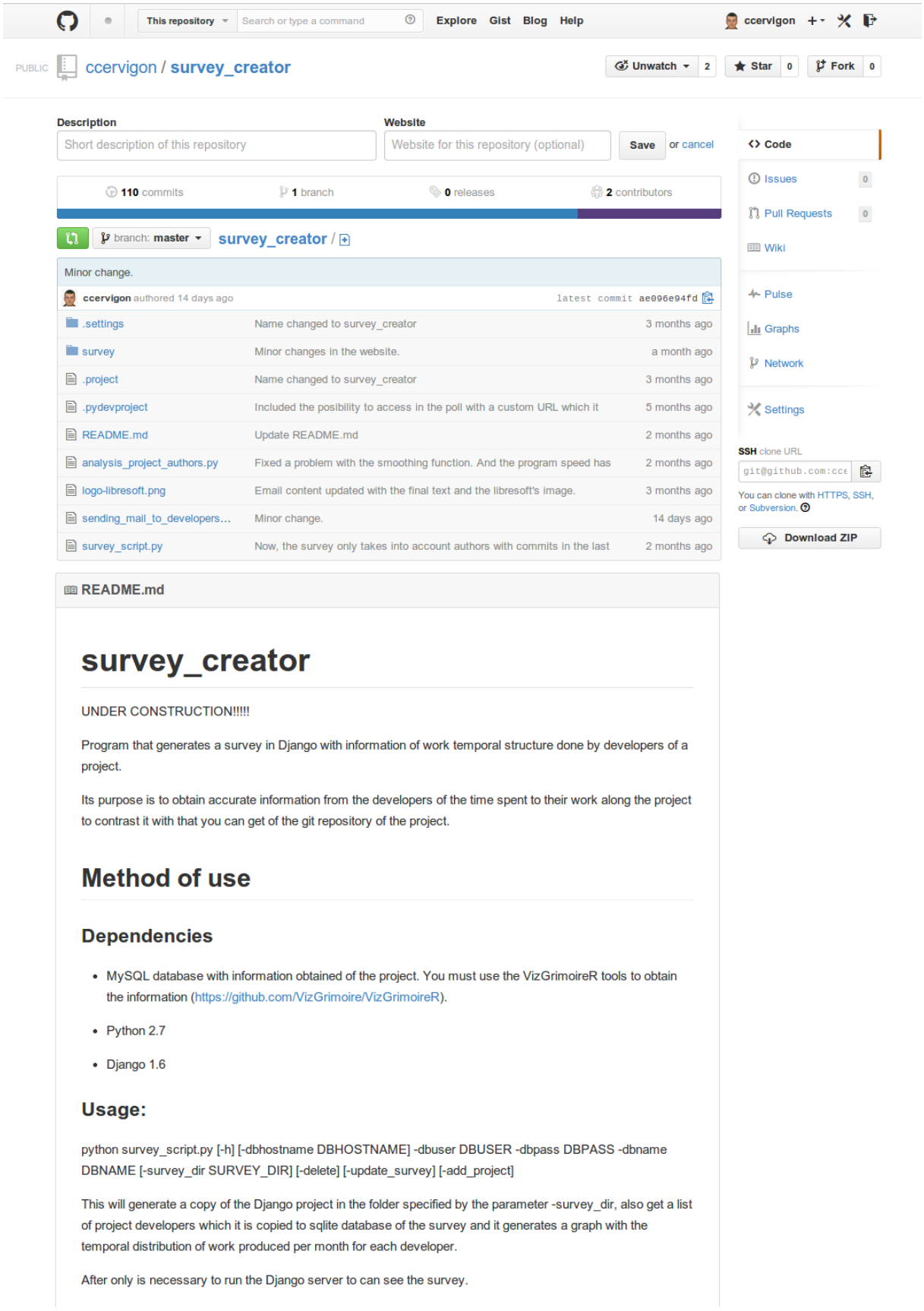


Figura 6.7: Repositorio de la encuesta en GitHub.

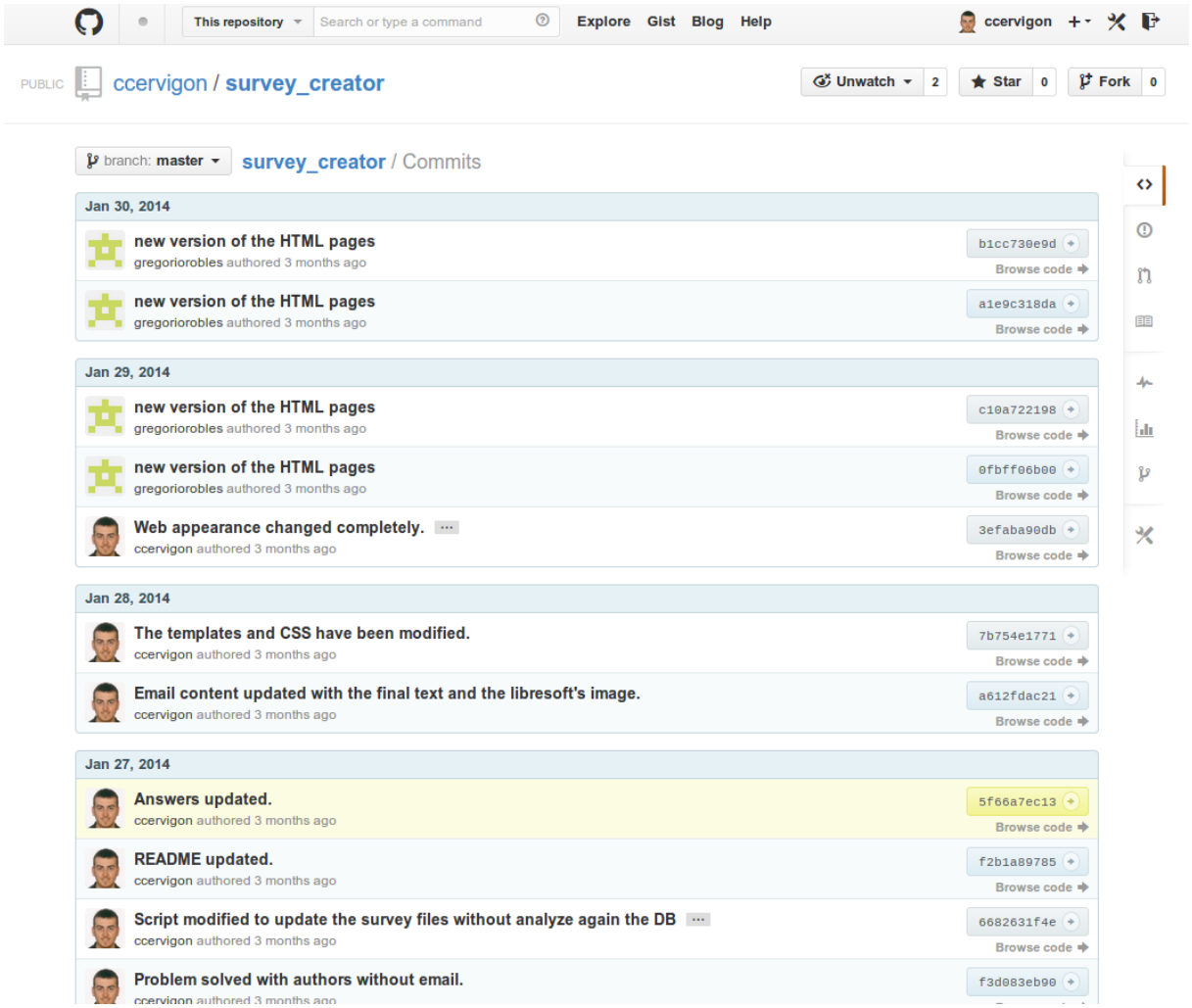


Figura 6.8: Fragmento de commits realizados en la encuesta.

- **Precisión y exhaustividad**. Más conocido por su nombre en inglés *precision and recall*[4].

- **F-measure**[5]. Como medida de evaluación de resultados que combina mediante una media harmónica los valores de precisión y exhaustividad.

- **Precisión (Accuracy)**. Porcentaje de aciertos (tanto de clasificación en profesionales como no-profesionales) sobre el total de clasificaciones realizadas.

$$Accuracy = \frac{t_p + t_n}{t_p + t_n + f_p + f_n}$$

- **Bondad (Goodness)**. Medida que tiene un cuenta la compensación entre falsos positivos y falsos negativos.

$$Goodness = \frac{|(t_p + f_p) - (t_n - f_n)|}{t_p + f_p + f_n}$$

6.3.2. Encuesta a OpenStack

La primera encuesta que se realizó para obtener datos de desarrolladores con los que validar nuestros último método fue con OpenStack. En esta encuesta realizada a los desarrolladores se enviaron **1407** emails, de los cuales:

- Alrededor de **300** fueron devueltos por la inexistencia de la cuenta.

- **131** desarrolladores contestaron a la encuesta.

- **5** respuestas fueron removidas debido a que estaban vacías, o se podía observar de su respuesta en el cuadro de texto libre, que el desarrollador no había entendido la encuesta o la había malinterpretado.

- **7** respuestas fueron corregidas debido a incongruencias entre varias de sus respuestas, por ejemplo, indicar que ha dedicado 40 horas al proyecto pero se considera desarrollador ocasional en él.

Representatividad

Lo primero que se ha tenido que hacer es comprobar la representatividad de la muestra. En la Figura 6.9 se muestra dos box-plots con el análisis de la actividad en los últimos 6 meses tanto de la población que ha respondido a la encuesta, como de la población total activa.

[4] `http://es.wikipedia.org/wiki/Precisión_y_exhaustividad`
[5] `http://en.wikipedia.org/wiki/F1_score`

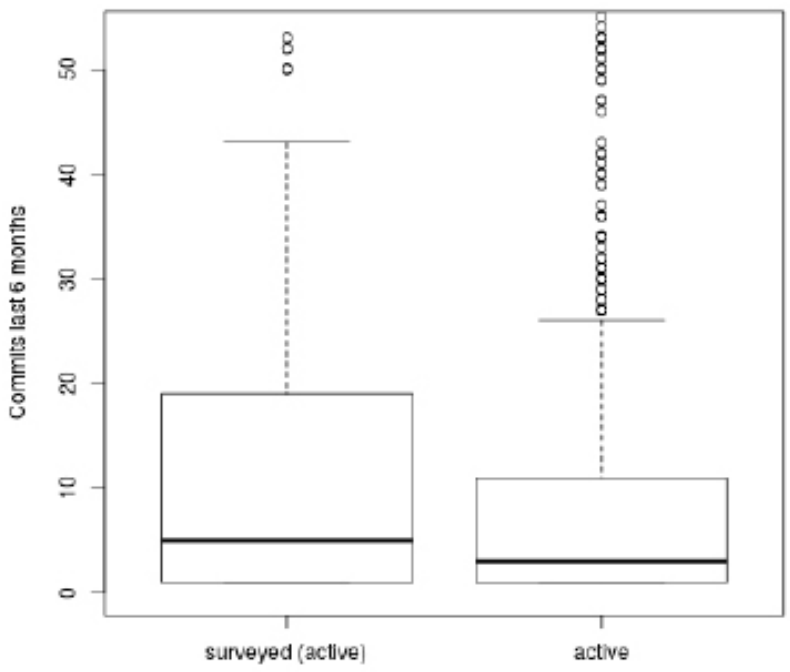


Figura 6.9: Box-plot con la distribución de los commits de la encuesta a OpenStack.

Y en la Tabla 6.4 podemos observar el resultado de aplicar el test de Wilcoxon eligiendo sólo a los desarrolladores con un mínimo de commits.

Como podemos observar para un valor de significación de 0.05, si incluimos a todos los desarrolladores que han realizado la encuesta tengan o no actividad en los últimos 6 meses, el test nos indica que la muestra no es representativa. Esto ocurre debido a que al incluir a los desarrolladores inactivos estamos introduciendo mucha interferencia. Pues la inclusión de antiguos desarrolladores produce una transformación severa. Pero como nuestro modelo está basado en la actividad, la existencia de respuestas de desarrolladores sin actividad en los últimos 6 meses no es ningún problema, sino un acto solidario, por lo

Commits	Población	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	p-value
0 ó más	Toda	0.00	0.00	1.00	9.62	6.00	491.00	0.005063
	Encuesta	0.00	0.00	4.00	14.12	14.00	201.00	
1 ó más	Toda	1.00	1.00	3.00	13.76	11.00	491.00	0.2008
	Encuesta	1.00	1.00	5.00	16.35	19.00	201.00	
5 ó más	Toda	5.00	7.00	13.00	29.37	30.00	491.00	0.2822
	Encuesta	5.00	8.50	16.50	29.38	34.75	201.00	
8 ó más	Toda	8.00	12.00	19.00	37.77	41.00	491.00	0.4171
	Encuesta	8.00	13.00	22.00	35.57	42.75	201.00	
11 ó más	Toda	11.00	14.75	24.00	44.34	52.00	491.00	0.9797
	Encuesta	11.00	14.00	26.00	38.83	50.00	201.00	

Tabla 6.4: Test de representatividad para diferentes valores mínimos de commits de la encuesta a OpenStack.

que tan sólo llevaremos a cabo la clasificación con los desarrolladores activos. Por lo que la muestra en este caso es representativa del total del proyecto.

Resultados

Los resultados para valores de *threshold* entre 1 y 50 de cada uno de los indicadores, así como de las métricas empleadas a partir de estos indicadores, pueden observarse en la Tabla 6.5 y de manera gráfica en la Figura 6.10.

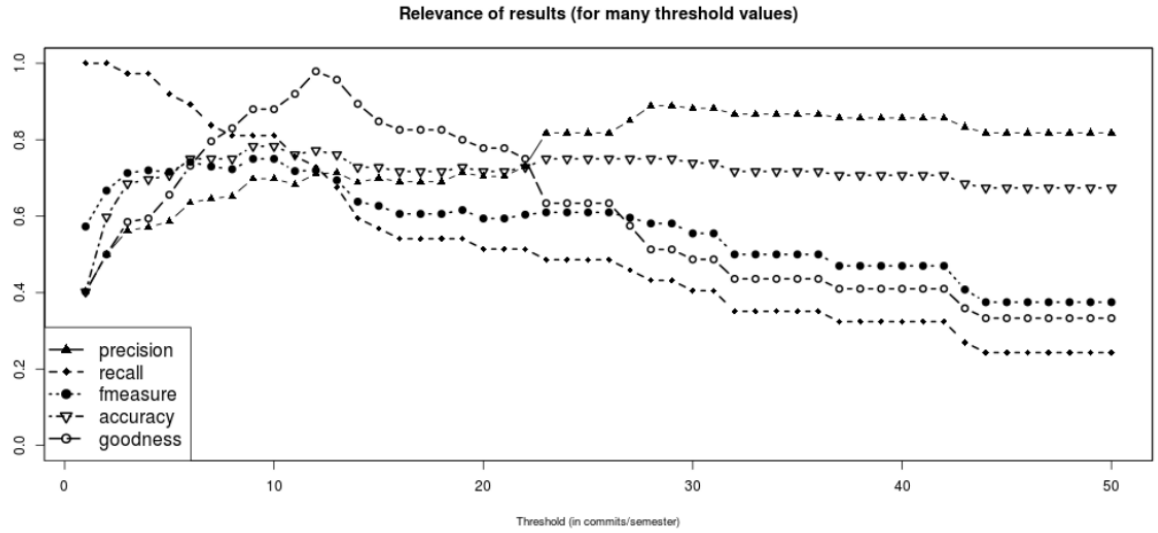


Figura 6.10: Figura de la evolución de los indicadores y métricas para cada valor de t.

Como se observa, los mejores resultados se obtienen en el intervalo de valores de t entre 9 y 13, obteniendo un pico en el valor de *goodness* en t=12. Esto es debido a que en dicho valor de t, aunque no se esté clasificando a todos los desarrolladores profesionales como tal, el error que se comete al clasificar erróneamente tanto desarrolladores profesionales como no-profesionales se compensa, como se puede ver en la Figura 6.11, y por tanto este valor será el que con más precisión se estimaría el esfuerzo en el proyecto.

Una vez tenemos la franja de valores de *threshold* óptimos para realizar una estimación de esfuerzo en el proyecto, vamos a observar cual es el resultado que obtenemos en Personas-Mes al emplearlos. Estos valores de esfuerzo pueden observarse en total y por semestres en la Tabla 6.6. Como podemos comprobar el valor de esfuerzo en el proyecto para los valores entre 9 y 13 es la mitad que si considerásemos un valor de t=1, es decir, valorar a todos los desarrolladores como profesionales.

Si observamos el último semestre para t=12, el número de Personas-Semestre ha sido de 2634, es decir, unas 440 Personas-Mes. Con este valor de esfuerzo se ha preguntado a miembros de OpenStack sobre su fiabilidad, el cual les ha sonado bastante razonable, pues

threshold	tp	fn	tn	fp	precision	recall	fmeasure	accuracy	goodness
1	37	0	0	55	0.402	1.0	0.573	0.402	0.402
2	37	0	18	37	0.5	1.0	0.667	0.598	0.5
3	36	1	27	28	0.563	0.973	0.713	0.685	0.585
4	36	1	28	27	0.571	0.973	0.72	0.696	0.594
5	34	3	31	24	0.586	0.919	0.716	0.707	0.656
6	33	4	36	19	0.635	0.892	0.742	0.75	0.732
7	31	6	38	17	0.646	0.838	0.73	0.75	0.796
8	30	7	39	16	0.652	0.811	0.723	0.75	0.83
9	**30**	**7**	**42**	**13**	**0.698**	**0.811**	**0.75**	**0.783**	**0.88**
10	**30**	**7**	**42**	**13**	**0.698**	**0.811**	**0.75**	**0.783**	**0.88**
11	**28**	**9**	**42**	**13**	**0.683**	**0.757**	**0.718**	**0.761**	**0.92**
12	**27**	**10**	**44**	**11**	**0.711**	**0.73**	**0.72**	**0.772**	**0.979**
13	**25**	**12**	**45**	**10**	**0.714**	**0.676**	**0.694**	**0.761**	**0.957**
14	22	15	45	10	0.688	0.595	0.638	0.728	0.894
15	21	16	46	9	0.7	0.568	0.627	0.728	0.848
16	20	17	46	9	0.69	0.541	0.606	0.717	0.826
17	20	17	46	9	0.69	0.541	0.606	0.717	0.826
18	20	17	46	9	0.69	0.541	0.606	0.717	0.826
19	20	17	47	8	0.714	0.541	0.616	0.728	0.8
20	19	18	47	8	0.704	0.514	0.594	0.717	0.778
21	19	18	47	8	0.704	0.514	0.594	0.717	0.778
22	19	18	48	7	0.731	0.514	0.604	0.728	0.75
23	18	19	51	4	0.818	0.486	0.61	0.75	0.634
24	18	19	51	4	0.818	0.486	0.61	0.75	0.634
25	18	19	51	4	0.818	0.486	0.61	0.75	0.634
26	18	19	51	4	0.818	0.486	0.61	0.75	0.634
27	17	20	52	3	0.85	0.459	0.596	0.75	0.575
28	16	21	53	2	0.889	0.432	0.581	0.75	0.513
29	16	21	53	2	0.889	0.432	0.581	0.75	0.513
30	15	22	53	2	0.882	0.405	0.555	0.739	0.487
31	15	22	53	2	0.882	0.405	0.555	0.739	0.487
32	13	24	53	2	0.867	0.351	0.5	0.717	0.436
33	13	24	53	2	0.867	0.351	0.5	0.717	0.436
34	13	24	53	2	0.867	0.351	0.5	0.717	0.436
35	13	24	53	2	0.867	0.351	0.5	0.717	0.436
36	13	24	53	2	0.867	0.351	0.5	0.717	0.436
37	12	25	53	2	0.857	0.324	0.47	0.707	0.41
38	12	25	53	2	0.857	0.324	0.47	0.707	0.41
39	12	25	53	2	0.857	0.324	0.47	0.707	0.41
40	12	25	53	2	0.857	0.324	0.47	0.707	0.41
41	12	25	53	2	0.857	0.324	0.47	0.707	0.41
42	12	25	53	2	0.857	0.324	0.47	0.707	0.41
43	10	27	53	2	0.833	0.27	0.408	0.685	0.359
44	9	28	53	2	0.818	0.243	0.375	0.674	0.333
45	9	28	53	2	0.818	0.243	0.375	0.674	0.333
46	9	28	53	2	0.818	0.243	0.375	0.674	0.333
47	9	28	53	2	0.818	0.243	0.375	0.674	0.333
48	9	28	53	2	0.818	0.243	0.375	0.674	0.333
49	9	28	53	2	0.818	0.243	0.375	0.674	0.333
50	9	28	53	2	0.818	0.243	0.375	0.674	0.333

Tabla 6.5: Resultados de los indicadores y métricas para cada valor de t.

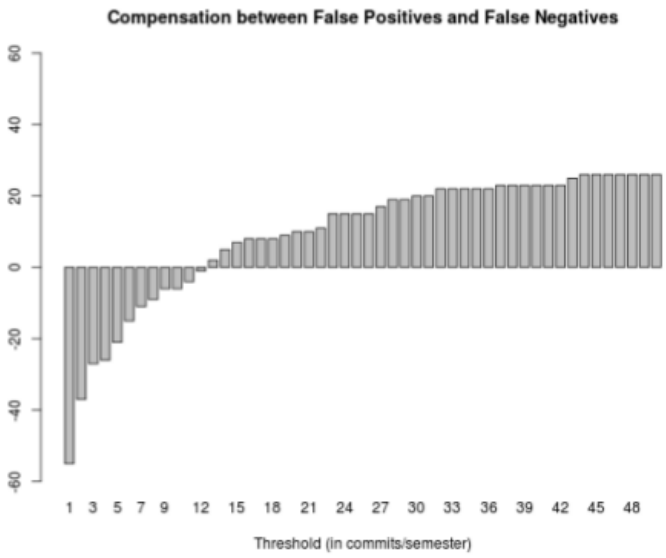


Figura 6.11: Gráfica de compensación de desarrolladores clasificados erróneamente.

t	Effort	10s2	11s1	11s2	12s1	12s2	13s1	13s2
1	17400	426	816	1242	1890	2742	4368	5916
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
9	9683	345	721	866	1065	1493	2189	3003
10	9298	337	711	841	1025	1434	2086	2864
11	8957	330	700	819	987	1381	1997	2743
12	**8655**	**324**	**690**	**800**	**955**	**1334**	**1919**	**2634**
13	8370	318	680	782	924	1289	1847	2531

Tabla 6.6: Estimación total de esfuerzo (en PM), para varios valores de t. Valor de estimación de esfuerzo para todos los semestres.

actualmente la estimación de la OpenStack Foundation es de que existen alrededor de 250 desarrolladores profesionales trabajando en el proyecto pagados por empresas. Por lo que podemos corroborar cómo este método de estimación de esfuerzo no sólo es más realista y preciso, sino que el error que se comete al queda limitado al que pudiera introducirse a la hora de estimar erróneamente el esfuerzo realizado en contribuciones esporádicas.

6.3.3. Encuesta realizada a Linux kernel, WebKit, Moodle, Mediawiki y Ceph

Posteriormente a la encuesta a los desarrolladores de OpenStack, se realizó una encuesta conjunta a los desarrolladores de Linux kernel, WebKit, Moodle, Mediawiki y Ceph. A diferencia de la encuesta a OpenStack, sólo se envió la encuesta a los desarrolladores que habían tenido actividad en los últimos 12 meses, con esto se consigue evitar la aportación de gran parte de los desarrolladores que ya no están ligados al proyecto, y por tanto no tienen actividad, consiguiendo eliminar en gran medida la distorsión que sufríamos en la encuesta a OpenStack con las aportaciones solidarias.

También se prolongó el tiempo de recogida de datos a 2 semanas en lugar de 1, enviando al inicio de la segunda semana un email de recordatorio para recordar amablemente la participación en la encuesta.

Por lo que en total en esta encuesta se enviaron **4584** emails (135 a Ceph, 3187 a Linux, 152 a Moodle, 449 a Mediawiki y 661 a WebKit), de los cuales:

- **363** fueron devueltos por la inexistencia de la cuenta.
- **937** desarrolladores contestaron a la encuesta.
- **49** respuestas fueron removidas (28 de Linux, 6 de Mediawiki, 10 de Moodle y 5 de WebKit) por respuestas repetidas, entradas vacías y malinterpretación de la encuesta.
- **66** respuestas fueron corregidas (1 de Ceph, 48 de Linux, 2 de Moodle, 9 de Mediawiki y 6 de WebKit) debido a incongruencias entre varias de sus respuestas.

Por lo que después de las eliminaciones de respuestas no válidas y sin tener en cuenta los emails inexistentes (debido a que los emails fueron enviados sin distinguir proyecto), la participación queda en:

- **Ceph**: 24 / 135 (**17,78 %**)
- **Linux**: 652 / 3187 (**20,46 %**)

- **MediaWiki**: 94 / 449 (**20,94 %**)
- **Moodle**: 42 / 152 (**27,63 %**)
- **WebKit**: 85 / 661 (**12,86 %**)

Representatividad

Siguiendo los mismos pasos que en la encuesta a OpenStack, vamos a proceder a comprobar la representatividad de la muestra de los proyectos analizando el resultado de p-value del test de Wilcoxon con un grado de significación de 0,05.

De la Tabla 6.7, Tabla 6.8, Tabla 6.9, Tabla 6.10 y Tabla 6.11 podemos observar como la muestra, empleando las respuestas con actividad, es representativa para los proyectos Ceph, Moodle, MediaWiki y WebKit, pero no para Linux. Esto puede deberse a varios factores, cómo un filtrado no muy exhaustivo en busca de errores en las respuestas de autores con baja actividad. Mientras que si no tenemos en cuenta a los autores con más de 1 commits de actividad, es decir, no a los que han tenido un único aporte al proyecto, la muestra es representativa.

Ceph

Commits	Población	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	p-value
1 ó más	Toda	1.00	1.00	2.50	58.86	24.00	1320.00	0.07583
	Encuesta	1.00	2.00	10.00	147.40	144.20	1320.00	

Tabla 6.7: Test de representatividad de la encuesta a Ceph.

Moodle

Commits	Población	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	p-value
1 ó más	Toda	1.00	3.00	7.00	52.65	23.00	702.00	0.2741
	Encuesta	1.00	5.00	9.00	55.69	21.00	664.00	

Tabla 6.8: Test de representatividad de la encuesta a Moodle.

MediaWiki

Commits	Población	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	p-value
1 ó más	Toda	1.00	1.00	5.00	48.92	36.00	865.00	0.4422
	Encuesta	1.00	1.00	6.50	33.26	29.00	568.00	

Tabla 6.9: Test de representatividad de la encuesta a MediaWiki.

WebKit

Commits	Población	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	p-value
1 ó más	Toda	1.00	2.00	6.00	32.33	28.00	639.00	0.2127
	Encuesta	1.00	3.00	10.00	34.87	28.00	599.00	

Tabla 6.10: Test de representatividad de la encuesta a WebKit.

Linux

Commits	Población	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	p-value
1 ó más	Toda	1.00	1.00	3.00	12.61	10.00	457.00	0.02486
	Encuesta	1.00	1.00	3.00	16.19	15.00	256.00	
2 ó más	Toda	2.00	3.00	7.00	18.38	18.00	457.00	0.1556
	Encuesta	2.00	3.00	8.00	22.38	25.75	256.00	

Tabla 6.11: Test de representatividad de la encuesta a Linux.

Resultados

Una vez comprobada la representatividad de las muestras de cada uno de los proyectos vamos a buscar el valor de t óptimo que nos permitirá conseguir una estimación de esfuerzo precisa.

Al igual que en OpenStack, podemos ver en la figuras: Figura 6.12, Figura 6.13, Figura 6.14, Figura 6.15 y Figura 6.16, la evolución de los valores de los indicadores y las métricas empleadas, y la compensación que se obtiene al clasificar erróneamente desarrolladores profesionales y no-profesionales para cada uno de los threshold entre 1 y 50. Observando el valor de goodness en cada uno de los proyectos, los valores de threshold en el que se hace máximo son:

Proyecto	**threshold**
Ceph	24
Moodle	14
MediaWiki	29
WebKit	17
Linux	17

Tabla 6.12: Valores óptimos de threshold de los proyectos

Como puede observarse, estos valores son superiores en todos los casos al obtenido para OpenStack. Esto ocurre debido a que en OpenStack siguen un estricto sistema de revisión múltiple de los cambios, por lo que el número de commits medio por desarrollador es menor, y por tanto, el valor de threshold óptimo es inferior.

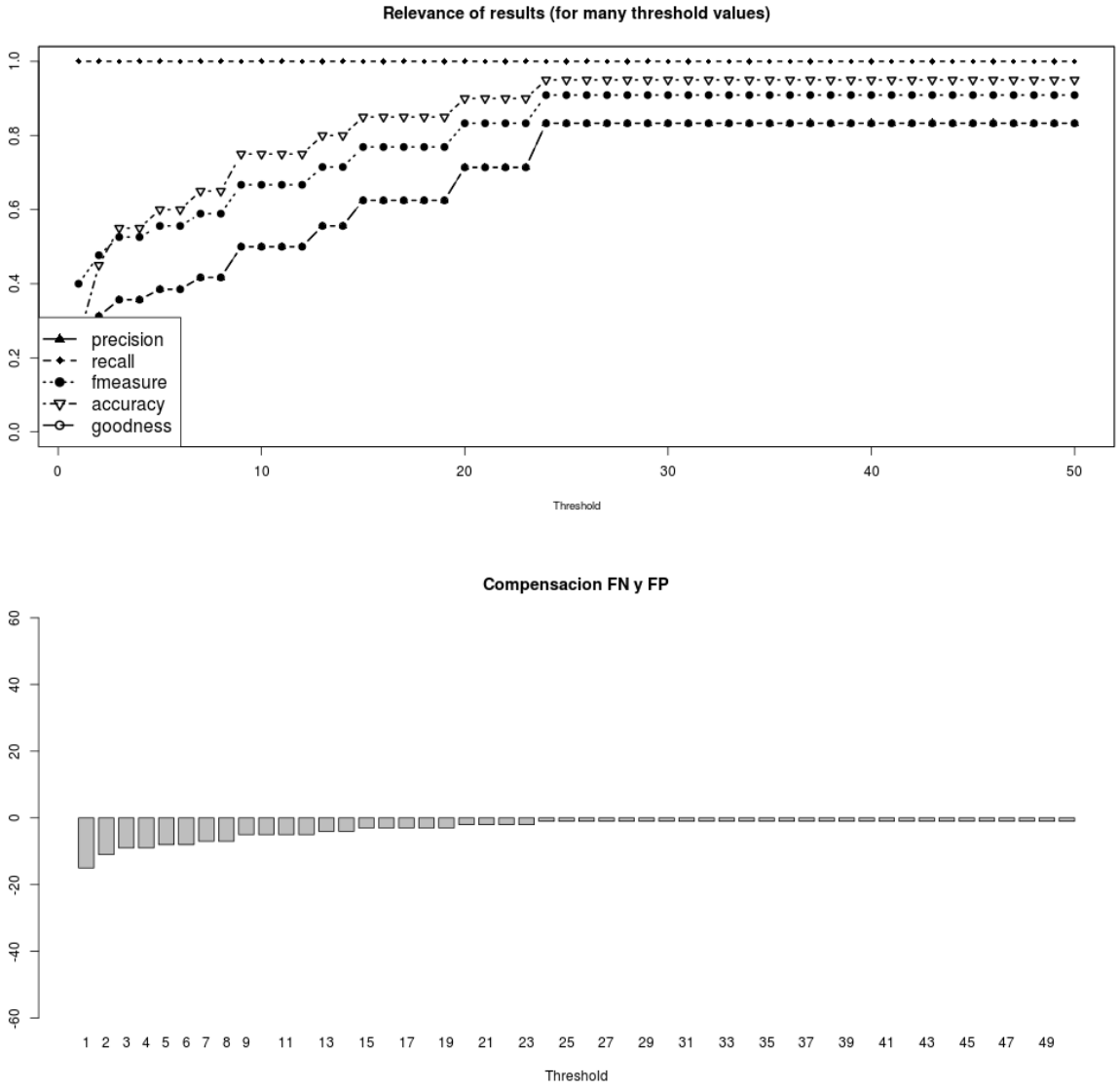


Figura 6.12: Figura de la evolución de los indicadores y métricas para cada valor de t. Y gráfica de compensación de desarrolladores clasificados erróneamente del proyecto Ceph.

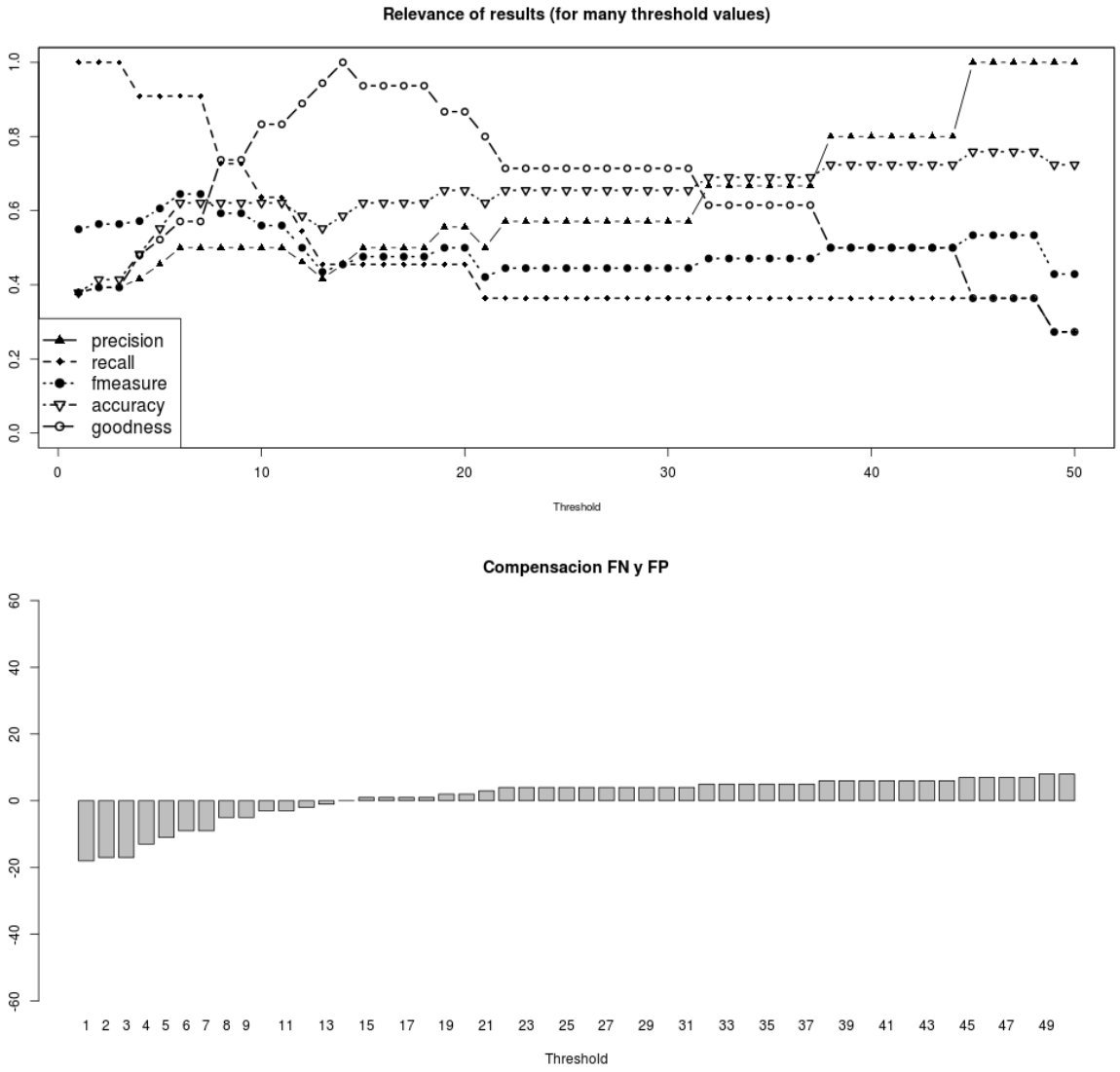


Figura 6.13: Figura de la evolución de los indicadores y métricas para cada valor de t. Y gráfica de compensación de desarrolladores clasificados erróneamente del proyecto Moodle.

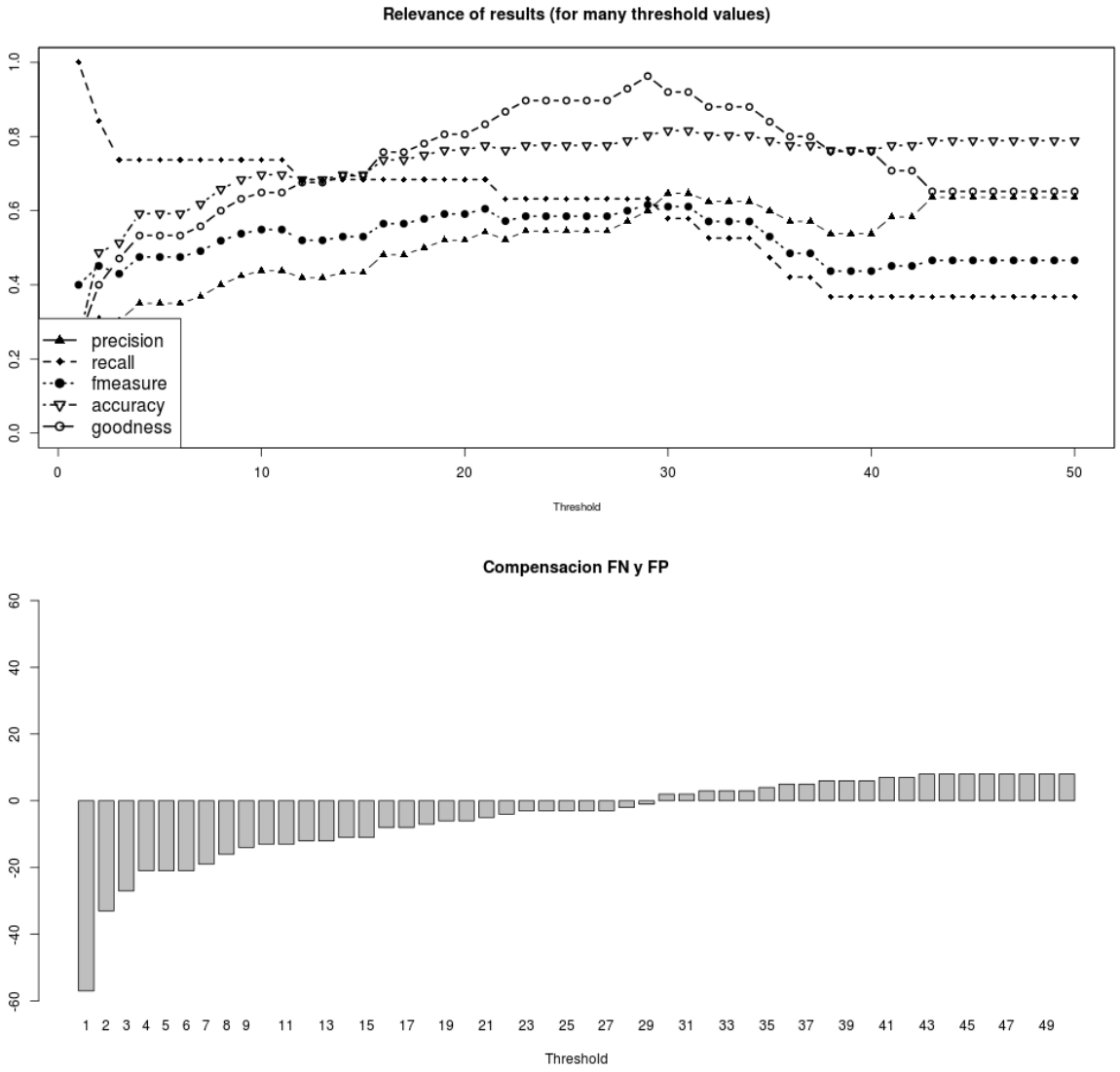


Figura 6.14: Figura de la evolución de los indicadores y métricas para cada valor de t. Y gráfica de compensación de desarrolladores clasificados erróneamente del proyecto MediaWiki.

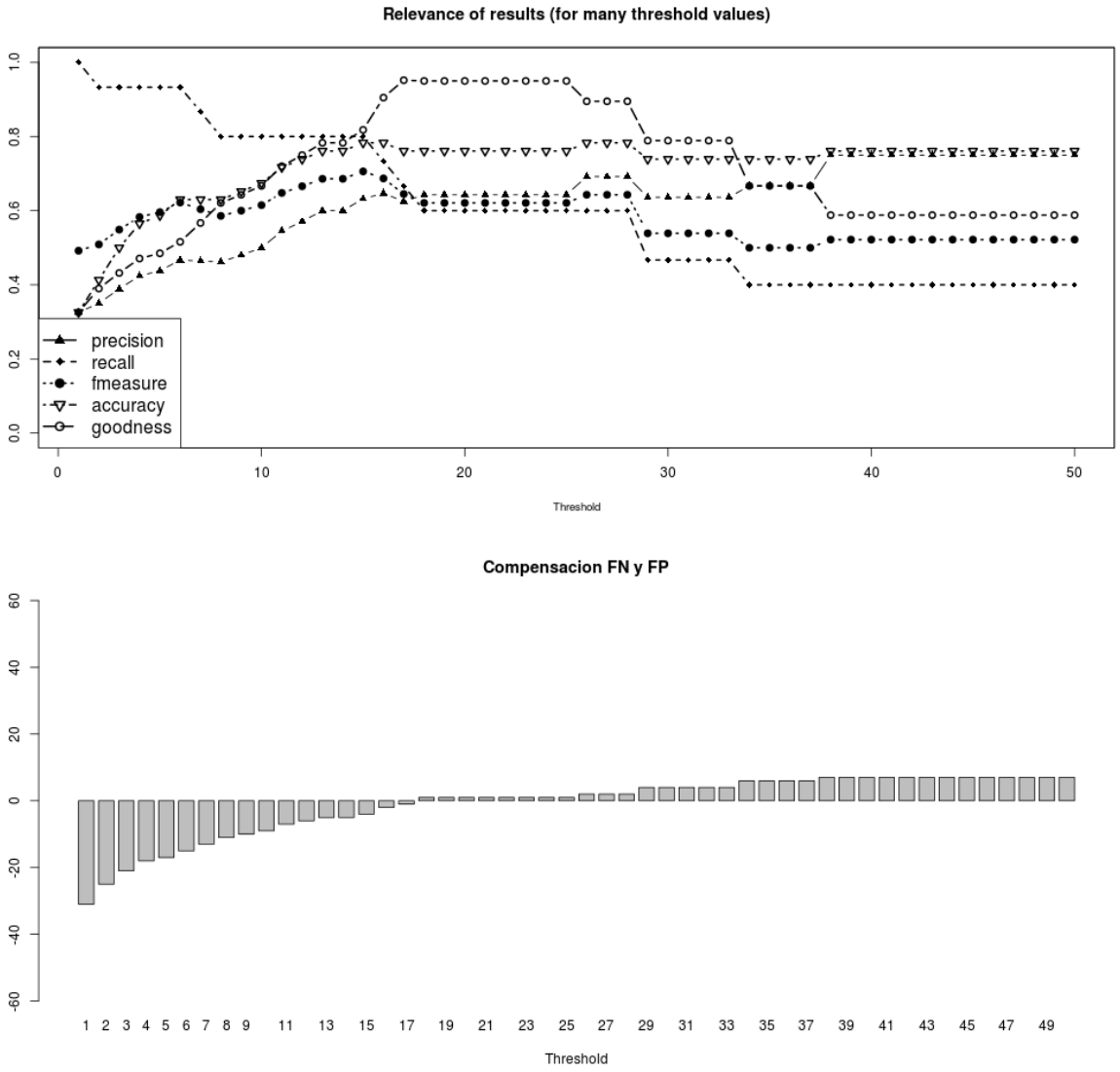


Figura 6.15: Figura de la evolución de los indicadores y métricas para cada valor de t. Y gráfica de compensación de desarrolladores clasificados erróneamente del proyecto WebKit.

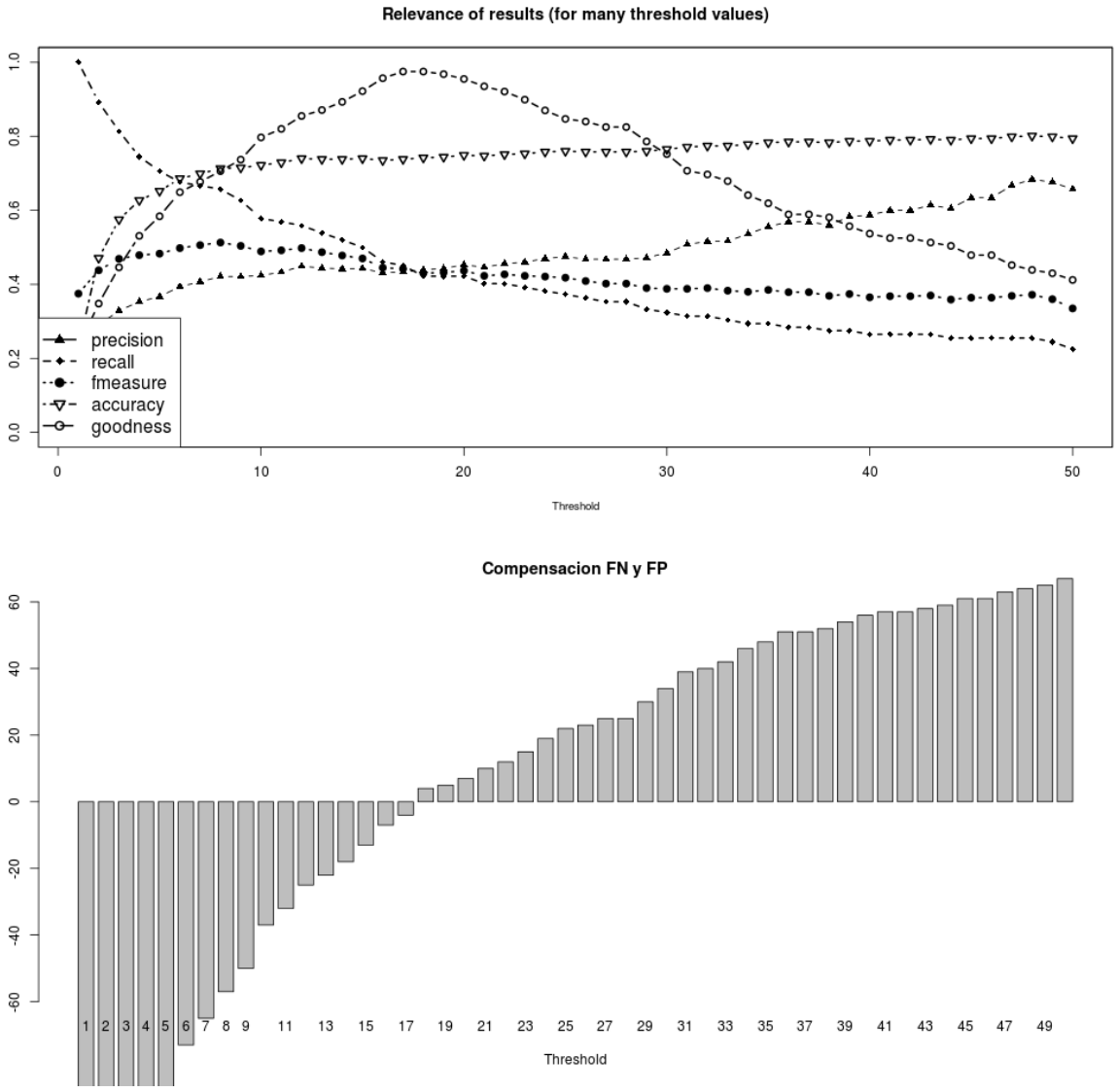


Figura 6.16: Figura de la evolución de los indicadores y métricas para cada valor de t. Y gráfica de compensación de desarrolladores clasificados erróneamente del proyecto Linux.

Ceph		Moodle		MediaWiki		WebKit		Linux	
t	**Esfuerzo**	**t**	**Esfuerzo**	**t**	**Esfuerzo**	**t**	**Esfuerzo**	**t**	**Esfuerzo**
1	1584	1	3162	1	7266	1	13260	1	64710
24	732	14	1980	29	4075	17	8034	17	29041

Tabla 6.13: Estimación total de esfuerzo (en PM), para t igual a 1 y el valor óptimo.

Y si observamos los resultados de esfuerzo de la Tabla 6.13, realizados en cada uno de los proyectos para un valor de threshold de 1 y el valor óptimo. Comprobamos cómo el esfuerzo realizado para el valor óptimo comparado con el de t=1 se reduce a valores entorno a la mitad, fenómeno parecido al que ocurría para el caso de OpenStack.

Capítulo 7

Conclusiones y trabajos futuros

7.1. Logros

Después de varios meses de investigación, estudio y prueba de diversos métodos de estimación de esfuerzo, realización de varias encuestas con su respectivo soporte a desarrolladores y la elaboración de diversos *papers* con los resultados que se iban obteniendo, se ha podido confirmar como la minería de repositorios software es un excelente sistema de estimación de esfuerzo. Con tan sólo encontrar el valor de *threshold* óptimo para el proyecto con el que obtener los desarrolladores profesionales en cada uno de los meses, se puede obtener unos resultados de esfuerzo válidos y fiables. Es decir, este método es una herramienta simple y rápida, la cual no necesita una calibración compleja para empezar a obtener resultados que reflejen la realidad de los proyectos (a diferencia de otros métodos actuales). Además se podría reducir aún más el error que se comete y mejorar su precisión empleando nueva información obtenida de *Mailing Lists* o *Bugs Trackers*.

7.2. Trabajos Futuros

A partir del trabajo realizado en el proyecto y los resultados obtenidos de las encuestas, las principales hilos de trabajo futuro podrían resumirse en:

- Búsqueda del mecanismo de obtención del threshold óptimo del proyecto.
- Realizar un experimento científico para obtener margenes de error de la estimación realizada al clasificar autores como a tiempo completo.
- Investigar teniendo en cuenta varios proyectos FOSS, cual de las dos vías de medición de esfuerzo (commits o días activos) es más adecuada para el modelo.
- Comparar nuestros resultados de esfuerzo con los obtenidos empleando software de estimación tradicional, como puede ser COCOMO.

- Después de cuantificar el esfuerzo requerido en un proyecto FOSS, ¿qué es más rentable para un empresa?, ¿adoptar un modelo de rehacer su propio sistema?, ¿o invertir en un sistema de software libre existente?

- Comparar nuestro enfoque con técnicas de estimación de esfuerzo de proyectos FOSS previos, por ejemplo con el propuesto en [23] basado en la medición de la entropía para calcular el coste de mantenimiento del proyecto.

7.3. Lecciones aprendidas

Cuando empecé en el proyecto fin de carrera nunca antes había participado en un proyecto de investigación, por lo que participar en este me ha permitido aprender mucho sobre la forma de trabajar, la metodología empleada y el ritmo de trabajo. Así como gracias a la oportunidad dada por Gregorio de presentar en inglés nuestros avances a la comunidad en el BENEVOL 2013, darme cuenta de la importancia de adquirir un mejor manejo de este idioma tan importante hoy en día y sobre todo en este sector.

También he adquirido bastantes conocimientos sobre el manejo de bases de datos (MySQL y SQLite), pues en la doble titulación de Telecomunicación y Administración y Dirección de Empresas la asignatura referente a las bases de datos no se imparte, por lo que debido al alto empleo de las BBDD en el proyecto, por ejemplo, toda la información que extrae Metrics Grimoire es almacenada en MySQL, he tenido que aprender su funcionamiento de forma autónoma, y así poder trabajar rápida y fácilmente con ellas.

Otro conocimiento interesante que he tenido que aprender para trabajar en el proyecto ha sido el lenguaje estadístico R, debido al gran potencial que ofrece en el estudio estadístico de los resultados. Aunque el conocimiento del lenguaje ha sido básico, es más que suficiente para poder analizar la información obtenida de las encuestas y la elaboración de gráficas para una mejor representación.

Por último destacar el empleo de LaTeX. Un sistema de composición de texto muy diferente al comúnmente empleado Microsoft Word. El potencial del que se dispone empleando esta herramienta es increíble, por lo que la necesidad de uso para la elaboración de esta memoria, me ha permitido adquirir una fracción de conocimiento sobre él, pues como con cualquier lenguaje de programación, obtener un alto grado de conocimiento debido a la gran cantidad de extensiones que existen requiere muchos tiempo de uso.

7.4. Lecciones aplicadas de la carrera

De los conocimientos aprendidos en la carrera los más destacados que han sido empleados en el proyecto son los referentes al lenguaje de programación **Python**. Pues todos los scripts han sido elaborados con dicho lenguaje, además las herramientas de Metrics Grimoire tienen gran parte del código escrito en Python, permitiendo poder entender su funcionamiento más fácilmente. Y los conocimientos obtenidos en el empleo del framework **Django**, pues es la herramienta empleada para la elaboración de la encuesta enviada a los desarrolladores. Además que al emplear Python su integración con el resto de scripts no suponía la necesidad de realizar ningún cambio.

Otros conocimientos aprendidos y utilizados en el proyecto han sido:

- **El manejo de la shell de Linux**, pues me ha facilitado la tarea de mantenimiento de todos archivos empleados, el envío de información entre mi ordenador y la máquina virtual en la que se alojaba la encuesta y la creación de tareas programadas mediante el uso de crontab.
- **El empleo de SSH** para acceder a servidores virtuales. Pues como ya se ha indicado la encuesta se alojaba en una maquina virtual remota.
- **HTML y CSS**. Para la elaboración y maquetación de la página web.
- **Git**. Tanto para el análisis de los proyectos, como para crear el repositorio con el código de la encuesta y así permitir llevar un control de cambios y poder realizar dichos cambios por varios desarrolladores al mismo tiempo.

Apéndices

Apéndice A

Instalación de las herramientas empleadas

A.1. Python y modulos de Python necesarios

```
$ sudo apt-get install python2.7 python-psycopg2
python-matplotlib python-mysqldb python-pip
python-setuptools python-pysqlite2 python-storm
python-launchpadlib python-beautifulsoup
python-feedparser python-configglue
```

A.2. Django

```
$ sudo pip install Django==1.6.1
```

A.3. MySQL

```
$ sudo apt-get install mysql-server mysql-client
```

A.4. R y modulos de R necesarios

```
$ sudo apt-get install r-base r-cran-boot r-cran-class
r-cran-cluster r-cran-codetools r-cran-dbi r-cran-foreign
r-cran-kernsmooth r-cran-lattice r-cran-mass r-cran-matrix
r-cran-mgcv r-cran-nlme r-cran-nnet r-cran-rgl r-cran-rmysql
r-cran-rpart r-cran-spatial r-cran-survival
```

En el prompt de R (entrando como *sudo*):

```
> p<-c("ggplot2", "rjson", "optparse", "zoo", "ISOweek")
> install.packages(p)
> q()
```

A.5. Otros paquetes necesarios

```
$ sudo apt-get install sqlite3 flex automake1.11 git autoconf
make gawk
```

A.6. MetricsGrimoire y VizGrimoire

```
$ cd <home>/directorio
$ git clone https://github.com/MetricsGrimoire/CVSAnalY.git
$ git clone https://github.com/MetricsGrimoire/Bicho.git
$ git clone https://github.com/MetricsGrimoire/RepositoryHandler.git
$ git clone https://github.com/MetricsGrimoire/MailingListStats.git
$ git clone https://github.com/MetricsGrimoire/CMetrics.git
$ git clone https://github.com/VizGrimoire/VizGrimoireR.git
$ git clone https://github.com/VizGrimoire/VizGrimoireJS.git
$ git clone https://github.com/VizGrimoire/VizGrimoireUtils.git

$ cd <home>/directorio/RepositoryHandler
$ python setup.py build
$ sudo python setup.py install

$ cd ../CVSAnalY
$ python setup.py build
$ sudo python setup.py install

$ cd ../Bicho
$ python setup.py build
$ sudo python setup.py install

$ cd ../CMetrics
$ ./autogen.sh
$ make
$ make install

$ cd ../MailingListStats
$ python setup.py build
$ sudo python setup.py install

$ cd ../VizGrimoireR
$ sudo R CMD INSTALL vizgrimoire
```

Apéndice B

Función de Smoothing

Para realizar este método de cuantificación se ha empleado la función de *smoothing* disponible en la librería de NumPy de python[1], esta función implementa las siguientes ventanas con formas de ponderación diferente. Los tipos de ventana disponibles en la librería son: hanning[2], hamming[3], bartlett[4] y blackman[5], además empleamos una función más llamada flat que implementa un pulso rectangular unitario. Gráficamente estas ventanas las podemos comparar en la Figura B.1.

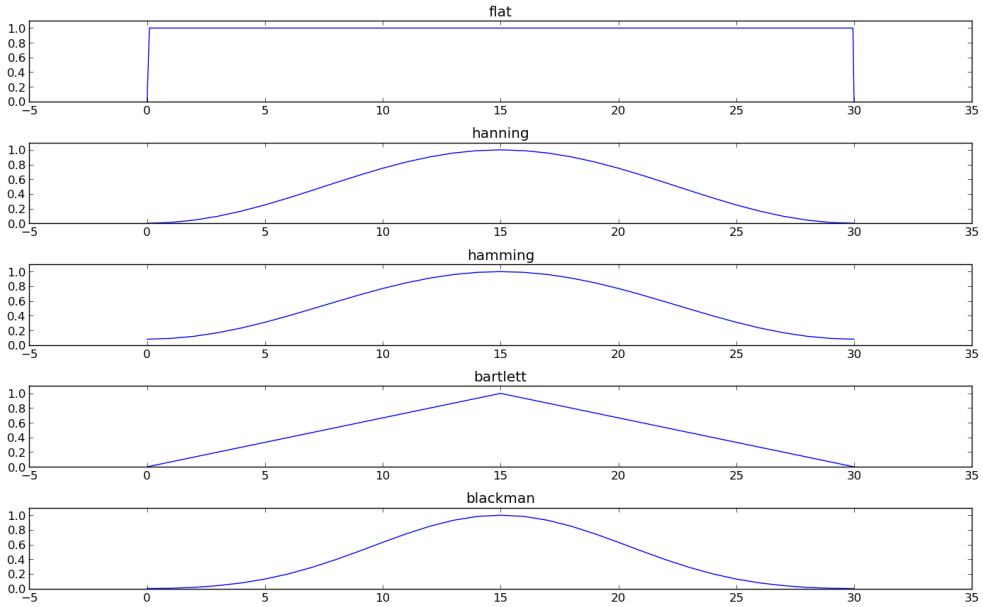


Figura B.1: Representación de las diferentes funciones de *smoothing*

[1] http://wiki.scipy.org/Cookbook/SignalSmooth
[2] http://docs.scipy.org/doc/numpy/reference/generated/numpy.hanning.html
[3] http://docs.scipy.org/doc/numpy/reference/generated/numpy.hamming.html
[4] http://docs.scipy.org/doc/numpy/reference/generated/numpy.bartlett.html
[5] http://docs.scipy.org/doc/numpy/reference/generated/numpy.blackman.html

B.1. Consideraciones previas sobre las funciones de smoothing

Lo primero que hay que tener en cuenta es que el vector de meses trabajados por desarrollador es una señal discreta y por tanto a la hora de elegir un tamaño de ventana sólo se puede emplear valores impares, pues sino se estaría valorando incorrectamente el trabajo realizado en cada muestra al no tener un único valor central correspondiente a la muestra principal.

Lo segundo que hay que tener en cuenta es que la función flat otorga el mismo peso tanto al mes analizado como a los contiguos seleccionados en la ventana. Este echo no es coherente, ya que aunque el esfuerzo realizado un mes es influido por los contiguos, este esfuerzo realizado en el mes central no tiene el mismo peso que los demás. Por tanto para nuestro caso podemos descartar dicha función de *smoothing*.

La tercera consideración a tener en cuenta es respecto a los tamaños de ventana de 1 y 3 muestras. Un tamaño de 1 muestra nos devuelve la función original por lo que no se estaría realizando ningún suavizado y por tanto no es válido. Y para un tamaño de 3 muestras ocurre exactamente lo mismo, pues como se puede observar en la Figura B.2, al ser una función discreta la ponderación que da a las 2 muestras adyacentes seleccionadas es cero, lo que devuelve la función original nuevamente. Aunque en la función hamming las muestras adyacentes la ponderación no es cero, se trata de un valor muy cercano a cero, lo que hace que el resultado no difiera significativamente y por tanto puede ser considerarlo también como cero.

La cuarta consideración y con cierta relación a la anterior indicación, las ventanas como ya se ha observado tienen un valor de ponderación en las muestras extremas de cero, por tanto cuando se elige un tamaño de ventana de 5 muestras en realidad sólo son efectivas 3. Aún así se va a seguir hablando del tamaño de ventana introducido en lugar del que realmente es efectivo, pues es el valor con el que se llama a las funciones de *smoothing* de la librería NumPy.

Y por último hay que indicar que la finalidad de aplicar una función de *smoothing* es la de suavizar picos y valles de trabajo esporádicos de la estructura temporal de trabajo de los desarrolladores, por tanto no se puede elegir ventanas muy grandes, pues se incurriría en el error de suavizar dicho reparto temporal del trabajo excesivamente, teniendo como caso extremo un valor continuo si se eligiera una ventana de tamaño igual al número de

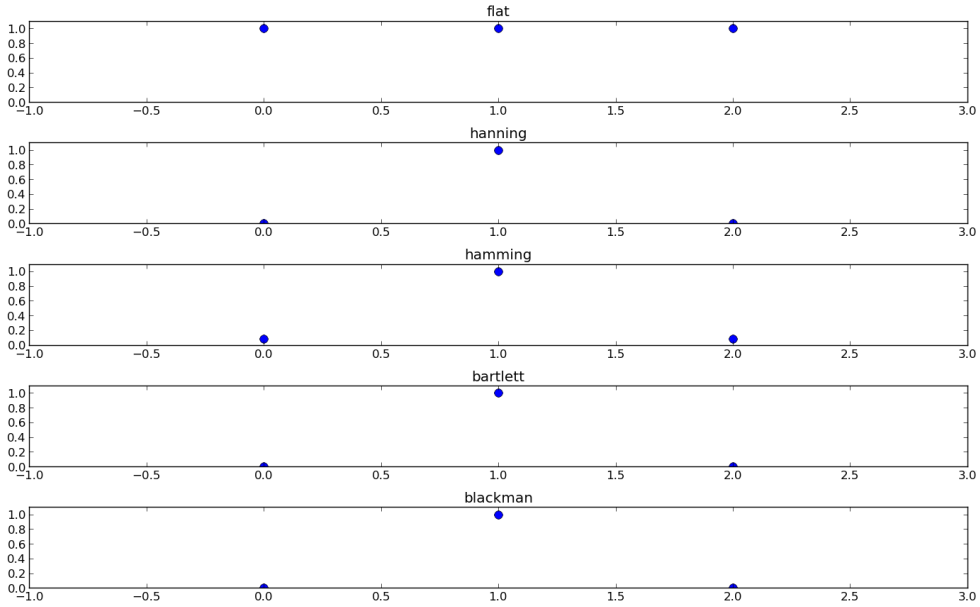


Figura B.2: Comparación de la ponderación efectuada en los diferentes tipos de ventana con un tamaño de ventana de 3 muestras

meses total del proyecto. Lo cual impediría diferenciar posibles períodos de trabajo a lo largo de la vida laboral del desarrollador en el proyecto.

Por tanto se tiene que buscar el o los tamaños de ventana que consigan suavizar picos y valles en el trabajo realizado por el desarrollador a lo largo del tiempo sin perder la estructura temporal principal de dicho trabajo.

B.2. Elección de tamaño y tipo de ventana

Para elegir el modelo óptimo lo primero que hay que hacer es localizar gráficamente entre que valores de tamaño de ventana conseguimos suavizar los picos y valles de trabajo sin llegar a perder la estructura principal del trabajo realizado por el desarrollador a lo largo del tiempo. Para seguidamente comparar numéricamente los resultados de los tamaños seleccionados para hacer la elección correcta.

Por ello primero hay que comparar para cada tipo de ventana y periodo temporal de aplicación el resultado obtenido de emplear los tamaños de ventana 5, 7, 9, 11 y 15.

Para hacer estas comparaciones se ha seleccionado al desarrollador que más commits ha realizado en el proyecto Tempest de Openstack. La Figura B.3 muestra el número de días que ha trabajado dicho desarrollador agrupado en meses sin aplicar ninguna función de *smoothing*.

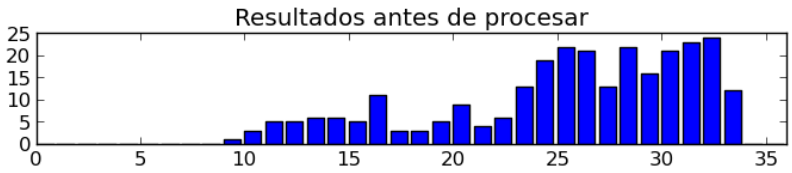


Figura B.3: Estructura del trabajo por meses sin aplicar la función de *smoothing*

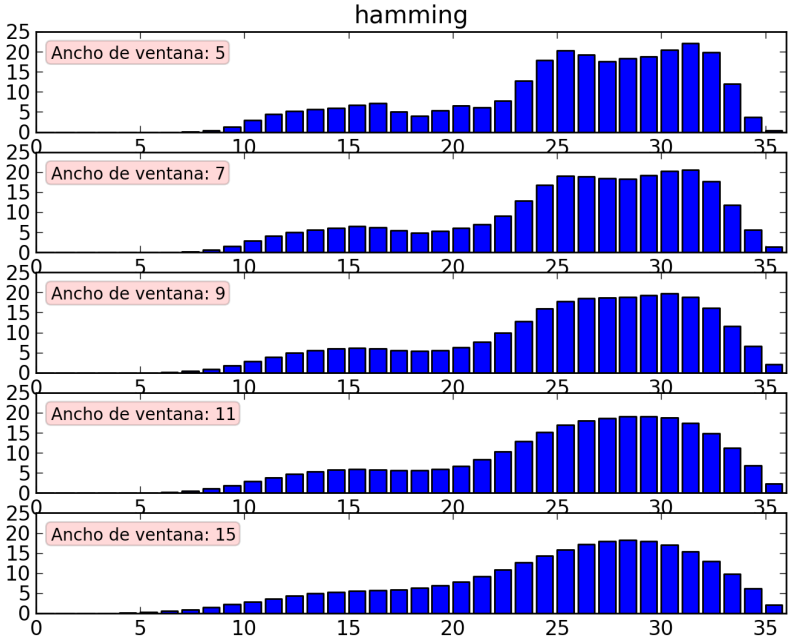


Figura B.4: Estructura del trabajo por meses al aplicar la función hamming

Y de la Figura B.4 a la Figura B.7 se muestra para cada tipo de ventana el resultado de aplicar los diferentes tamaños de ventana.

Si se observa las figuras se puede ver que hasta un ancho de ventana de 9 meses en todos los casos se puede diferenciar claramente las 2 etapas existentes en el trabajo del desarrollador en el proyecto. Mientras que para tamaños superiores, salvo el caso de la ventana blackman que todavía pueden diferenciarse los 2 periodos con claridad, se dificulta bastante poder elegir un valor claro de cambio de periodo. Mientras que visualizar un limite inferior en cuanto al ancho de ventana es más complejo, pues aunque con un tamaño de ventana de 5 meses se ha conseguido suavizar los picos y valles, no se logra un suavizado en la forma de la estructura como para valores de 7 o 9, por lo que habrá que observar los resultado numéricamente para poder decidir. Para analizar si existen diferencias entre las diferentes ventanas y los tamaños de 5, 7 y 9 meses, se va a proceder

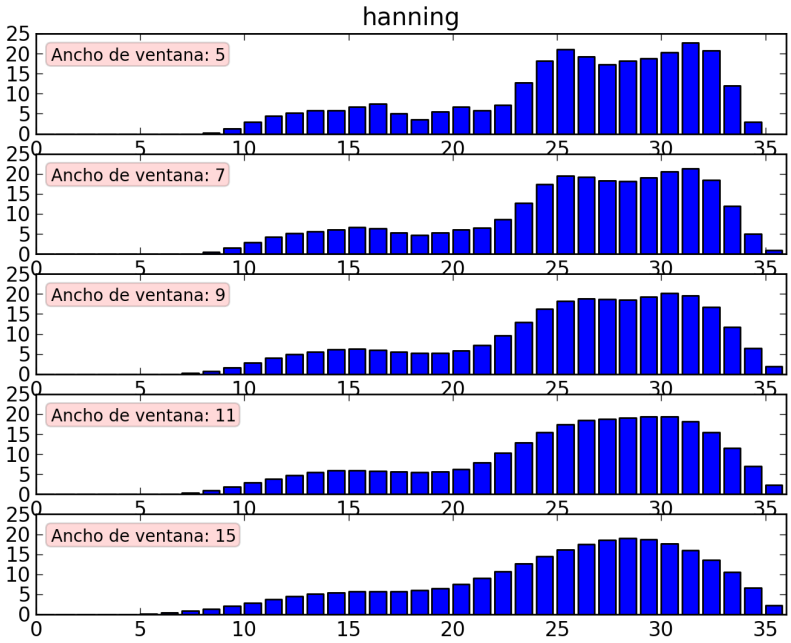


Figura B.5: Estructura del trabajo por meses al aplicar la función hanning

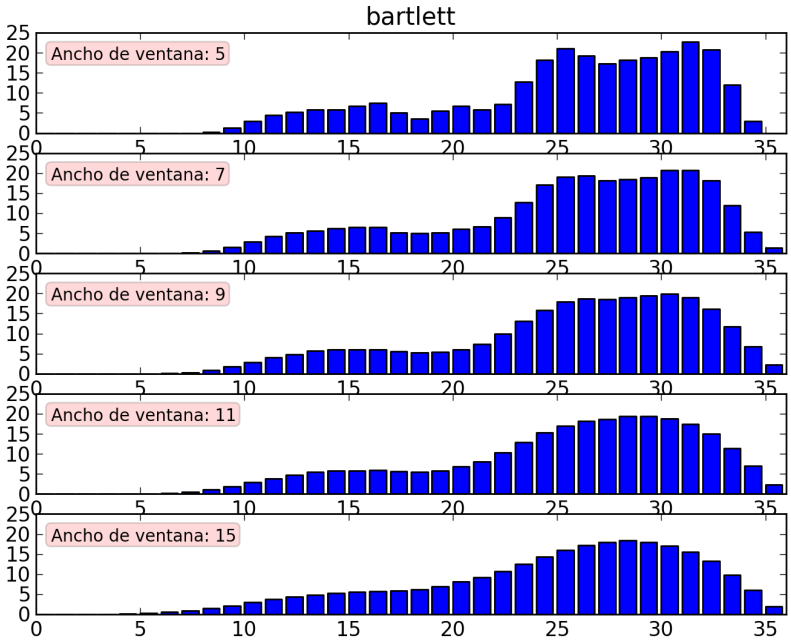


Figura B.6: Estructura del trabajo por meses al aplicar la función bartlett

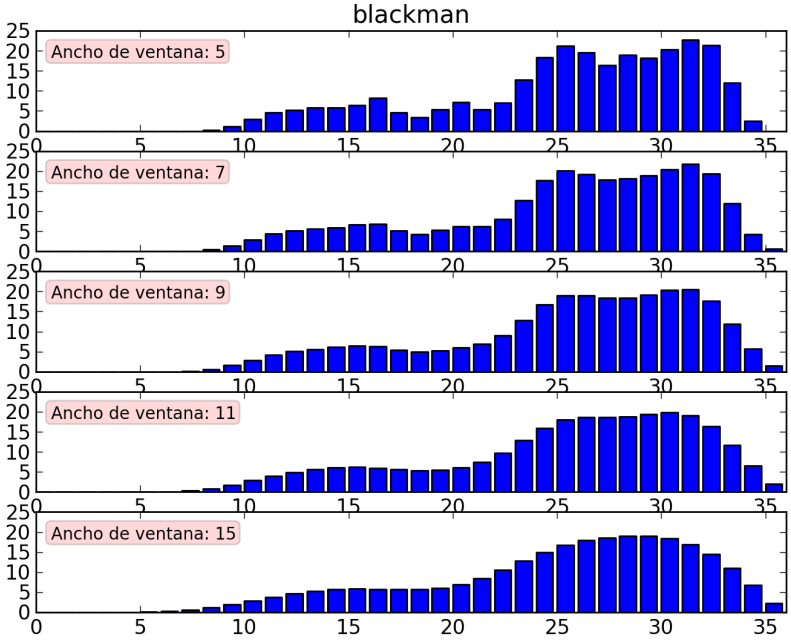


Figura B.7: Estructura del trabajo por meses al aplicar la función blackman

a comparar 2 a 2 la diferencia que hay cada mes en los resultados, para posteriormente obtener la media de dichas variaciones y su desviación típica. Con esto se podrá observar si entre dichas ventanas se produce un cambio significativo en el número de días obtenidos cada mes, o en cambio se pueden tratar como similares para este estudio.

Como se observa, tanto en la Tabla B.1 de las medias como de la Tabla B.2 de las desviaciones típicas las mayores diferencias las encontramos en los casos extremos en los que comparamos 2 ventanas distintas con tamaños de ventana de 5 y 9 meses. Aún así la media de dichas variaciones son entorno al 20 % con una desviación típica cercana a dicho valor, lo que indica que aunque sea más alta que el resto esta diferencia se da sobre todo en

VENTANA		**Hanning**			**Hamming**			**Bartlett**			**Blackman**		
	TAM. VENT	**5**	**7**	**9**	**5**	**7**	**9**	**5**	**7**	**9**	**5**	**7**	**9**
Hanning	**5**	0	5.62	11.01	2.46	8.32	12.33	0	7.64	12.05	2.61	4.05	8.53
	7		0	5.97	5.36	2.68	8.02	8.54	2	7.59	11.29	3.25	2.80
	9			0	12.09	3.83	2.45	15.47	5.49	2.38	17.82	10.01	3.58
Hamming	**5**				0	8.21	13.63	3.52	7.43	13.40	6.38	2.31	8.55
	7					0	6.73	12.37	2.11	6.09	15	6.61	0.43
	9						0	19.89	8.82	1.66	22.13	13.88	6.87
Bartlett	**5**							0	7.64	12.05	2.61	4.05	8.53
	7								0	6.43	12.18	4.78	1.89
	9									0	18.82	11.49	5.47
Blackman	**5**										0	9.95	16.94
	7											0	8.07
	9												0

Tabla B.1: Medias de variación entre ventanas y tamaños (Porcentajes)

VENTANA		Hanning			Hamming			Bartlett			Blackman		
	TAM. VENT	5	7	9	5	7	9	5	7	9	5	7	9
Hanning	5	0	5.62	9.65	2.08	7.34	11.30	0	6.46	10.67	2.10	3.48	7.49
	7		0	5.97	5.56	2.33	6.97	8.93	1.75	7.09	11.37	3.48	2.74
	9			0	12.65	4.07	1.63	15.91	5.66	2.47	18.15	10.65	3.89
Hamming	5				0	10.07	15.93	4.30	9.08	16.04	7.29	2.70	10.51
	7					0	8.25	13.99	1.35	7.76	16.40	7.82	0.43
	9						0	22.82	8.85	1.85	23.93	16.13	7.16
Bartlett	5							0	6.46	10.67	2.10	3.48	7.49
	7								0	5.02	12.20	4.71	1.29
	9									0	18.61	11.78	5.63
Blackman	5										0	14.60	26.19
	7											0	11.85
	9												0

Tabla B.2: Desviaciones típicas de variación entre ventanas y tamaños (Porcentajes)

Media de las medias	Desviación típica de las medias
7.92	5.85

Tabla B.3: Valores en porcentaje de la media de las medias y su desviación típica

las colas de la función, donde aparecen valores en la ventana de tamaño 9 inexistentes en la de tamaño 5, lo que origina que exista una media de variación más elevada pero también con una mayor dispersión. Pero salvo estos casos extremos las diferencias se encuentran normalmente con variaciones menores al 10 %. Por lo que si nos fijamos en la media de la Tabla B.3 y su desviación típica se puede concluir que la diferencia entre los 4 tipos de ventana y los 3 tamaños elegidos es despreciable para el propósito de su uso y por tanto es posible seleccionar cualquiera de ellas con un resultado similar. Eligiendo para el resto de modelos la ventana hanning con un tamaño de 7 meses.

B.3. Resultados obtenidos con desarrolladores de otros proyectos

Llegada a la conclusión anterior en la que se comprueba que es indiferente elegir entre los tipos de ventana hanning, hamming, bartlett y blackman y los tamaños de ventana de 5, 7 y 9 meses. Se va a proceder a comprobar gráficamente el resultado que se obtiene aplicando la ventana hanning de tamaño 7 al trabajo realizado por diferentes desarrolladores elegidos al azar entre los diferentes proyectos que se han ido empleando en el proyecto. Y así ver si el resultado que se ha obtenido en el desarrollador empleado para dicha elección es válido para el resto de casos.

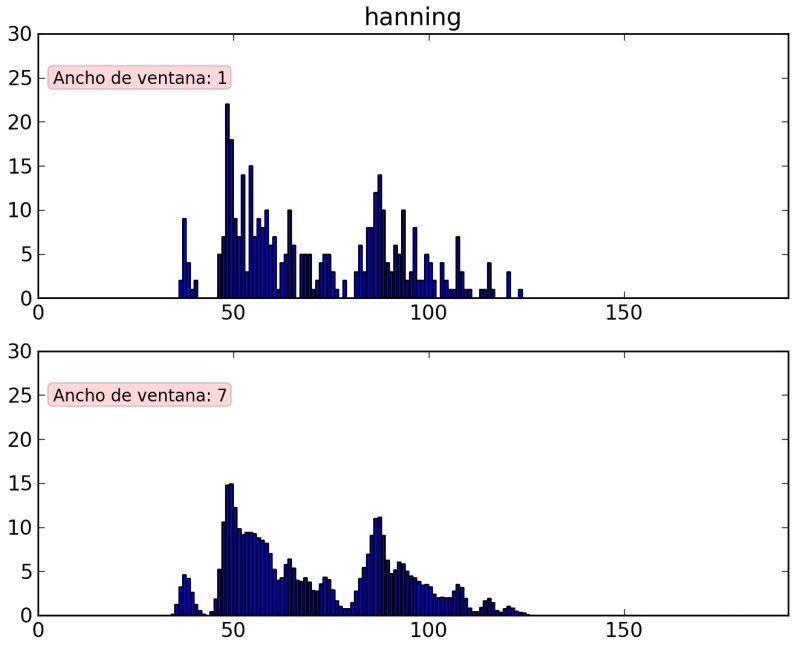


Figura B.8: Resultado del desarrollador con id 126 en el proyecto Gnome Gedit

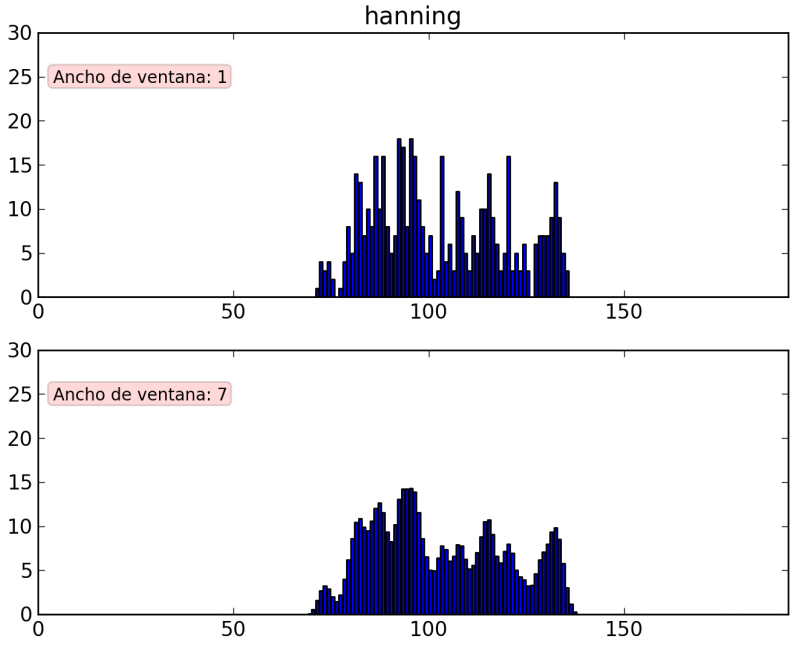


Figura B.9: Resultado del desarrollador con id 243 en el proyecto Gnome Gedit

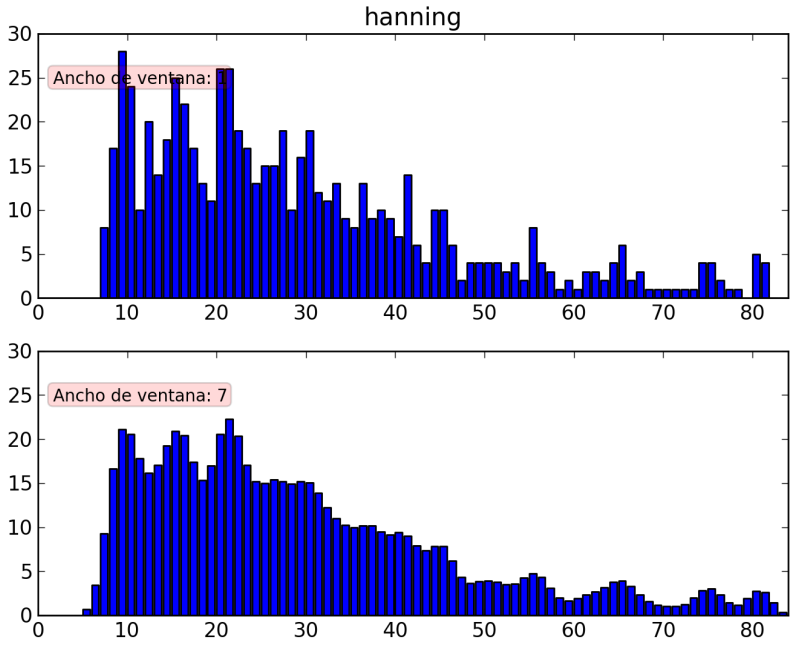


Figura B.10: Resultado del desarrollador con id 1 en el proyecto Gnome Packagekit

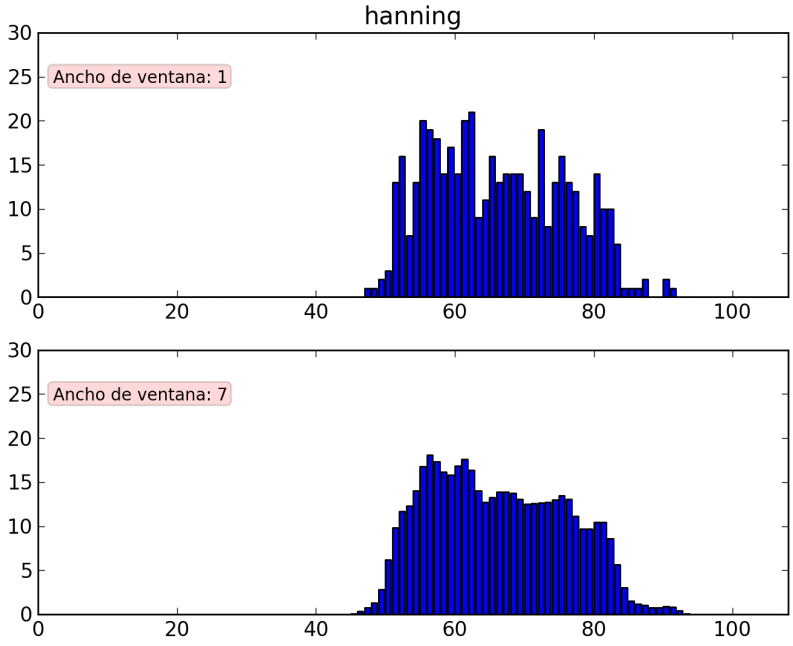


Figura B.11: Resultado del desarrollador con id 83 en el proyecto Gnome Tracker

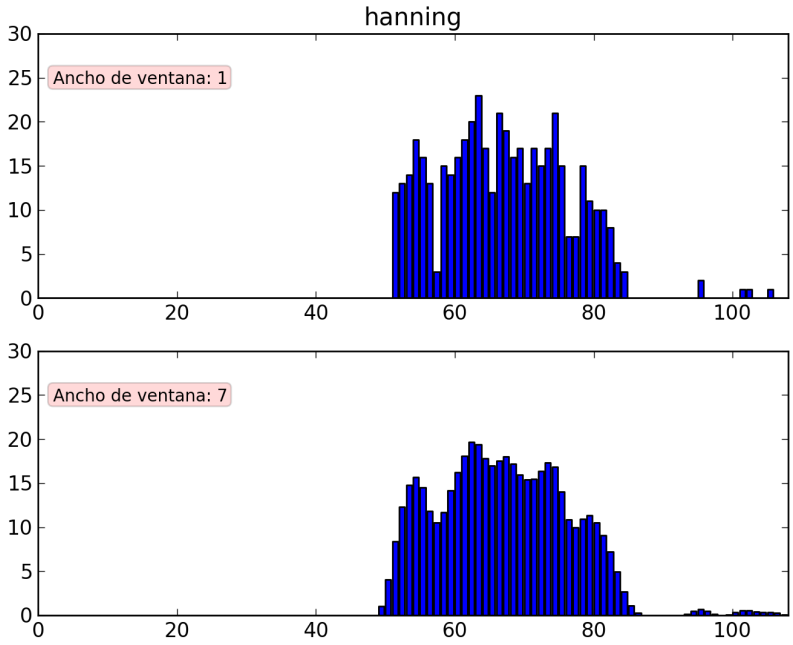


Figura B.12: Resultado del desarrollador con id 84 en el proyecto Gnome Tracker

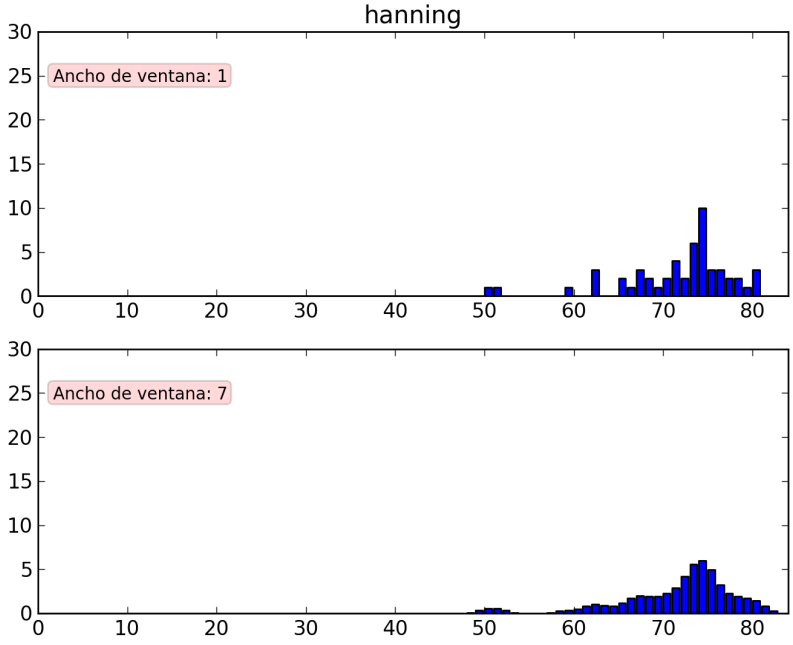


Figura B.13: Resultado del desarrollador con id 274 en el proyecto Gnome Libgweather

Como se puede observar los resultados obtenidos empleando dicha ventana y tamaño son similares a los obtenidos anteriormente, es decir, se logra suavizar la estructura de trabajo de los desarrolladores consiguiendo que los valles y picos esporádicos se ajusten

mejor a la tendencia temporal de trabajo seguida, permitiendo analizar con más precisión las diferentes etapas en el desarrollo del proyecto que ha tenido a lo largo del tiempo.

B.4. Mejora implementada adicionalmente

Aunque la función de *smoothing* realiza su labor como debería, para el objetivo que se busca al emplearla en este proyecto la función sin modificar genera colas en los extremos de los períodos de trabajo. Estas colas son días de trabajo inexistentes por los desarrolladores en el proyecto, y por tanto, errores que se introducen en el modelo de estimación de esfuerzo.

Para evitar estas colar se ha implementado una función adicional al proceso de *smoothing*. Esta función recibe el vector de actividad a suavizar y analiza los períodos de trabajo del desarrollador, llamando a la función de *smoothing* tan sólo cuando ha habido un período de trabajo continuo, con un máximo de 1 mes sin actividad entre meses activos, de como mínimo el ancho de ventana elegido.

El código de dicha función es el siguiente:

```
def fix_smooth(f_original, len_window, window):
    f_smooth = np.array([], dtype=np.int)
    flag = False
    position = 0
    for i in range(len(f_original)):
        if f_original[i] != 0:
            if i+1 == len(f_original):
                if flag:
                    if len(f_original[position:]) == 7:
                        smooth_aux = smooth(np.array(f_original[position
                            -1:]), len_window, window)
                        smooth_aux = smooth_aux[1:]
                    else:
                        smooth_aux = smooth(np.array(f_original[position:])
                            , len_window, window)
                    f_smooth = np.append(f_smooth, smooth_aux)
                else:
                    f_smooth = np.append(f_smooth, f_original[i])
            elif not flag:
                position = i
                flag = True
        elif f_original[i] == 0 and flag:
            if i+1 == len(f_original):
                if len(f_original[position:i]) == 7:
                    smooth_aux = smooth(np.array(f_original[position-1:i]),
                         len_window, window)
                    smooth_aux = smooth_aux[1:]
                else:
                    smooth_aux = smooth(np.array(f_original[position:i]),
                        len_window, window)
                f_smooth = np.append(f_smooth, smooth_aux)
                f_smooth = np.append(f_smooth, 0)
```

```
                flag = False
            elif f_original[i+1] == 0:
                if len(f_original[position:i]) == 7:
                    smooth_aux = smooth(np.array(f_original[position:i+1]),
                        len_window, window)
                    smooth_aux = smooth_aux[:-1]
                else:
                    smooth_aux = smooth(np.array(f_original[position:i]),
                        len_window, window)
                f_smooth = np.append(f_smooth, smooth_aux)
                f_smooth = np.append(f_smooth, 0)
                flag = False
        else:
            f_smooth = np.append(f_smooth, 0)
    return f_smooth
```

Como se puede observar en la Figura B.14, al aplicar la mejora desaparecen las colas de actividad inexistentes que nos producirían un error en la estimación de esfuerzo.

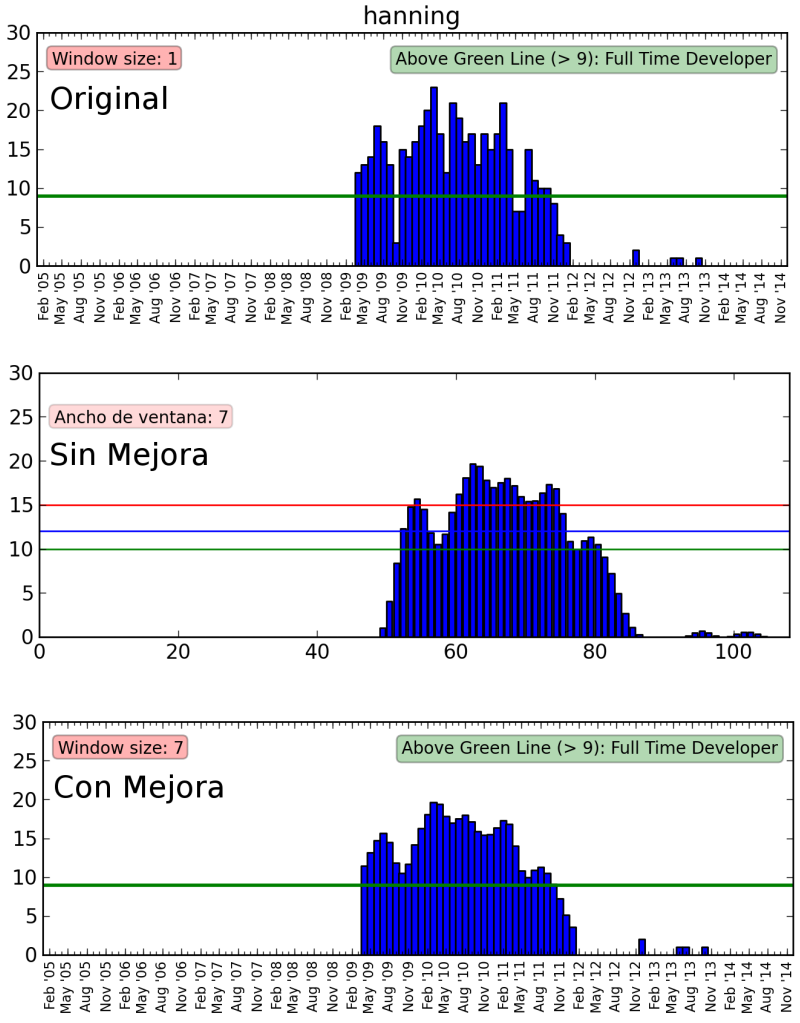


Figura B.14: Comparación de la función de smoothing original con la mejorada

Apéndice C

Abstract BENEVOL 2013

Resumen enviado para participar en el seminario sobre evolución del software BENEVOL 2013 que tuvo lugar los días 16-17 de Diciembre del 2013 en Mons, Bélgica. En éste se explica de manera resumida el objetivo de nuestro estudio, la metodología empleada hasta el momento y los numerables beneficios que supone.

Effort estimation based on Mining Software Repositories - A Preliminary Approach

Carlos Cervigón, Gregorio Robles, Andrea Capiluppi
Universidad Rey Juan Carlos, Brunel University
Madrid (Spain), London (United Kingdom)
c.cervigon@alumnos.urjc.es, grex@gsyc.urjc.es, andrea.capiluppi@brunel.ac.uk

***Abstract*—A commercial company participating in an free/libre/open source project has a clear view of the effort it applies to the projects it chose to contribute to. The effort by other companies contributing to the same project, or by the volunteers participating in it, should be quantified, so that any company could evaluate a return-on-investment on their devoted effort.**

We propose a new way of estimating effort based on the activity of the developers by mining software repositories. This approach is focused on the development of free/open source projects, where there may be companies and volunteers collaborating and information of effort allocated is difficult to obtain by other means.

***Keywords*-effort estimation; mining software repositories; software evolution; free software; open source;**

I. Introduction

In this extended abstract we propose a different approach to effort estimation from the one that has been developed so far in the related literature [1]. Instead of predicting the future effort of a software development, our approach focuses on the estimation of the how much effort has been produced for a given project. If feasible and accurate, this method could be adequate for some scenarios, such as in free/libre/open source software (FLOSS) projects: in particular, it could be used to estimate the return-on-investment of the effort devoted by a company, when contributing to a FLOSS project.

It should be noted that in traditional settings, the effort devoted (to an activity or a project) is known to the company from its records: in these cases, the company has an interest in predicting the future effort based on the historical records. However, and particularly in FLOSS development scenarios, the effort utilised in the activities is not known a-priori, since the stakeholders participating in the project may be volunteer developers or, for sponsored projects, developers affiliated to a company [2]. Companies participating in a FLOSS project are in general aware of the effort they put into the projects in both monetary and human terms. On the other hand, the same companies cannot clearly quantify the effort applied by the rest of the community, including the other companies involved in their same FLOSS project.

Our approach is based on estimating the effort by characterizing the developer activity [3], in particular using data obtained from the *git* repositories. When mining a repository, it is straightforward to obtain "when" developers performed their commits: an open question that the current repositories cannot answer is "how long" the developers devoted to these commits. This missing information could be used to formulate a first draft of the effort devoted by an individual to each commit.

In our approach, we assume that we can identify the developers affiliated to a specific company, and that those developers are full-time committed to the project. These developers are assigned an effort equivalent to 40 hours per week, although we are trying other timespans to include days off, vacations and other circumstances where developers cannot be tracked in the repository, but are still working on the project.

Assuming that we can identify these developers and that we can estimate the effort from the rest of developers from their activity in the versioning system, we expect to have a good estimation of the effort that has been devoted to the project.

II. Benefits

Our approach could benefit companies different scenarios. In the following list we are proposing a few:

- *Return-on-investment*: a company knows well how much effort it has invested in the project, but it is likely that they are not completely aware of the total effort that all participants have devoted in the project (or in the part of the project where the company is involved).
- *Open process visibility*: FLOSS foundations, such as the ones for Mozilla, GNOME or OpenStack, are very interested in these figures and quantitative analyses, since one of their goals is to promote the visibility and openness of their processes while attracting new developers and companies [4].
- *Fairness of contributions*: the involvement of commercial companies in FLOSS projects has been always been a very sensitive issue, as volunteers and other participating companies demand a "fair" collaboration environment [5], [6].
- *Effort-saving synergies*: when a patch is accepted by a FLOSS project, the responsibility of its maintenance and future evolution is shifted from the original authors to the overall project. For a commercial company, getting a contribution accepted in "upstream" may be considered as an added value, that reduces the future

effort in maintaining the same patch. Therefore, a company could benefit from such analysis, by measuring the effort saved when any of its development becomes "upstream".
- It offers valuable strategic information for the project.

III. Methodology

The methodology that we are following is composed of following steps that are briefly described:

1) Extract log information from the versioning system.
2) Identify authors and committers from the activity. We can use in this step, ideas and methods used for the classification of the Linux kernel developers presented by Capiluppi et al. [7].
3) Merge author information [8], as authors may use several identities.
4) Assign authors to companies.
5) Identify professional authors (and assign them 40 hours per week or any reasonable measure for a full-time developer).
6) Obtain an estimation of the effort for the rest of developers. This could be done by clustering the rest of contributors and transforming their activity to effort, but we are currently trying several approaches.
7) Obtain set of metrics: total sum of effort, effort by companies, ratio affiliated versus volunteer developers, RoI, etc.
8) Evaluate the activity to effort transformation by means of a survey of developers.
9) Compare the results obtained with traditional models, such as COCOMO and COCOMO II [11], or previous studies that have researched similar issues, such as "What does it take to develop a million lines of open source code?" [9] or "Indirectly predicting the maintenance effort of open-source software" [10]. .

Acknowledgments

The work of Gregorio Robles has been funded in part by the Spanish Government under project SobreSale (TIN2011-28110).

References

[1] M. Jorgensen and M. Shepperd, "A systematic review of software development cost estimation studies," *Software Engineering, IEEE Transactions on*, vol. 33, no. 1, pp. 33–53, 2007.

[2] B. Fitzgerald, "The transformation of Open Source Software," *Mis Quarterly*, pp. 587–598, 2006.

[3] J. J. Amor, G. Robles, and J. M. Gonzalez-Barahona, "Effort estimation by characterizing developer activity," in *Proceedings of the 2006 international workshop on Economics driven software engineering research*. ACM, 2006, pp. 3–6.

[4] D. Riehle, "The economic case for Open Source foundations," *Computer*, vol. 43, no. 1, pp. 86–90, 2010.

[5] L. Dahlander and M. G. Magnusson, "Relationships between open source software companies and communities: Observations from Nordic firms," *Research Policy*, vol. 34, no. 4, pp. 481–493, 2005.

[6] E. Capra, C. Francalanci, and F. Merlo, "An empirical study on the relationship between software design quality, development effort and governance in Open Source Projects," *Software Engineering, IEEE Transactions on*, vol. 34, no. 6, pp. 765–782, 2008.

[7] A. Capiluppi and D. Izquierdo-Cortázar, "Effort estimation of FLOSS projects: a study of the Linux kernel," *Empirical Software Engineering*, vol. 18, no. 1, pp. 60–88, 2013.

[8] E. Kouters, B. Vasilescu, A. Serebrenik, and M. G. van den Brand, "Who's who in GNOME: Using LSA to merge software repository identities," in *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*. IEEE, 2012, pp. 592–595.

[9] J. Fernandez-Ramil, D. Izquierdo-Cortazar, and T. Mens, "What does it take to develop a million lines of Open Source code?" in *Open Source Ecosystems: Diverse Communities Interacting*. Springer, 2009, pp. 170–184.

[10] L. Yu, "Indirectly predicting the maintenance effort of Open-Source Software," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 18, no. 5, pp. 311–332, 2006.

[11] B. W. Boehm, R. Madachy, B. Steece *et al.*, *Software Cost Estimation with Cocomo II*. Prentice Hall PTR, 2000.

Apéndice D

Presentación BENEVOL 2013

Para realizar la presentación del proyecto en el BENEVOL 2013 se emplearon las siguientes transparencias, en ellas están reflejadas, a modo de resumen, las principales ideas y puntos principales.

Effort estimation based on Mining Software Repositories - A Preliminary Approach

Carlos Cervigón
Gregorio Robles
Andrea Capiluppi

What are we searching with the model?

- A model able to estimate the applied effort in FLOSS projects
- Be able to assess separately the effort applied per developer or company into the project

Benefits

- Return-on-investment
- Open process visibility
- Fairness of contributions
- Effort-saving synergies

Comparison with CoCoMo

- Cocomo only takes into account source lines of code
- To configure it and to obtain reliable data it is necessary to introduce a lot of information
- Only gives an overall estimate of the project, it can't be disaggregated by developers or companies

What do we have for the moment?

- Model 1 – Assessed by number of commits
  - Authors are classified in 3 types (Prof, Semi, Amat)
    - Prof → at least 10% of total commits
    - Semi → at least 5% of total commits
    - Amat → under 5% of total commits
  - Assessment
    - Prof → 1 PM per month worked
    - Semi → 5/22 PM per month worked
    - Amat → 3/30 PM per month worked

What do we have for the moment?

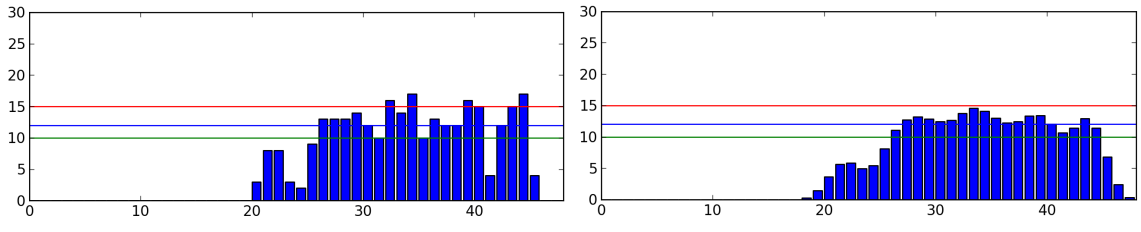
- Model 3 – Assessed by number of workdays
  - Authors are classified in 2 types (Prof, Rest)
    - Prof → at least 15 days per month
    - Rest → under 15 days per month
  - Assessment
    - Prof → 1 PM
    - Rest → Linear function between last professional author (1 PM) and last non-professional author (near zero PM)

What do we have for the moment?

- Model 5 – Assessed similarly to Model 3
  - Authors are classified in 2 types (Prof, Rest)
    - Prof → at least 15 days per month
    - Rest → under 15 days per month
  - Assessment
    - Prof → 1 PM
    - Rest → by percentage of commits (XX)
    - PM_tot = Prof_Authors * 1.XX

What do we have for the moment?

- Model 7 – Smoothing function used
  - Authors are classified in 2 types (Prof, Rest)
    - Prof → at least 10 days per month
    - Rest → under 10 days per month
  - Assessment
    - Prof → 1 PM
    - Rest → Proportional share of number of workdays (days / 10)



Feedback

- Biggest problem → Little info
- What do we need? → Feedback
- Possible solution → Poll

Conclusions

- Several benefits compared with CoCoMo
- Oriented to FLOSS projects
- Its precision can be upgrade with MailLists and Bug trackers

Apéndice E

Paper MSR 2014

Una vez analizados los resultados de la encuesta y validado el modelo con ellos, se ha elaborado un paper que reflejase todo el trabajo realizado hasta ahora, los resultados obtenidos en la encuesta y las conclusiones extraídas. Este paper fue enviado al MSR 2014 para su aceptación y exposición en dicha conferencia los días 31 de Mayo y 1 de Junio del 2014.

Estimating Development Effort in Free/Open Source Software Projects by Mining Software Repositories

Gregorio Robles, Jesus M. González-Barahona, Carlos Cervigón
GSyC/LibreSoft
Universidad Rey Juan Carlos
Madrid, Spain
{grex,jgb,ccervigon}@gsyc.urjc.es

Andrea Capiluppi
Information Systems and Computing
Brunel University
London, UK
andrea.capiluppi@brunel.ac.uk

Daniel Izquierdo-Cortázar
Bitergia S.L.
Madrid, Spain
dizquierdo@bitergia.com

ABSTRACT

Because of the distributed and collaborative nature of free / open source software (FOSS) projects, the development effort invested in a project is usually unknown, even after the software has been released. However, this information is becoming of major interest, especially —but not only— because of the growth in the number of companies for which FOSS has become relevant for their business strategy. In this paper we present a novel approach to estimate effort by considering data from source code management repositories. We apply our model to the OpenStack project, a FOSS project with more than 1,000 authors, in which several tens of companies cooperate. Based on data from its repositories and together with the input from a survey answered by more than 100 developers, we show that the model offers a simple, but sound way of obtaining software development estimations with bounded margins of error.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management—*Cost estimation*

Keywords

Effort estimation, open source, free software, mining software repositories

1. INTRODUCTION

Traditional, effort estimation is generally used by companies in the early stages of a project to estimate the number of developers and the amount of time that it will require to develop a software. Once the project is over, the estimations are no longer needed, because companies usually know the actual effort devoted.

In free / open source software (FOSS) projects, volunteer developers may co-operate with others paid by companies to work on the project [9]. In these cases the total amount of effort invested in the project is usually not known even afterwards.

FOSS projects gained relevance in the late 1990s. Since then, the amount of companies involved in FOSS projects has been growing, and new collaborations have emerged. While during its beginnings FOSS was mostly developed by volunteers, nowadays the collaboration between developers is more varied, and projects range from those still developed only by volunteers, to those which are based on the collaboration of companies with volunteers (e.g., GNOME, Linux), to the clear industrial ones, in which the main driving force are companies (e.g., WebKit or OpenStack) [11, 7].

In the case of a company participation, obtaining overall the development effort for a certain period of time, which is usually an important measure for a company, is not straightforward. Individual companies are usually aware of the effort they put into projects, but they do not have a clear picture of the effort by the rest of the community, including other companies collaborating in the same project.

Not only the companies involved in FOSS development are interested in estimating effort in FOSS projects: foundations that act as umbrellas for many FOSS projects have also interest in this data, as it may offer insights into the project and attract industrial interest [22].

In this paper we present a quantitative approach to effort estimation, based on the observation of real development work. For estimating effort in a project, we calibrate a general model, based on measurements, with the aim of limiting the margin of error and obtaining a good estimation. The estimated effort takes into consideration (i) the individuals who do not dedicate their whole work-day to FLOSS development, and (ii) developers -hired by companies or not- that work full-time in the project. The challenge consists in selecting the appropriate measurements for estimating the work performed, and in mapping such work to developers in a unique way, considering the extremely different working patterns observable in FOSS projects [24, 23, 15, 5].

Our model examines and estimates the effort from the activity tracked in a project's source code repository. In particular, a simple method is sought, so that a reasonable estimation should be obtained from mining of the source code repository. Apart from specific tasks that require human interaction, the proposed method is completely automatable. Two main measurements of activity will be used: (1) the number of changes to source code (commits) per developer

MSR '14

(as a proxy of the amount of activity per person), and (2) the number of days with commits per person (as a proxy of the time periods when a person is actively developing software).

A part of the study was a survey sent to developers of OpenStack, asking for the effort they devoted to the project. These results were used both to calibrate our estimation model, and to validate the results of its application.

The rest of this paper comprises the following: in the next section, the model for effort estimation is presented. Then, we introduce OpenStack, the software project used as a case study. Being our approach of empirical nature, we offer in section 4 some details about the experimental design, which includes details on the survey that was sent to OpenStack developers. The next section is devoted to the analysis of the parameters of the model that provide estimations with minor margin of error. The parameters obtained will be used to obtain some estimations (section 6). Our approach is then validated. Section 8 presents the related research; threats to validity follow. Conclusions are finally given and avenues for future work discussed.

2. MODEL

In this section, a model for estimating the effort based on tracing the developers' activity is presented. This model translates the activity of a developer, as recorded in the source code management system, into effort measured in person-months. Although there are other development repositories where developer activities can be tracked, we have found that the results obtained from SCM systems are good enough to produce a reasonably accurate estimation of effort, as will be shown below.

When mapping the activity traces to effort, the main issue to be solved is that the activity can be traced only in certain points in time (e.g., when a commit is introduced into the code base). The information about how long a developer has devoted to produce the contribution is completely missing.

It is well known that contributions to FOSS projects are not uniform: a few developers are responsible for a major amount of the work, while a large number of developers contribute with a comparatively small one [18, 20]. So, one of the problems when measuring effort of the overall FLOSS development process consists in translating the uneven nature of contributions [12] into a consistent estimation model.

Inferring the effort devoted by regular and occasional contributors is not easy, and in addition is a source of high inaccuracies. However, it is known that there are developers working full-time on the project, usually but not necessarily hired by companies. Based on the normal work hours in a paid job, we can assume that these developers devote to the project around 40h/week. We could also estimate for them one person-month of effort with a low margin of error. Thus, the model we propose is based at first on identifying those developers that collaborate full-time in the project.

We base our approach on two assumptions:

- With good precision, we can identify those developers working full-time on the project. If that is the case, those developers can be considered to devote one person-month (i.e., 40h/week) of effort during the months they work full-time in the project.

- The number of contributions by full-time developers is sufficiently large: conversely, the other developers contribute a relatively small amount of effort to the project. Since the contributions by the rest of developers are small, even gross estimations of it, which could have a relatively large error, will have a little influence in the total estimation error.

As an example, let's imagine a project P with 1,000 developers of which 100 can be identified as full-time. As each month we assign one person-month to each full-time developer, we would have a total effort of 100 person-months from these developers every month. If the 100 full-time developers author 90% of the contributions to the project in that month, we could estimate the contribution of the rest of the developers in the range of 10 person-months.

The total estimation for the project in a month would then be 110 person-months, which is composed of a larger part (around 100 person-months) in which the margin of error is small; and a smaller part (around 10 person-months) where the margin of error can be large.

At the current state of our research we are not able to provide a value to the margin of error of our estimation. Given the proportion between the efforts involved, we can state that the large error introduced by occasional contributors is mitigated by the fact that its weight on the total effort is low. Hence, in our previous example, the effect of a 20% margin of error for the effort estimated by full-time contributors is equivalent to a 100% margin of error by the rest.

In order to identify the full-time developers, we have used two possible measures for each developer:

- Number of commits merged in the code base during a given period.

- Number of active days during a given period, considered as days in which the developer performs at least one commit to the code base.

In both cases, the model is based on finding a threshold value **t** for the number of commits (or active days) for which we identify full-time developers with a minimum error, while the number of non-full-time developers identified by mistaken is very low. Ideally, activity of all full-time developers would be above **t** while non full-time developers would stay below it. In the real cases, the error for mistakenly identified full-time and non full-time developers should be estimated, with the aim of keeping it as low as possible.

In order to calculate values for the threshold **t**, we sent a survey to OpenStack developers. In this survey, we asked them how many hours per week in the mean they spent in the project in the last six months, as well as how they see their commitment (full-time, part-time, occasional). This information is contrasted with the activity of the developer in the versioning system during the last six months. We calculate the best fits for several values of **t** and obtain effort estimations in person-months.

We would like to point out that we have anecdotal evidence from some FOSS managers at companies that use what we could call a naive version of the model for their estimations. What is done is to assign 1 person-month of effort to any developer with at least one commit in a month. In our model, this would be equivalent to have a threshold value of **t**=1.

3. CASE STUDY

OpenStack is a software project to build a SaaS (software as a service) platform. Even being a *young* project, with its first official release in 2010, over 200 companies have become involved in the project in such a short period of time. AMD, Brocade Communications Systems, Canonical, Cisco, Dell, Ericsson, Groupe Bull, HP, IBM, InkTank, Intel, NEC, Rackspace Hosting, Red Hat, SUSE Linux, VMware Yahoo! and many others have been participating in OpenStack since.

First release	October 2010
# Authors (distinct)	1410
# Commits	89,871
# Commits (without bots)	68,587
SLoC (approx.)	1,650,000

Table 1: Main parameters of the OpenStack project, January 2014.

Table 1 provides a summary of the project. There are 1,840 distinct author identifiers in the source code management system. After applying a merging algorithm [19], manual inspection and feedback from the community, 1,410 distinct real authors have been identified, once duplicate ids were merged. The total number of commits is almost 90,000, although many of these have been done automatically by bots.

Bot name	Bot mail address
Jenkins	jenkins@review.openstack.org
OpenStack Jenkins	jenkins@openstack.org
Jenkins	jenkins@review.stackforge.org

Table 2: Main bots found as authors in the OpenStack source code management.

Table 2 contains the bots that merge commits in the versioning system. They are responsible for 21,284 commits, almost 25% of all commits in the project.

OpenStack, as many other FOSS projects, has an uneven contribution pattern where a small group of developers have authored a major part of the project. In summary, 80% of the commits have been authored by slightly less than 8% of the authors, while 90% of the commits correspond to about 17% of all the authors. We have thus a situation where the assumptions of our model above are true. In addition, as the corporate involvement in OpenStack is significant, this should allow us to identify full-time developers from companies. The OpenStack Foundation estimates that around 250 developers work professionally in the project.

4. EXPERIMENTAL DESIGN

4.1 Activity data

This paper presents an empirical approach to measure the effort within FLOSS projects, based on the number of commits logged onto the versioning system by the developers of that project. We base our research on following concepts:

- **Commit:** action by which a developer synchronizes a set of changes to the versioning system repository. A commit is given by a point in time (the timestamp can be obtained from its metadata), and does not include information on how much time or effort the developer has spent on it [1]. In OpenStack commits require usually to be approved, as a review process is in practice. This makes contributing a commit a very elaborated process, slow and with a big granularity (i.e., with many files), closer to the traditional Modification Requests in industry [21] than to commit patterns found in other FOSS projects.

- **Active day:** day in which the developer performs at least one commit to the code base. We have used it as an alternative measure of activity to commits.

- **Committer:** developer who performs the commit to the repository. May not have to be the author of the changes. In our model, we do not consider committers.

- **Author (or developer, or contributor):** individual who contributes the changes, but may not be the one who performs the commit (which is done by the committer). Depending on the amount of contributions (in number of commits or number of active days), we identified their status as a "full-time" or "non-full-time" contributor. Depending on their status, the empirical approach assigns a person-month (PM) for full-time developers or a fraction of a person-month for the rest.

- **Person-Month (PM):** measure of effort. We suppose that a person-month is observed when a developer devotes the relative amount of time of a 40 hour-week to the project. Full-time developers hence account for one PM for each month worked on the project.

- **Period of study**: timespan during which developers with more activity than a threshold t will be considered as full-time developers. Although the *natural* measure of the period could be thought to be one month, our experience has shown that using larger timespans has many benefits as it smooths the effect of vacations, travels, and non-constant work, among others. Being the release cycle of OpenStack of six months, we have selected periods of study of that length in order to avoid influences of the process in our measures. A developer who surpasses the threshold in the six month period will be thus considered to have devoted 6 PMs. A developer with activity a, less than the requited threshold t, will be assigned an effort of $6 * a / t$ PMs.

For this study, we have used the data gathered by Bitergia for the OpenStack Foundation, which has been carefully curated and validated [13]. The original data is available, as MySQL database dumps, from the OpenStack Development Dashboard[1]. The data has been discussed and commented with several experts from the OpenStack community to find issues in it, which have been addressed to improve its quality.

The information from the source code management repository is obtained by means of the CVSAnalY tool, which is part of the MetricsGrimoire toolset[2]. CVSAnalY retrieved

[1] `http://activity.openstack.org`
[2] `http://metricsgrimoire.github.io`

information from repositories (in the case of OpenStack, git repositories) of the project, and stores it in a MySQL database, ready for analysis.

Real authors of commits, corresponding to persons, are identified by using several algorithms to merge the different author identities they may use (usually, different email address used as git author identifiers) [19]. This process has been manually checked and completed. All this work has been done on the dataset by Bitergia, the company maintaining the OpenStack Foundation Development Dashboard. Methods and tools used in the production of the curated dataset have been discussed and checked by the authors.

4.2 Survey data

To obtain data of the time commitment of the OpenStack developers, we designed an on-line survey. We obtained the e-mail addresses of the developers from their authorship information in the versioning system, and sent them an e-mail with an invitation to participate.

The questionnaire was composed of two web pages and contained following questions (information of the page in which the question appeared and the type of question is offered in brackets before the question; possible answers are given in brackets after the question):

- (1, Selection): On average, how many hours in a week have you spent in the project in the last six months? (>40h, 40h, 30h, 20h, 10h, <5h)
- (1, Selection): How much of the time you spent in the project is devoted to coding? (>95%, approx. 75%, approx. 50%, approx. 25%, <10%)
- (1, Selection): Do you make at least one commit to the repository the days you code? (yes, no)
- (2, Selection): What do you consider yourself in the project? (full-time, part-time, occasional contributor)
- (2, Free-text box): Did you always work on the project the same amount of hours, or did you have different phases of commitment? If you had different phases, could you tell us about the various phases? (the graph below may help you, as it is based in your recorded activity in the repository)

In the displayed graph, the developer could see a bar-graph with the amount of commits per month in the versioning system. In order to obtain clearer phases, we used a Hanning window function to smooth the irregularities. In order to obtain feedback on possible threshold values, we also included an horizontal threshold (see green line) to mark when we considered the developer full-time. Figure 1 shows an example of such a graph for a developer.

The survey web site[3] included information about our research goals, a privacy statement (informing that the survey data will not be released as it contains personal information according to the Spanish data protection law) and contact information.

1407 personalized mails were sent, and the survey was answered by a total of 131 respondents. From these, we removed five survey responses because they were empty or we could see from the answer in the free text box that the respondent had misunderstood/misinterpreted the questions

[3]http://ccervigon.libresoft.es

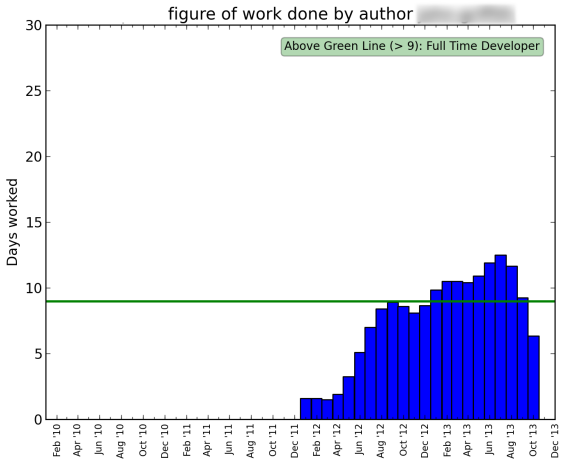


Figure 1: Personalized activity chart for a developer.

(for instance, there was a developer referring to his contributions on the LibreOffice project, not on OpenStack). We also changed 7 surveys to normalize the set, mainly because some respondents devoted 40 or more hours a week to the project mostly in programming, but stated they were occasional/part-time developers.

5. ANALYSIS

Our model is based on identifying developers as full-time (or not full-time), so a classification can be performed. We will have a true positive when we identify a full-time developer as such; not identifying him/her as such will cause a false negative. In the case of a non-full-time developer, if we guess correctly we will have a true negative; otherwise, we will consider him/her as full-time, and have a false negative.

From the relationship of these values we can infer how good our classification is using several standard information retrieval and pattern recognition measures. Figure 2 offers the most relevant ones: precision, recall, F-measure and accuracy. These measures depend on the value **t** chosen as the threshold. As it can be seen, the recall starts at 1.0 for t=1, and then decreases steadily to values approximately 0.3 for t=50. Precision, on the other hand, has a value 0.4 for t=1, and increases to values of 0.8 for t>20.

Figure 3 offers a detailed view of the values of t where the F-measure and accuracy obtain their maximum values. The F-measure, which is the harmonic mean of recall and precision, peaks at t=9 with a value of 0.75. Accuracy is calculated as:

$$Accuracy = \frac{t_p+t_n}{t_p+t_n+f_p+f_n}$$

and provides another measure used in classification to determine the quality of the classification criteria. The maximum value of accuracy (0.78) is obtained for t=9 and t=10.

Figure 4 provides visual detail on how the classification is performed and how this affects these measures. On the left, we can see how the identification of full-time developers becomes less accurate while the threshold t grows. However, as can be seen from the figure on the right, for non-full-time developers the effect is the contrary: the classifier gets better

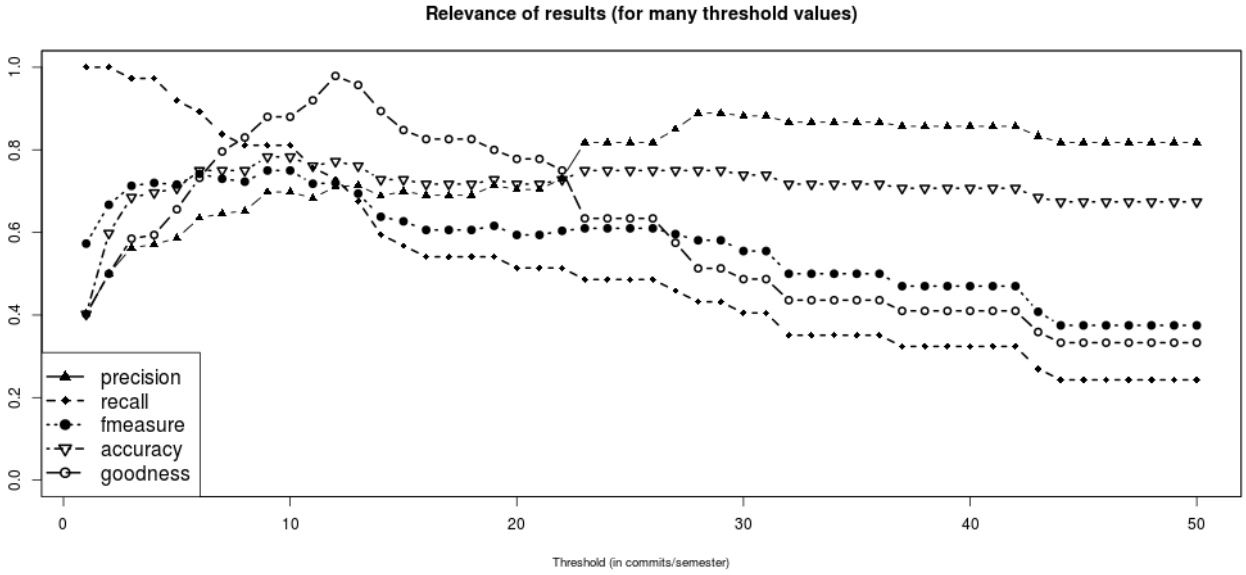


Figure 2: Relevant measures (precision, recall, F-measure, accuracy and goodness) for several values of threshold t. Figure 3 provides a detail of this graph for values of t between 7 and 15.

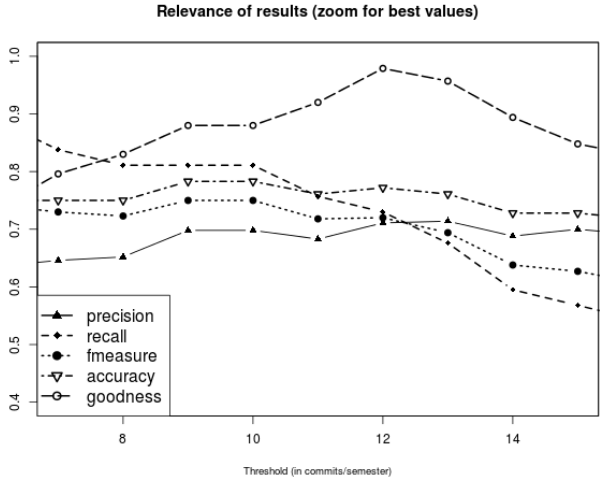


Figure 3: Zoom of relevant measures (precision, recall, F-measure, accuracy and goodness) for values of t between 7 and 15.

with higher values of t. It can be observed that the curve for non-full-time developers saturates quickly, while the one for full-time developers decreases more slowly. This means that there are many non-occasional contributors with a low number of contributions, while contributions by full-time developers are almost uniformly spread among all amounts of contributions.

There is an effect in our effort estimation model that should be taken into consideration as it is usually not to be found in information retrieval and pattern recognition: false positives and false negatives may compensate each other. This happens because a wrongly classified (false positive) non-full-time developer is assigned the effort of a full-time developer; if a full-time developer exists that has been classified as non-full-time developer (false negative), then this error is compensated by the former.

Figure 5 presents this situation graphically by subtracting the number of wrong classified non-full-time developers from the number of incorrectly classified full-time developers. As it can be seen, for t=1 no full-time developer has been incorrectly classified, but all non-full-time developers have. No compensation occurs then. However, as t increases the error of classifying full-time developers incorrectly is compensated by the error of classifying non-full-time developers. The value for which the number of non-compensated errors is minimum corresponds to t=12, where the difference is 1 (i.e., there is one false negative more than false positives).

Because of this compensation phenomenon, we have conceived a new measure that takes this effect into consideration. We have called it *goodness*; it depends on the number of non compensated classifications (the numerator, an absolute value, since compensation works both ways) and is normalized by dividing it with the total number of positives and false negatives. To obtain a value of *goodness* which becomes better as it gets closer to 1, we subtract the result of the fraction from 1. Thus, *goodness* can be calculated as follows:

$$Goodness = 1 - \frac{|(t_p+f_p)-(tp+fn)|}{tp+fn+fp}$$

where t_p stands for *true positive*, f_p for *false positive* and f_n for *false negative*. The reader should note that we do not need to consider *true negatives* in this measure; *true negatives* are given by those non-full-time contributors that have been correctly categorized. Given that their effect in our model is assumed to be small, this omission is in line

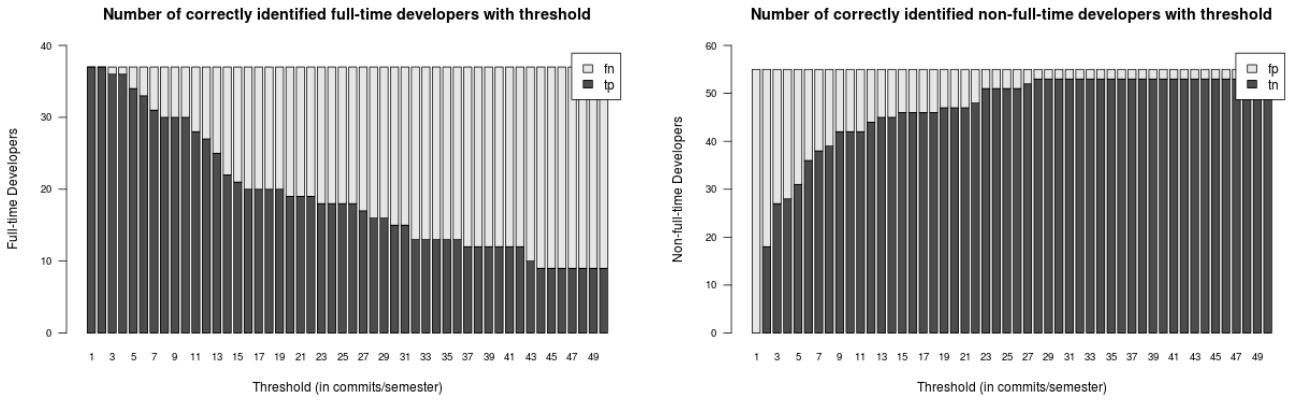


Figure 4: (l) True positives (tp) and false negatives (fn) for full-time developer identification for several values of threshold t. (r) False positives (fp) and true negatives (tn) for non-full-time developer identification for several values of threshold t.

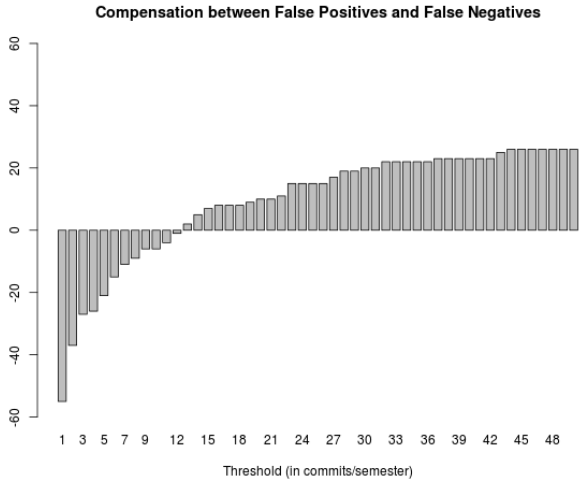


Figure 5: Compensation between false positives and false negatives for values of t. Negatives values of the y axis indicate false positives that do not get compensated by false negatives, while positive values of the y axis indicate false negatives not compensated by false positives.

with the model.

If all classification errors get compensated, the numerator of the *goodness* formula will be 0 and the value of *goodness* will be 1. If there are many errors which are not compensated, the numerator will grow and the resulting *goodness* would be low.

Figure 3 shows that the value of *goodness* peaks at t=12 (0.979). So, even if the F-measure and accuracy have slightly lower values than their maximum for that threshold, the error for our effort estimation should be smaller for t=12 due to the effect of compensation that the *goodness* factor recognizes.

All in all, our analysis shows that the best effort estimation can be obtained with threshold values in the range from t=9 to t=12.

A side result of our analysis is that in OpenStack the difference between using commits or active days as an activity measure is not significant. Talking with OpenStack developers about this fact, they recognized that the development process in OpenStack puts a very high priority on stability. Thus, code is not formally committed to the repository until it has been through extensive review, usually including several revisions. This has as a consequence that commits are *larger* and more *costly* (in time and effort) than in other FOSS projects. This is also the reason why we have obtained in our analysis threshold values of 9 to 12 commits every semester to be considered a full-time OpenStack developer, a number of commits that in other projects is easier to achieve. Although this does not affect the validity of our case study, further research should address this issue to offer a general model.

6. RESULTS

Table 3 shows the results obtained with our approach using several values of the threshold **t**. Considering the global effort through the whole lifespan of the project, we can see that -as expected- the number of person-months decreases while the threshold increases. The naive approach, where a single commit in a semester would have been coded as 6 person-months, results in 17,400 person-months, which as already discussed could be considered as an upper bound of the effort.

According to the analysis, the best values for the **t** threshold are between 9 and 12. For such scenarios, the estimation of the amount of effort that has invested in the project lies around 9,000 person-months (750 person-years), which is half of the estimation value produced by the naive approach.

The table also provides information for every semester. As it can be observed, the effort devoted to the project is increasing with time, with a maximum for all the thresholds in the second semester of 2013. If we take the value of 12 as the threshold, 2,634 person-months have been working during the last six months. This implies that we estimate that the actual effort on OpenStack has been around

440 person-months during the last six months. When asking OpenStack members about these figures, they confirmed that they sound reasonable as the estimation of the OpenStack foundation is that currently over 250 professional developers work on the project hired by companies.

t	Effort	10s2	11s1	11s2	12s1	12s2	13s1	13s2
1	17400	426	816	1242	1890	2742	4368	5916
2	15255	420	813	1170	1647	2385	3783	5037
3	13804	402	798	1110	1486	2140	3384	4484
4	12747	389	785	1056	1383	1964	3074	4098
5	11891	378	768	1008	1295	1830	2821	3791
6	11213	370	757	964	1224	1726	2618	3554
7	10629	362	743	927	1164	1635	2451	3346
8	10128	355	732	896	1112	1560	2310	3164
9	9683	345	721	866	1065	1493	2189	3003
10	9298	337	711	841	1025	1434	2086	2864
11	8957	330	700	819	987	1381	1997	2743
12	8655	324	690	800	955	1334	1919	2634
13	8370	318	680	782	924	1289	1847	2531
14	8112	313	672	767	896	1247	1781	2437
15	7876	308	663	753	872	1208	1721	2351
16	7662	303	656	740	850	1173	1666	2275
17	7466	298	648	729	830	1140	1615	2206
18	7284	294	641	719	812	1107	1568	2142
19	7109	291	634	708	794	1077	1522	2083
20	6945	288	628	698	777	1048	1481	2025

Table 3: Estimated total effort (in PM), given per threshold value (column t). Estimated effort values for all semesters.

On a side note, it should be noted that the ratio between the effort estimated with the naive approach and the one with a threshold of 12 is steadily increasing. In the second semester of 2010 the ratio was 1.31, while for the second semester of 2013 it has grown to 2.25. This is because the number of non-full-time contributors has grown as well. Thus we can see how much the "minor contributors" in the project affect the estimations given by our model. While the error (noise) introduced by these small contributions is included using the naive approach, higher values of **t** filtered it out. The result is not only a more realistic estimation, but as well an estimation where the more error-prone estimations have a smaller weight in the total effort.

7. VALIDATION

7.1 Share of commits

The approach is valid if the number of commits performed by those developers which we have identified as full-time developers consists of a large fraction of the total number of commits. Figure 6 provides information of this value for the first 50 values of t. Logically, for t=1 the total number of the 68,587 commits corresponds to developers identified as full-time (and thus the share is 1.0). This value decreases as t grows, but very slowly; for t=12, the share is still over 0.9, and it stays over 0.8 up to values of t close to 30.

In summary, we see how the amount of work performed by those who we classify as full-time developers is over 10 times larger for values of t that provide the most accurate effort estimations (t=9 to t=12). This means that being our classification accurate, the margin of error that is introduced by the rest of the effort is bounded.

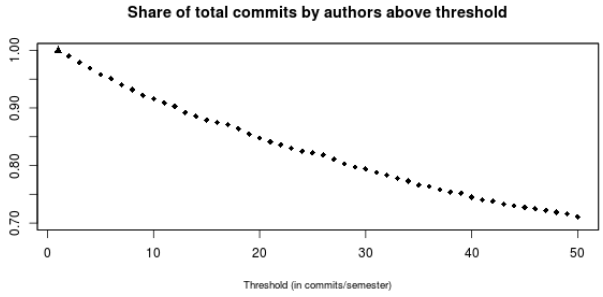


Figure 6: Share of total number of commits by developers identified as full-time for several values of the threshold t.

At this moment, we are unable to calculate with precision values for margins of error. However, with the current proportion of 10:1, margins of error of the margin of error of the rest can be up to 5 times the one of full-time developers to have the same effect on our estimation. So, for a margin of error of 20% in estimation for full-time developers, we still could afford a 100% error margin for the rest of developers so that the global, aggregated estimation of the project error will be around 25%. This is an important outcome of our model, and does not depend on the calculation of the margin of error, which should be addressed as future research.

7.2 Representativity of the survey sample

The survey provides data of a sampled number of developers. As we have invited all OpenStack authors to fill out the survey, the method used has not been random sampling: survey respondents are self-selected, and thus may not be representative for the whole population under study.

Figure 7 shows two box-plots with the analysis of the active surveyed population and all active developers of the project in the last six months, which is the timespan we asked for in the survey.

As the normality assumption of the t-test is not given in our samples, the non-parametric alternative Wilcoxon signed-rank test (also known as Wilcoxon-Mann-Whitney test) has been used. The null hypothesis for this test is that no difference between the two populations exist. When the p-value for the two-tail hypothesis is larger than the significance, we conclude that the the null hypothesis cannot be rejected. This happens for all populations considered with at least one commit in the time period.

However, we also tested the hypothesis including inactive developers in the population. In that case, the survey is not representative and the null hypothesis can be rejected. The reason for this is that the inclusion of all inactive developers from all the project history introduces much interference. This is because the population of former developers is very different from the current one, so including it in the global population produces a severe transformation. As our model is based on activity, this fact is not a threat to its validity, but a supportive fact, as we perform the classification only with active developers. Nonetheless, as it can be seen from Table 4, as the values of activity increase, the survey

Commits	Population	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	p-value
0 or more	all	0.00	0.00	1.00	9.62	6.00	491.00	0.005063
	survey	0.00	0.00	4.00	14.12	14.00	201.00	
1 or more	all	1.00	1.00	3.00	13.76	11.00	491.00	0.2008
	survey	1.00	1.00	5.00	16.35	19.00	201.00	
5 or more	all	5.00	7.00	13.00	29.37	30.00	491.00	0.2822
	survey	5.00	8.50	16.50	29.38	34.75	201.00	
8 or more	all	8.00	12.00	19.00	37.77	41.00	491.00	0.4171
	survey	8.00	13.00	22.00	35.57	42.75	201.00	
11 or more	all	11.00	14.75	24.00	44.34	52.00	491.00	0.9797
	survey	11.00	14.00	26.00	38.83	50.00	201.00	

Table 4: Summary of population measures. Several populations have been selected, depending on a minimum number of commits. p-value as given by the Wilcoxon signed-rank test.

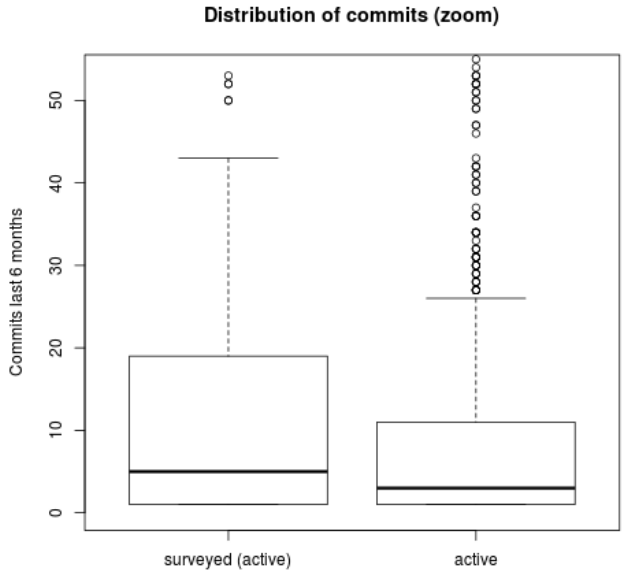


Figure 7: Boxplot with the activity (in number of commits during the last 6 months of 2013) for active surveyed developers and for all the active OpenStack developers in the last 6 months. Outliers with more than 50 commits are not shown.

population becomes more representative of the project.

8. RELATED RESEARCH

Effort estimation in FOSS projects has been the matter of research in several studies. The closest ones to the approach present in this paper are those that try to characterize developers in order to infer the effort devoted, as done in [1, 17, 5].

Amor et al. [1] propose to characterize the complete developer activity (versioning system, mails, bug-tracking system, etc.) in order to obtain a measure of the total effort invested in the project. This idea is extended by Kalliamvakou et al. in [17]. In comparison to our approach, this is a comprehensive focus that needs to gather data from many sources, and with the problem of estimating micro-contributions.

Capiluppi et al. [5] characterize the Linux kernel by the time in a day when the commits are observed in the repository, and when the author works most frequently. They divide a working day in traditional Office Hours (OH, from 9am to 5pm), After Office hours (AO, 5pm to 1am), and Late Hours (LH, from 1am to 8am). The authors present evidence that throughout the week the effort within the Linux kernel community is constant, different from the ones traditionally used in industrial settings where work is assumed 9am-5pm, Monday to Friday. The approach presented in this paper is based on the idea of characterizing developers, although we do this in a different way.

Most of the effort estimation techniques are based on statistical methods. So, Koch et al. tried to obtain effort estimation based on the Pareto distribution found in FOSS projects [18]. Fernandez-Ramil et al. note that traditional software effort estimation methods such as COCOMO do not fit well FOSS projects and study effort using linear regression models [10]. Linear regression is as well the method used by Yu to predict indirectly maintenance effort in FOSS projects [25]. Anbalagan et al. have worked on predicting how much time developers spend in corrective maintenance of FOSS projects [2].

There are several studies that analyze how effort in FOSS is related to other aspects of the development process. In this sense, Capra et al. discuss and investigate empirically how governance, process and design quality affect development effort in FOSS projects [7]. In [8] evidence is shown that in FOSS projects the maintenance effort is lower than in industrial projects due to the fact that they are less prone to software decay.

For a detailed, systematic review of software development effort estimation studies we refer to [16].

9. THREATS TO VALIDITY

All empirical studies, such as this one, are subject to threats to validity. In this section, we discuss how to mitigate or control these threats if possible.

Conceptual Assumptions and assertions have been done. We have tried to make them explicit, as for instance the assumptions that a person-month equates to 40h/week. Some of them may require further support, but this does not invalidate the plausibility of the argument.

Construct A replication package[4], following the guidelines in [14], is offered to others to reproduce our findings.

[4] `http://gsyc.urjc.es/~grex/repro/2014-msr-effort`

The replication package contains all scripts and public information, but not all the results of the survey, as personal information was collected. Aggregated and anonymized information from the survey is available.

Internal Having only considered one of the many sources of activity (and data) may result in our estimation model being not accurate. Further work is expected to be of broad interest and supportive of the claims presented.

External OpenStack was chosen as the illustrative case study. We consider that this project is representative of many FOSS projects, especially those where many industry players cooperate. There may be certain practices that could only be found for OpenStack: for instance, their review process produces large (and few) commits, so values for threshold should be selected differently for other FLOSS projects. This remains to be investigated. However, as Bird and Zimmermann [4] do, we refer to the importance of the detailed study of a single project and that it is a misconception that this type of studies do not provide to the academic community any value, nor contribute to scientific development.

10. CONCLUSION, DISCUSSION AND FUTURE WORK

Differently from other effort estimation research attempts, this paper tackles two challenges. The first is how to design a simple, but sound approach to measure the effort provided in a sparse, distributed and uneven development scenario, like the FOSS one. This is achieved by evaluating the activity of developers and identifying what authors could be considered as "full-time" developers, and separating them from the other contributors. The activity of full-time developers is assigned one person-month per month in the period of study; other developers are assumed to be working less than a full month on a project, hence a fraction is assigned to them.

The second challenge is how to design the model so that it offers a reasonable prediction. This affects primarily two aspects of the model: (1) the activity threshold to consider a developer as full-time or not, and (2) the credibility of the model. For the calculation of a threshold, we have obtained data from over a hundred developers, and we used traditional classification techniques to select the optimal threshold value. In addition, we have seen that the design of the model allows to compensate erroneous classifications of developers.

Regarding the credibility of the model, we observe that the surveyed developers are representative of the project, resulting in threshold values with high confidence. In addition, since the majority of the effort is performed by those above the threshold, we could limit the overall margin of error by specifically focusing on those developers' activities. This also limits the challenges in effectively measuring the activity of occasional, sporadic contributors, since their contributions are relatively low. As we have seen with our case study, the threshold calculation allows to identify with precision many full-time developers that account for approximately 90% of the total activity. This means that we have a large share of the effort in a reasonable confidence interval, and can narrow the effect of the rest of contributors.

The model is concerned exclusively with effort performed by developers. There are many other tasks in a FOSS project that are not considered in it, such as translation activities, art & graphic design, or documentation, which may follow different procedures. The effort devoted to community management and other community-related activities are also beyond the scope of our model.

We envisage to expand this study by 1) studying other FOSS projects to ascertain if our method is applicable in general, and if it is to what extent; 2) performing a scientific experiment to obtain margins of error for the estimation of error for full-time developers; 3) investigating, by taking into consideration other FOSS projects, which of the two ways of measuring activity (commits or active days) is more suited for the model; 4) comparing our results with the ones provided by traditional software estimation models used in industry, such as COCOMO; 5) after quantifying the effort required in a FOSS project, is it more profitable for a prospective adopting company to redo ("make") their own system, or to invest ("buy") in the existing FLOSS system as is [3]? 6) comparing our approach with previous effort estimation techniques for FLOSS projects, as the one proposed in [6] based on the measure of entropy to calculate maintenance costs.

11. ACKNOWLEDGMENTS

The work of Gregorio Robles, Carlos Cervigón and Jesús M. González-Barahona has been funded in part by the Spanish Government under project SobreSale (TIN2011-28110). The work of Daniel Izquierdo has been funded in part by the Torres Quevedo program (PTQ-12-05577). Our gratitude to all the OpenStack developers for their help and feedback.

12. REFERENCES

[1] J. J. Amor, G. Robles, and J. M. Gonzalez-Barahona. Effort estimation by characterizing developer activity. In *Proceedings of the 2006 international workshop on Economics driven software engineering research*, pages 3–6. ACM, 2006.

[2] P. Anbalagan and M. Vouk. On predicting the time taken to correct bug reports in Open Source projects. In *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on*, pages 523–526. IEEE, 2009.

[3] J. Asundi. The need for effort estimation models for open source software projects. *ACM SIGSOFT Software Engineering Notes*, 30(4):1–3, 2005.

[4] C. Bird and T. Zimmermann. Assessing the value of branches with what-if analysis. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, page 45. ACM, 2012.

[5] A. Capiluppi and D. Izquierdo-Cortázar. Effort estimation of FLOSS projects: a study of the Linux kernel. *Empirical Software Engineering*, 18(1):60–88, 2013.

[6] E. Capra, C. Francalanci, and F. Merlo. The economics of open source software: an empirical analysis of maintenance costs. In *Software Maintenance, 2007. ICSM 2007. IEEE International Conference on*, pages 395–404. IEEE, 2007.

[7] E. Capra, C. Francalanci, and F. Merlo. An empirical study on the relationship between software design quality, development effort and governance in Open

Source Projects. *Software Engineering, IEEE Transactions on*, 34(6):765–782, 2008.

[8] E. Capra, C. Francalanci, and F. Merlo. The economics of community open source software projects: an empirical analysis of maintenance effort. *Advances in Software Engineering*, 2010, 2010.

[9] L. Dahlander and M. G. Magnusson. Relationships between open source software companies and communities: Observations from Nordic firms. *Research Policy*, 34(4):481–493, 2005.

[10] J. Fernandez-Ramil, D. Izquierdo-Cortazar, and T. Mens. What does it take to develop a million lines of Open Source code? In *Open Source Ecosystems: Diverse Communities Interacting*, pages 170–184. Springer, 2009.

[11] B. Fitzgerald. The transformation of Open Source Software. *Mis Quarterly*, pages 587–598, 2006.

[12] R. A. Ghosh, R. Glott, B. Krieger, and G. Robles. Free/libre and open source software: Survey and study, 2002.

[13] J. Gonzalez-Barahona, G. Robles, D. Izquierdo, and S. Maffulli. Using software analytics to understand how companies interact in free software communities. *IEEE software*, 30(5):38–45, 2013.

[14] J. M. González-Barahona and G. Robles. On the reproducibility of empirical software engineering studies based on data retrieved from development repositories. *Empirical Software Engineering*, 17(1-2):75–89, 2012.

[15] I. Herraiz, G. Robles, J. J. Amor, T. Romera, and J. M. González Barahona. The processes of joining in global distributed software projects. In *Proceedings of the 2006 international workshop on Global software development for the practitioner*, pages 27–33. ACM, 2006.

[16] M. Jorgensen and M. Shepperd. A systematic review of software development cost estimation studies. *Software Engineering, IEEE Transactions on*, 33(1):33–53, 2007.

[17] E. Kalliamvakou, G. Gousios, D. Spinellis, and N. Pouloudi. Measuring developer contribution from software repository data. *MCIS*, 2009:4th, 2009.

[18] S. Koch and G. Schneider. Effort, co-operation and co-ordination in an open source software project: Gnome. *Information Systems Journal*, 12(1):27–42, 2002.

[19] E. Kouters, B. Vasilescu, A. Serebrenik, and M. G. van den Brand. Who's who in GNOME: Using LSA to merge software repository identities. In *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*, pages 592–595. IEEE, 2012.

[20] 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 (TOSEM)*, 11(3):309–346, 2002.

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

[22] D. Riehle. The economic case for Open Source foundations. *Computer*, 43(1):86–90, 2010.

[23] S. K. Shah. Motivation, governance, and the viability of hybrid forms in open source software development. *Management Science*, 52(7):1000–1014, 2006.

[24] G. Von Krogh, S. Spaeth, and K. R. Lakhani. Community, joining, and specialization in open source software innovation: a case study. *Research Policy*, 32(7):1217–1241, 2003.

[25] L. Yu. Indirectly predicting the maintenance effort of Open-Source Software. *Journal of Software Maintenance and Evolution: Research and Practice*, 18(5):311–332, 2006.

Apéndice F

Resumen de los resultados de la encuesta a OpenStack

En las siguientes transparencias se muestra un resumen de los resultados obtenidos con la encuesta realizada a los desarrolladores de OpenStack. Los resultados completos pueden encontrarse en el paper enviado al MSR 2014.

Estimating Development Effort in FOSS Projects by Mining Software Repositories - Openstack Survey Results

Gregorio Robles, Universidad Rey Juan Carlos
Carlos Cervigón-Ávila, Universidad Rey Juan Carlos
Andrea Capiluppi, Brunel University

Executive Summary

- We propose a model to quantify the effort invested in a FOSS project by obtaining data from the versioning system.
- Our model is based on identifying full-time developers. To every full-time developer, we assign 1 person-month of effort each month.
- A survey with OpenStack developers has been used to validate our results. Over 100 OpenStack developers participated in the survey.
- Our analysis shows that the minimum estimation error is when we consider those OpenStack developers with over 12 commits every 6 months as full-time.
- A total of 9,000 person-months (750 person-years) for the development of the OpenStack project since its beginnings has been estimated.
- ~2,650 person-months have been invested during the last six months of 2013. This implies that we estimate that the actual effort on OpenStack has been around 440 person-months during the last 6 months.

2

About The Model

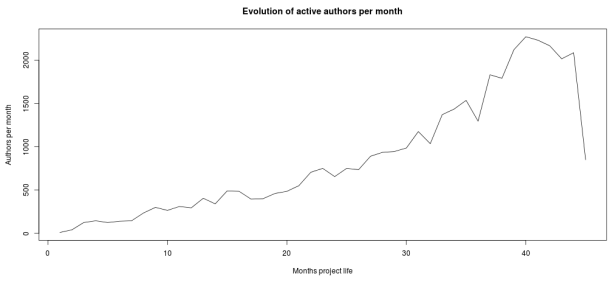
- Activity of a developer → effort measured in person-month
- Full-time developers → 40 hours/week → 1 person-month of effort each month
- Assumptions:
  - We can identify with high precision those developers working **full-time** in a FOSS project
  - The number of contributions by **full-time developers** is a large share of the total number of contributions
- We have used two possible measures for each developer:
  - Commits/month
  - Active days/month
- The model is based on finding a **threshold value "t"** for the number of commits (or active days) for which **we indentify full-time developers with a minimum error**
- The estimation error made for full-time developers **is much lower than** the error made for the rest of developers

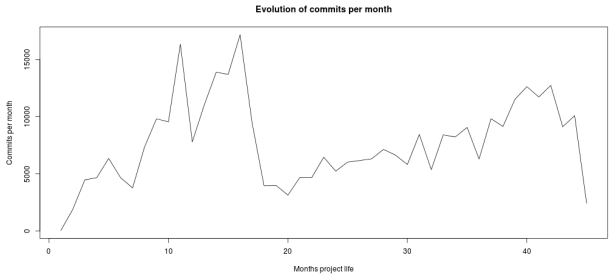
3

About OpenStack

- Young project with over 200 companies involved.
- 80% of the commits have been done by less than 8% of the overall authors.
- 90% of the commits have been done by about 17% of the overall authors.
- The OpenStack Fundation estimates that around 250 developers work professionally in the project.

First release	October 2010
# Authors (distinct)	1410
# Commits	89,871
# Commits (without bots)	68,587
SLoC (approx.)	1,650,000





4

Survey Results

- **1407 personalized emails** were sent:
  - About **300** emails bounced
  - **131 developers replied** the survey
  - **5** answers were removed because they were **empty** or we could see from the answer in the free text box that the respondent had **misunderstood/misinterpreted** the questions
  - We also changed **7** surveys to **normalize** the set, mainly because some respondents devoted 40 or more hours a week to the project mostly in programming, but stated they were occasional/part-time developers
  - To analyze if the survey is representative of the project we have used the Wilcoxon signed-rank test. The result is that the survey in fact **is respresentative** of the project, therefore we can use these values.

5

Analysis

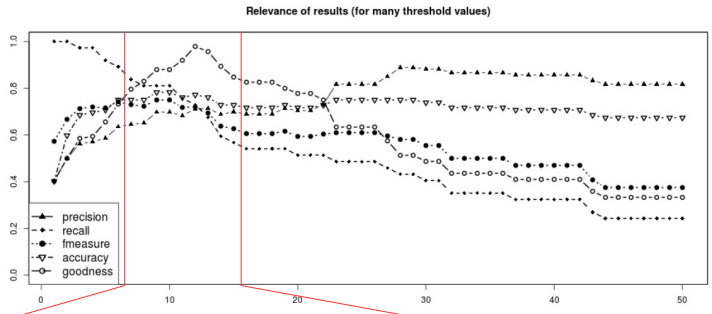
- To analyze the model we compare the classification of the authors (full-time or not full-time) from analyzing the project with the result obtained from the survey.
- Indicators used:
  - True positive (tp) → When we identify a full-time developer correctly
  - False negative (fn) → When we identify a full-time developer as non-full-time
  - True negative (tn) → When we identify a non-full-time developer correctly
  - False positive (fp) → When we identify a non-full-time developer as full-time
- From these indicators we calculate following five measures:
  - Precision
  - Recall
  - F-measure
  - Accuracy
  - Goodness (because false negatives compensate false positives!)

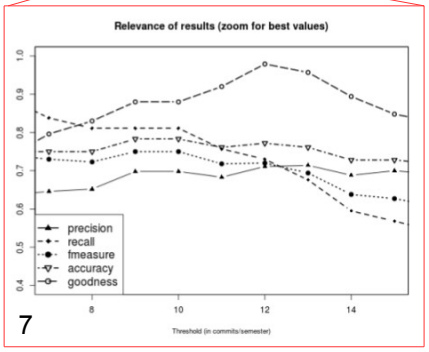
$$Accuracy = \frac{t_p + t_n}{t_p + t_n + f_p + f_n} \qquad Goodness = 1 - \frac{|(t_p + f_p) - (tp + fn)|}{tp + fn + fp}$$

- We analyze 50 different values of the threshold “t” between 1 to 50.

6

Analysis Results



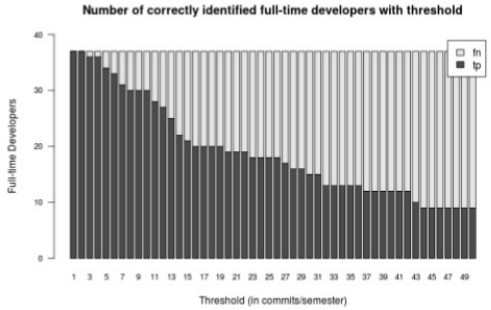


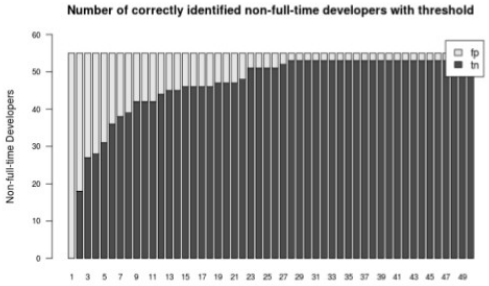
7

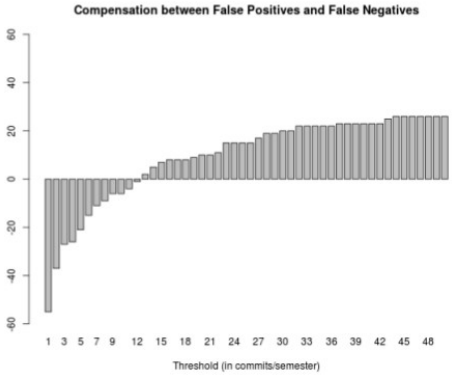
In these figures we can observe that for OpenStack the best results are obtained when the threshold "t" is between 9 and 12 commits every six months.

In this period the accuracy and goodness values achieve their maximum. From this value onwards, results get worse because the number of false positives increases.

Analysis Results







For values of "t" between 9 and 12, the model not only gets the best values for accuracy and goodness, the number of false positives and false negatives is compensated as well, mitigating the error that we obtain for the overall effort.

8

Effort Results

t	Effort	10s2	11s1	11s2	12s1	12s2	13s1	13s2
1	17400	426	816	1242	1890	2742	4368	5916
2	15255	420	813	1170	1647	2385	3783	5037
3	13804	402	798	1110	1486	2140	3384	4484
4	12747	389	785	1056	1383	1964	3074	4098
5	11891	378	768	1008	1295	1830	2821	3791
6	11213	370	757	964	1224	1726	2618	3554
7	10629	362	743	927	1164	1635	2451	3346
8	10128	355	732	896	1112	1560	2310	3164
9	9683	345	721	866	1065	1493	2189	3003
10	9298	337	711	841	1025	1434	2086	2864
11	8957	330	700	819	987	1381	1997	2743
12	8655	324	690	800	955	1334	1919	2634
13	8370	318	680	782	924	1289	1847	2531
14	8112	313	672	767	896	1247	1781	2437
15	7876	308	663	753	872	1208	1721	2351
16	7662	303	656	740	850	1173	1666	2275
17	7466	298	648	729	830	1140	1615	2206
18	7284	294	641	719	812	1107	1568	2142
19	7109	291	634	708	794	1077	1522	2083
20	6945	288	628	698	777	1048	1481	2025

Table 3: Estimated total effort (in PM), given per threshold value (column t). Estimated effort values for all semesters.

Assuming a threshold value t of 12:

- The estimation of effort invested in OpenStack lies around **9,000 person-months in total** (750 person-years).
- 2,634 person-months have been invested during the last six months. This implies that we estimate that the actual effort on OpenStack has been around **440 person-months during the last 6 months**.

The OpenStack foundation estimates that currently over 250 professional developers work on the project hired by companies, which backs these results.

9

Conclusions

- We faced two challenges, which have been achieved to a certain extent:
  - To design a simple method to measure the effort provided in a sparse, distributed and uneven development scenario, like the FOSS one.
  - To design a model so that it offers a reasonable prediction. This affects two aspects of the model:
    - The activity threshold to consider a developer as full-time or not.
    - The credibility of the model.
- We have seen that the design of the model allows to compensate erroneous classification of developers.
- Since the majority of the effort is performed by those above the threshold, we could limit the overall margin of error by specifically focusing on those developers' activities.
- The model is concerned exclusively with effort performed by developers. There are many other tasks in a FOSS project that are not considered in it.

10

Future Work

- We envisage to expand this study by...
  - studying other FOSS projects (WebKit, Linux, WikiMedia, among others).
  - performing a scientific experiment to obtain margins of error.
  - investigating which of the two ways of measuring activity (commits or active days) is more suited for the model.
  - comparing our results with the ones provided by traditional software estimation models used in industry, such as COCOMO.
  - quantifying the effort required in a FOSS project to see if is it more profitable for a prospective adopting company to redo ("make") their own system or to invest ("buy") in the existing FOSS system.
  - comparing our approach with previous effort estimation techniques for FOSS projects.

11

Contact

- More information can be found at the survey website:

  http://ccervigon.libresoft.es

- For the academic publication that contains the details of our research, refer to:

  Gregorio Robles, Carlos Cervigón-Ávila, Jesús M. González-Barahona, Andrea Capiluppi, Daniel Izquierdo-Cortázar: "Estimating Development Effort in Free/Open Source Software Projects by Mining Software Repositories – A Case Study on OpenStack". Proceedings of the 11th Working Conference on Mining Software Repositories, Hyderabad, India, June 2014.

- If you want to get in contact with the authors of this study / survey, please contact:

  Gregorio Robles – grex (at) gsyc.urjc.es

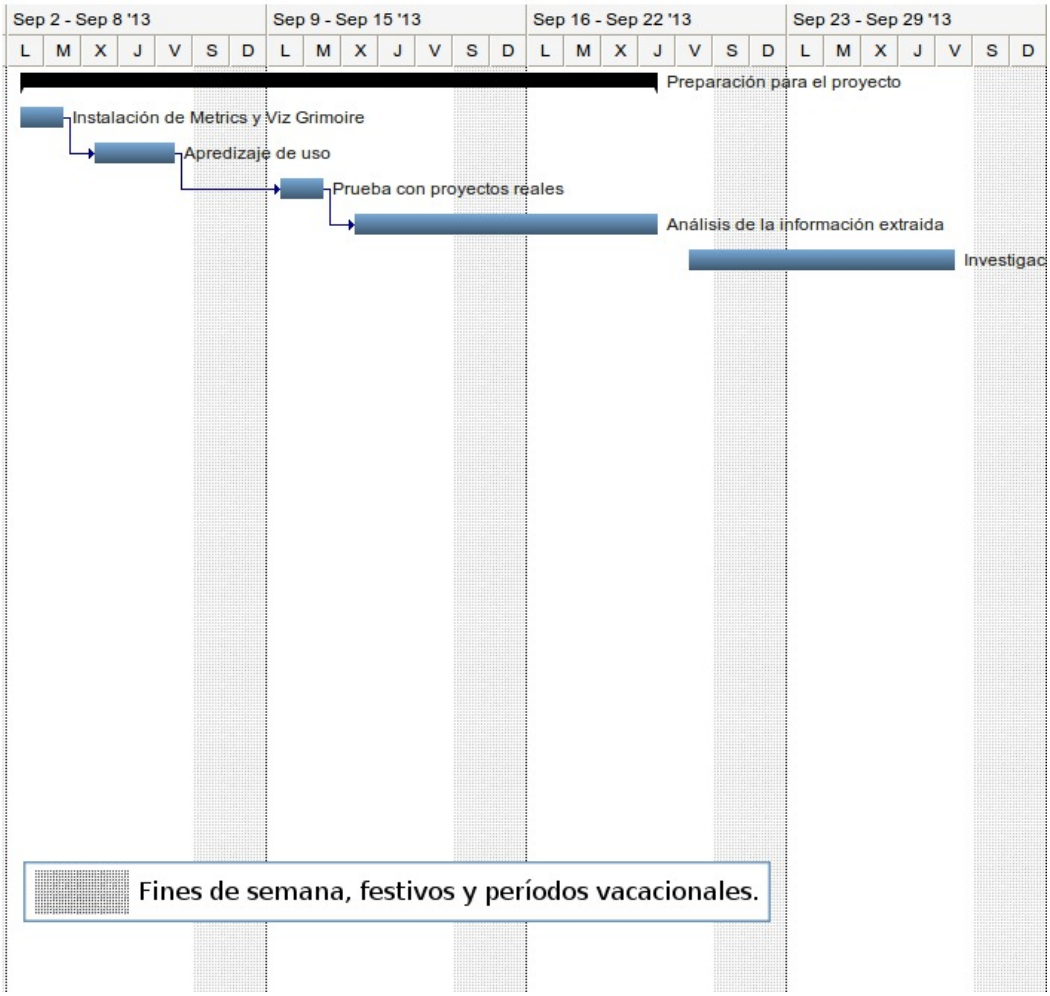
12

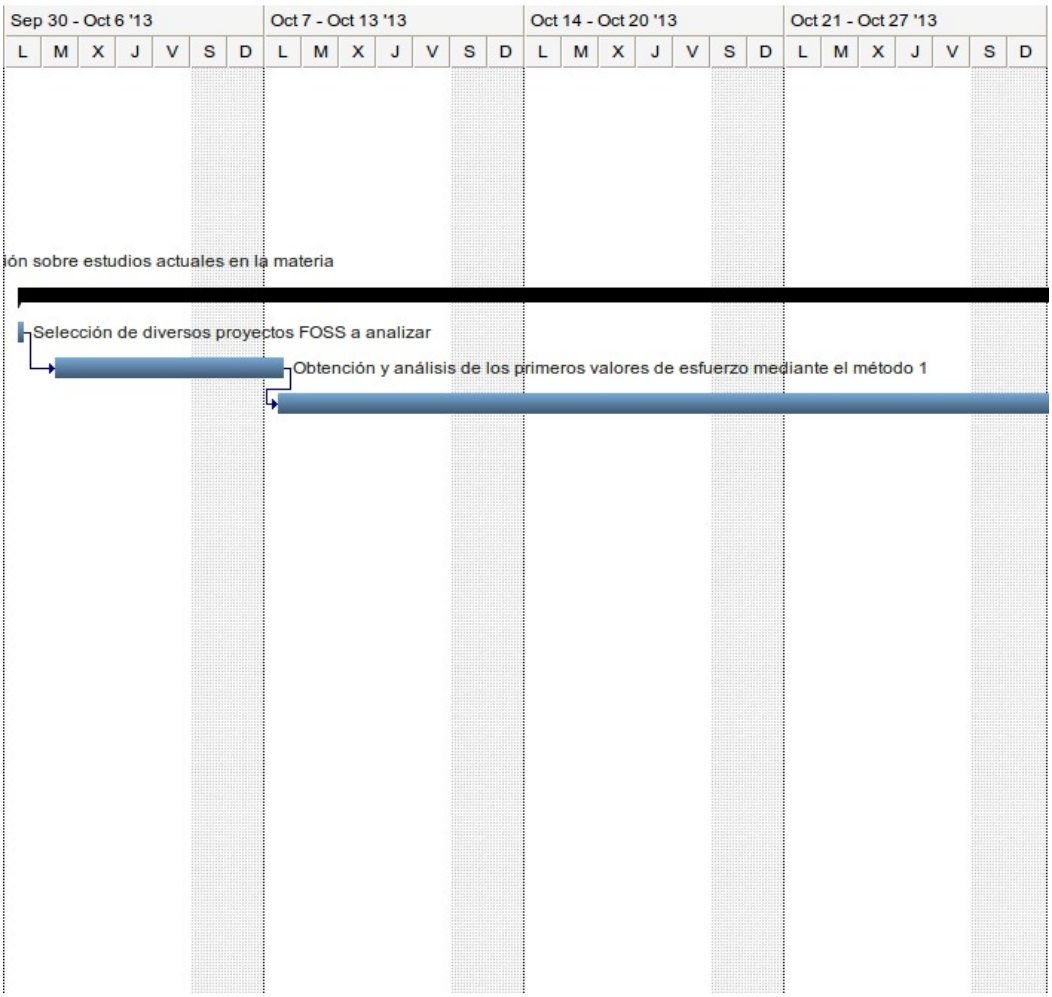
Apéndice G

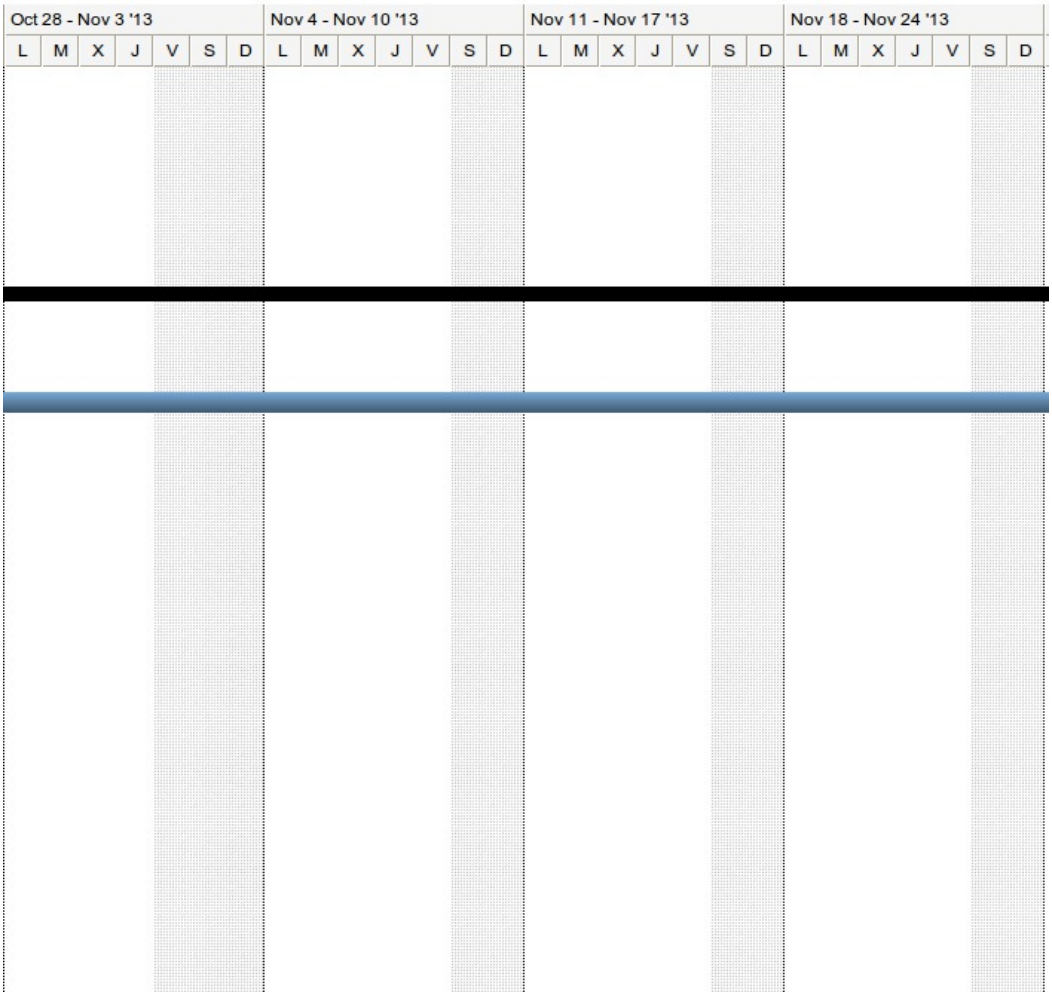
Diagrama GANTT

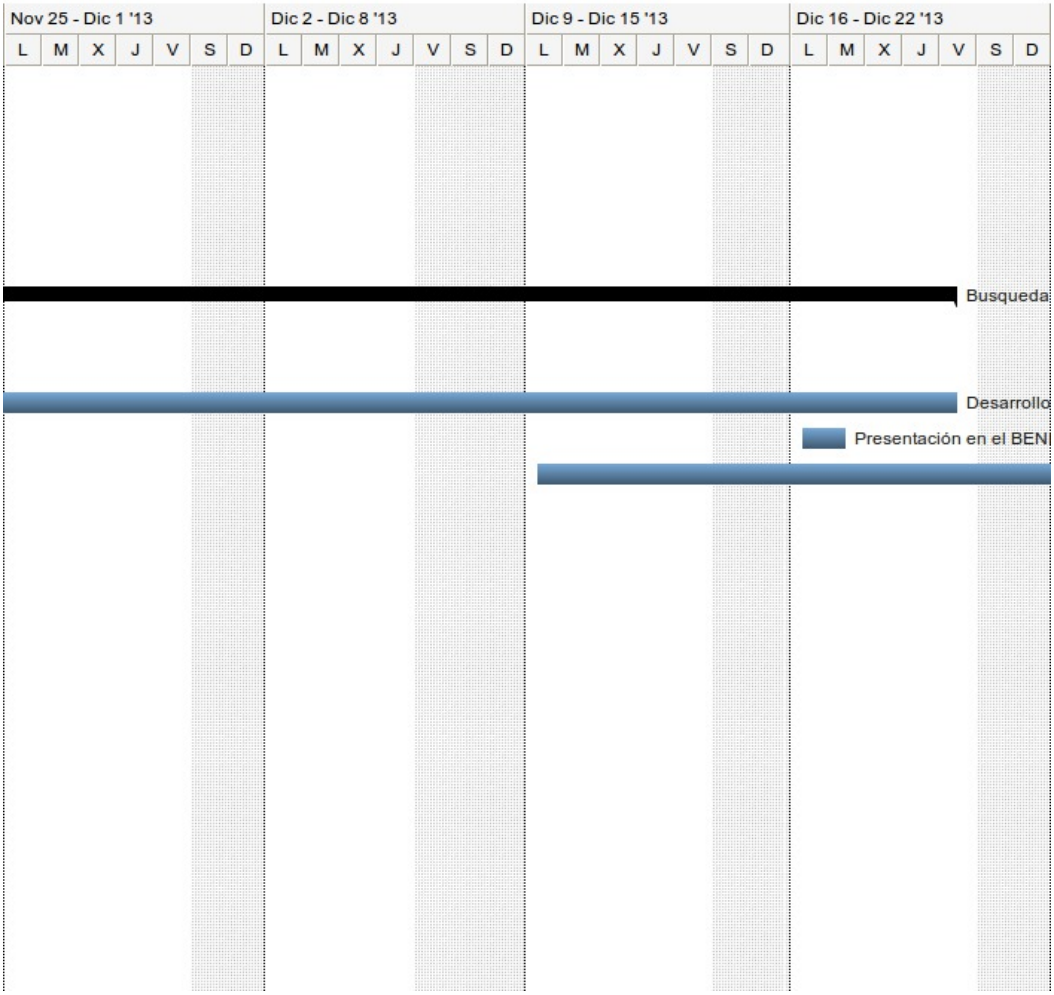
Diagrama GANTT de las tareas realizadas durante el proyecto. Así como los periodos en los que han sido realizadas y las conexiones existentes entre ellas.

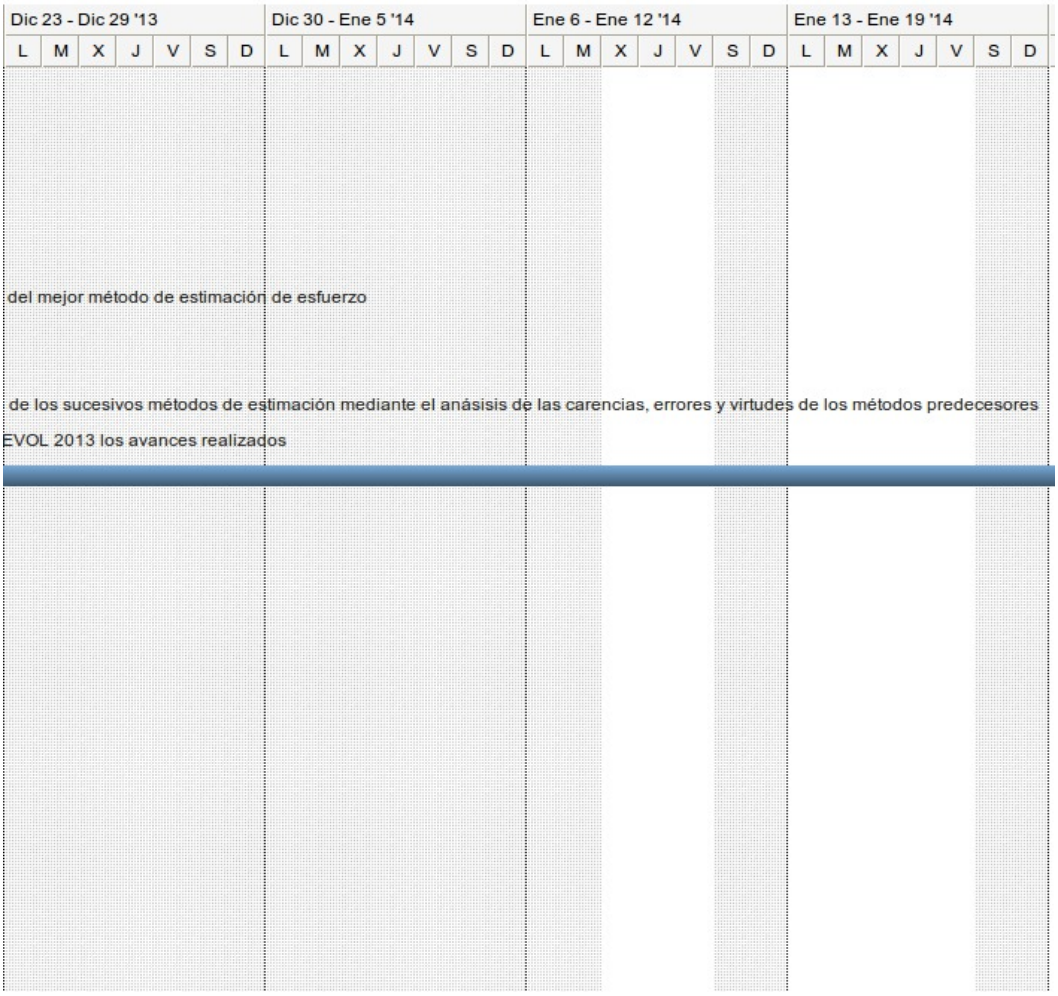
	Nombre	Duración	Inicio	Fin
1	**Preparación para el proyecto**	**14d**	**02/09/2013**	**19/09/2013**
2	Instalación de Metrics y Viz Grimoire	2d	02/09/2013	03/09/2013
3	Apredizaje de uso	3d	04/09/2013	06/09/2013
4	Prueba con proyectos reales	2d	09/09/2013	10/09/2013
5	Análisis de la información extraida	7d	11/09/2013	19/09/2013
6	Investigación sobre estudios actuales en la materia	6d	20/09/2013	27/09/2013
7	**Busqueda del mejor método de estimación de esfuerzo**	**58d**	**30/09/2013**	**20/12/2013**
8	Selección de diversos proyectos FOSS a analizar	1d	30/09/2013	30/09/2013
9	Obtención y análisis de los primeros valores de esfuerzo mediante el método 1	5d	01/10/2013	07/10/2013
10	Desarrollo de los sucesivos métodos de estimación mediante el anásisis de las car	53d	07/10/2013	20/12/2013
11	Presentación en el BENEVOL 2013 los avances realizados	2d	16/12/2013	17/12/2013
12	Elaboración, mantenimiento y actualización de la encuesta	72d	09/12/2013	04/04/2014
13	**Encuesta a OpenStack**	**7d**	**30/01/2014**	**07/02/2014**
14	Envio	1d	30/01/2014	30/01/2014
15	Soporte a desarrolladores	3d	30/01/2014	03/02/2014
16	Recolección de respuestas	1d	03/02/2014	03/02/2014
17	Análisis y elaboración de informe (abstract para el MSR 2014)	5d	03/02/2014	07/02/2014
18	Hightlight del informe de OpenStack	2d	03/03/2014	04/03/2014
19	**Encuesta a Linux, WebKit, Moodle y Ceph**	**52d**	**17/02/2014**	**09/05/2014**
20	Análisis de los datos obtenidos mediante MetricsGrimoire y solución de errores	20d	17/02/2014	14/03/2014
21	Envío de la encuesta	1d	25/03/2014	25/03/2014
22	Soporte a desarrolladores	14d	25/03/2014	11/04/2014
23	Envio de recordatorio de encuesta	1d	01/04/2014	01/04/2014
24	Recoleción de respuestas	1d	11/04/2014	11/04/2014
25	Análisis de resultados	5d	05/05/2014	09/05/2014
26	Elaboración de la memoria del PFC	37d	31/03/2014	30/05/2014

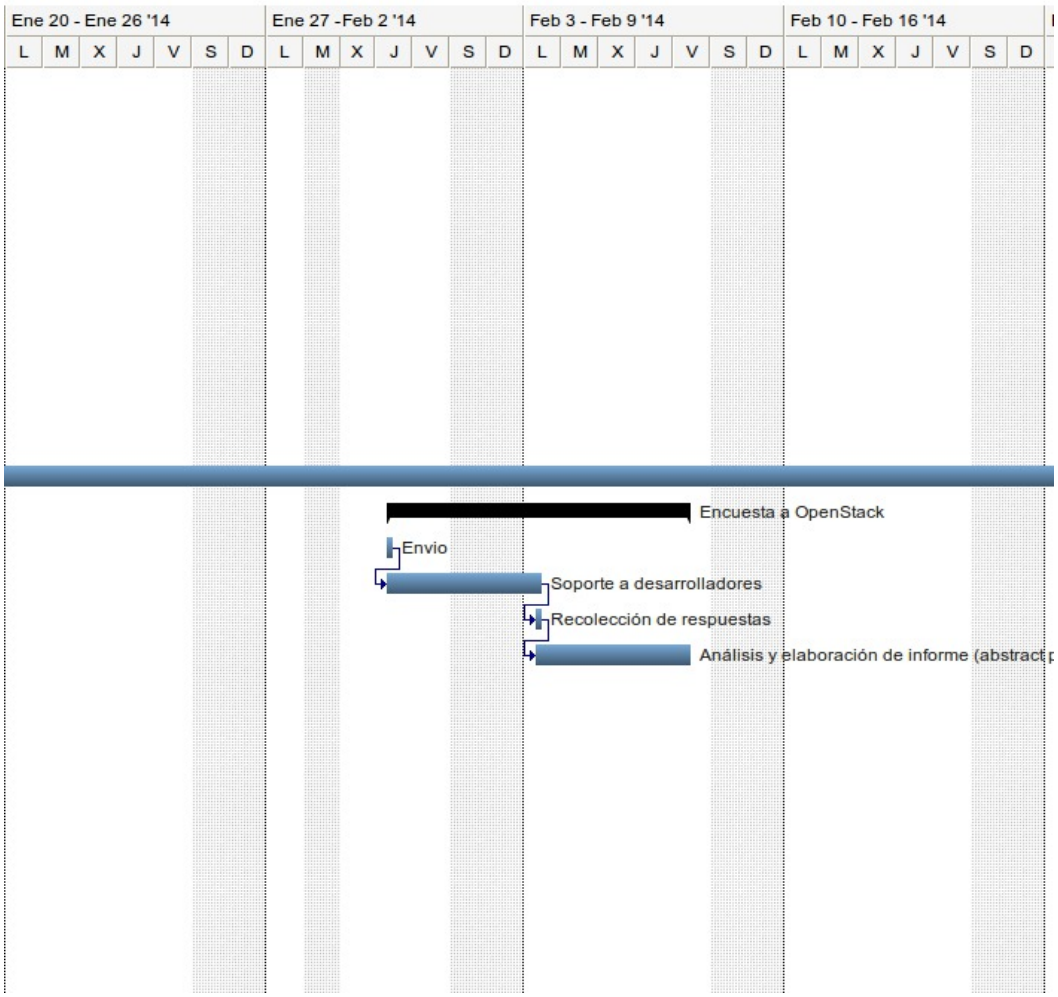

Sep 2 - Sep 8 '13
Sep 9 - Sep 15 '13
Sep 16 - Sep 22 '13
Sep 23 - Sep 29 '13
L M X J V S D
Preparación para el proyecto
Instalación de Metrics y Viz Grimoire
Apredizaje de uso
Prueba con proyectos reales
Análisis de la información extraida
Investigac
Fines de semana, festivos y períodos vacacionales.

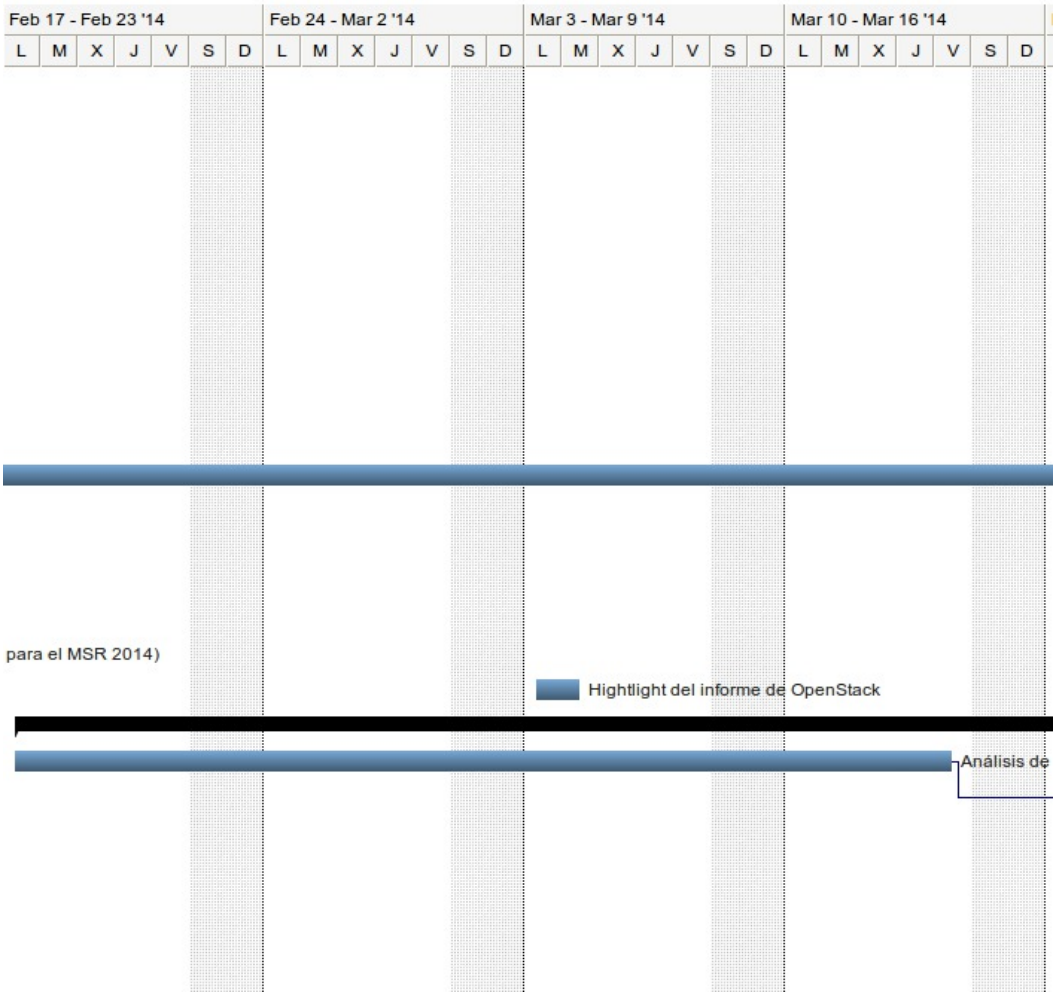

Sep 30 - Oct 6 '13
Oct 7 - Oct 13 '13
Oct 14 - Oct 20 '13
Oct 21 - Oct 27 '13
L M X J V S D
ión sobre estudios actuales en la materia
Selección de diversos proyectos FOSS a analizar
Obtención y análisis de los primeros valores de esfuerzo mediante el método 1

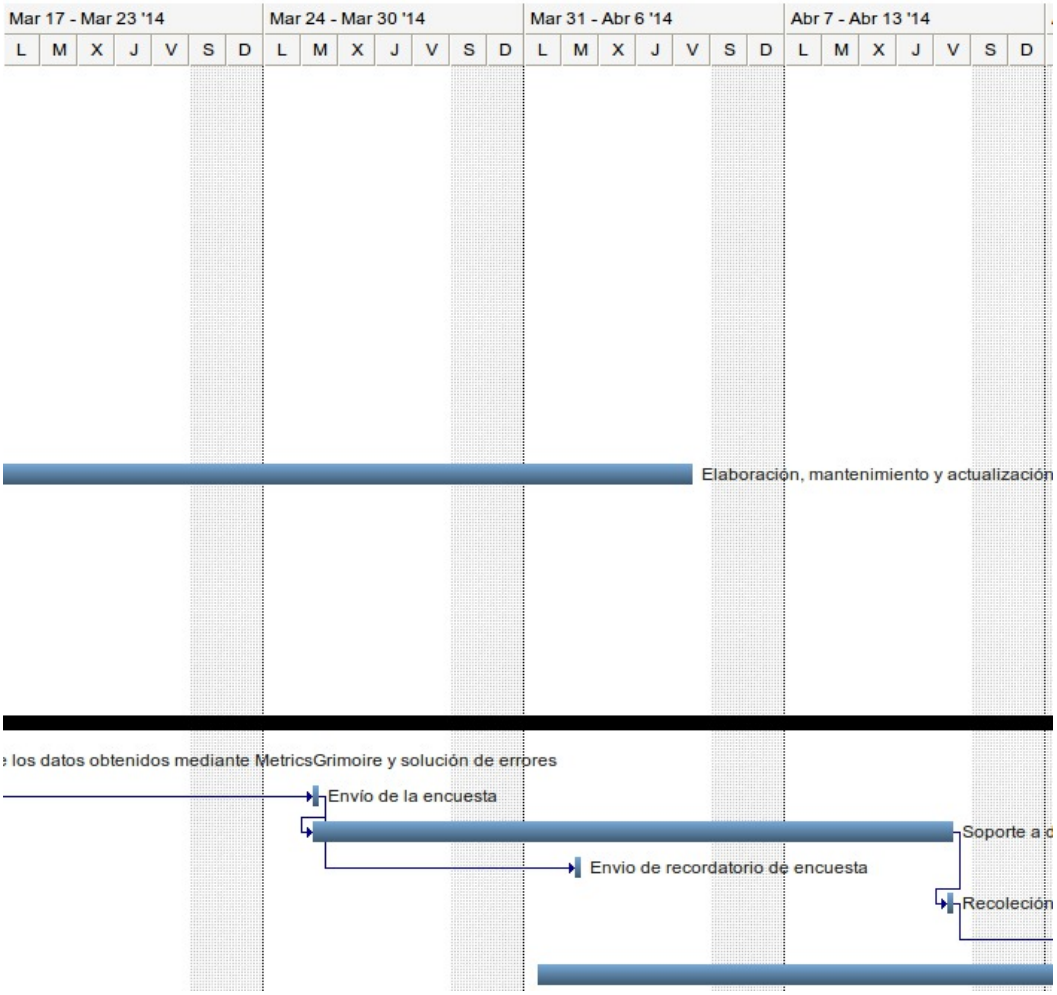

Oct 28 - Nov 3 '13
Nov 4 - Nov 10 '13
Nov 11 - Nov 17 '13
Nov 18 - Nov 24 '13
L M X J V S D

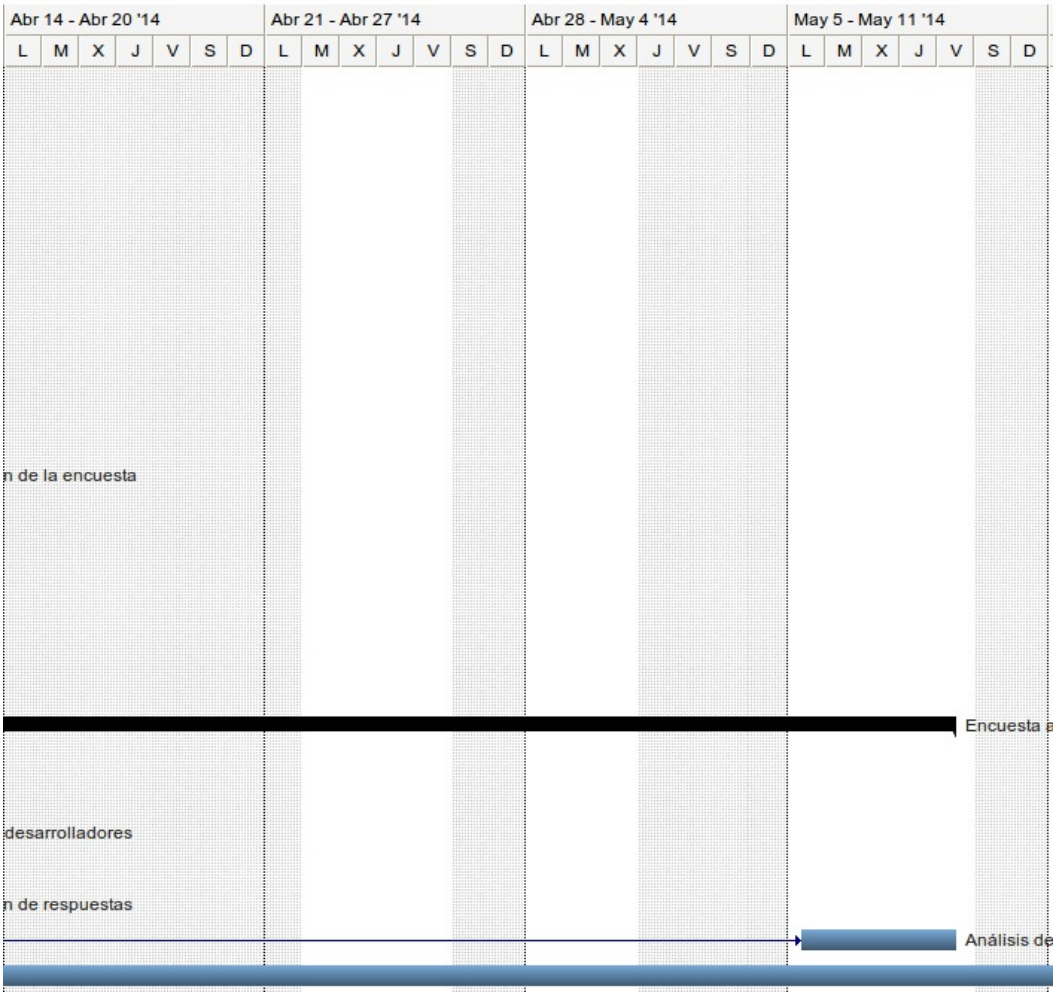

Nov 25 - Dic 1 '13
Dic 2 - Dic 8 '13
Dic 9 - Dic 15 '13
Dic 16 - Dic 22 '13
L M X J V S D
Busqueda
Desarrollo
Presentación en el BEN

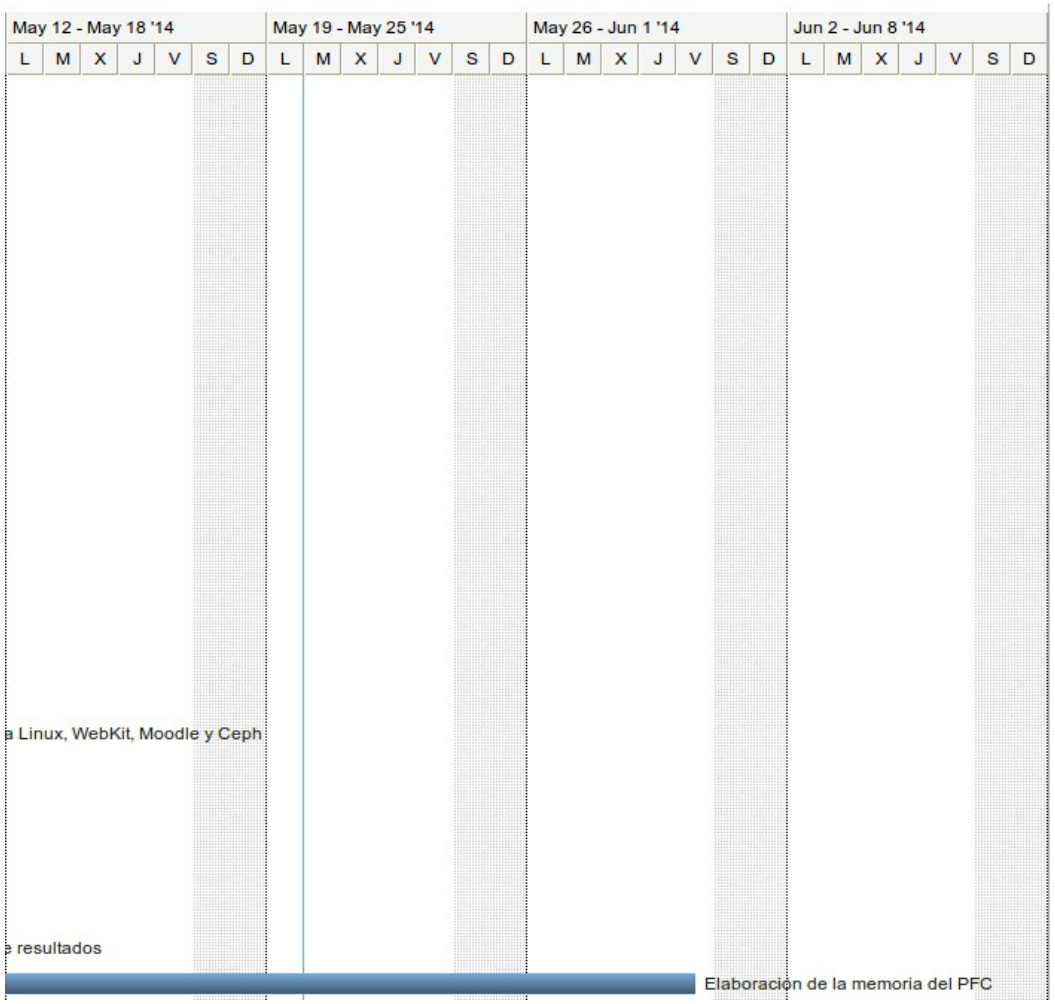

Dic 23 - Dic 29 '13
Dic 30 - Ene 5 '14
Ene 6 - Ene 12 '14
Ene 13 - Ene 19 '14
L M X J V S D
del mejor método de estimación de esfuerzo
de los sucesivos métodos de estimación mediante el anásisis de las carencias, errores y virtudes de los métodos predecesores
EVOL 2013 los avances realizados


Ene 20 - Ene 26 '14
Ene 27 -Feb 2 '14
Feb 3 - Feb 9 '14
Feb 10 - Feb 16 '14
L M X J V S D
Encuesta a OpenStack
Envio
Soporte a desarrolladores
Recolección de respuestas
Análisis y elaboración de informe (abstract


Feb 17 - Feb 23 '14
Feb 24 - Mar 2 '14
Mar 3 - Mar 9 '14
Mar 10 - Mar 16 '14
L M X J V S D
para el MSR 2014)
Hightlight del informe de OpenStack
Análisis de


Mar 17 - Mar 23 '14
Mar 24 - Mar 30 '14
Mar 31 - Abr 6 '14
Abr 7 - Abr 13 '14
L M X J V S D
Elaboración, mantenimiento y actualización
los datos obtenidos mediante MetricsGrimoire y solución de errores
Envío de la encuesta
Soporte a
Envio de recordatorio de encuesta
Recolección


Abr 14 - Abr 20 '14
Abr 21 - Abr 27 '14
Abr 28 - May 4 '14
May 5 - May 11 '14
L M X J V S D
n de la encuesta
Encuesta
desarrolladores
n de respuestas
Análisis de


May 12 - May 18 '14
May 19 - May 25 '14
May 26 - Jun 1 '14
Jun 2 - Jun 8 '14
L M X J V S D
a Linux, WebKit, Moodle y Ceph
e resultados
Elaboración de la memoria del PFC

Apéndice H

Versión digital de la memoria y programas empleados

Al tratarse de un proyecto de software libre, tanto la memoria cómo el código empleado en el proyecto es accesible públicamente. Por lo que ambos pueden encontrarse en las siguientes direcciones:

Memoria en PDF:

`http://gsyc.urjc.es/~grex/pfcs/2014-carlos-cervigon`

Código empleado en el proyecto:

`https://github.com/ccervigon/codigo_pfc`
`https://github.com/ccervigon/survey_creator`

Bibliografía

[1] Bicho. Repositorio oficial de Bicho. https://github.com/MetricsGrimoire/Bicho/.

[2] Bitergia. Página oficial de Bitergia. http://bitergia.com/.

[3] Cvsanaly. Repositorio oficial de CVSAnalY. https://github.com/MetricsGrimoire/CVSAnalY/.

[4] Django. Página oficial de Django. https://www.djangoproject.com/.

[5] Free Software Fundation. Página oficial de la Free Software Fundation, https://www.fsf.org/.

[6] General Public License. Página con los derechos de la licencia GPL. https://www.gnu.org/copyleft/gpl.html.

[7] Libresoft. Página oficial de Libresoft. http://www.libresoft.es/.

[8] Matplotlib. Página oficial de Matplotlib. http://matplotlib.org/.

[9] Metricsgrimoire. Página oficial de MetricsGrimoire. http://metricsgrimoire.github.io/.

[10] Mysql. Página oficial de MySQL. http://www.mysql.com/.

[11] Numpy. Página oficial de NumPy. http://www.numpy.org/.

[12] Open Source Initiative. Página oficial de la Open Source Initiative, http://opensource.org/.

[13] Python. Página oficial de Python. https://www.python.org/.

[14] R project. Página oficial de R Project. http://www.r-project.org/.

[15] Repositorio cran. Repositorio oficial de CRAN. http://www.cran.r-project.org/web/packages/.

[16] Repositoryhandler. Repositorio oficial de RepositoryHandler. https://github.com/MetricsGrimoire/RepositoryHandler/.

[17] Scipy. Página oficial de SciPy. http://www.scipy.org/.

[18] Sqlite. Página oficial de SQLite. https://sqlite.org/.

[19] Universidad Rey Juan Carlos. Página oficial de la Universidad Rey Juan Carlos. http://www.urjc.es/.

[20] Vizgrimoire. Página oficial de VizGrimoire. http://vizgrimoire.bitergia.org/.

[21] Vizgrimoireutils. Repositorio oficial de VizGrimoireUtils. https://github.com/VizGrimoire/VizGrimoireUtils/.

[22] Juan Jose Amor, Gregorio Robles, and Jesus M Gonzalez-Barahona. Effort estimation by characterizing developer activity. In *Proceedings of the 2006 international workshop on Economics driven software engineering research*, pages 3–6. ACM, 2006.

[23] Jai Asundi. The need for effort estimation models for open source software projects. *ACM SIGSOFT Software Engineering Notes*, 30(4):1–3, 2005.

[24] Barry W. Boehm. *Software Engineering Economics*. Prentice Hall, 1981.

[25] Barry W. Boehm, Chris Abts, A. Winsor Brown, Sunita Chulani, Bradford K. Clark, Ellis Horowitz, Ray Madachy, Donald J. Reifer, and Bert Steeco. *Software Cost Estimation with COCOMO II*. Prentice Hall, 2000.

[26] Andrea Capiluppi and Daniel Izquierdo-Cortázar. Effort estimation of FLOSS projects: a study of the Linux kernel. *Empirical Software Engineering*, 18(1):60–88, 2013.

[27] Andrea Capiluppi, Gregorio Robles, and Carlos Cervigón. Effort estimation based on mining software repositories - a preliminary approach. In *BENEVOL 2013*. http://informatique.umons.ac.be/genlog/benevol2013/BENEVOL_2013_Abstracts.pdf#page=50.

[28] Eugenio Capra, Chiara Francalanci, and Francesco Merlo. An empirical study on the relationship between software design quality, development effort and governance in Open Source Projects. *Software Engineering, IEEE Transactions on*, 34(6):765–782, 2008.

[29] Linus Dahlander and Mats G Magnusson. Relationships between open source software companies and communities: Observations from Nordic firms. *Research Policy*, 34(4):481–493, 2005.

[30] Brian Fitzgerald. The transformation of Open Source Software. *Mis Quarterly*, pages 587–598, 2006.

[31] Dirk Riehle. The economic case for Open Source foundations. *Computer*, 43(1):86–90, 2010.