UNIVERSIDAD
REY JUAN CARLOS

MASTER EN INGENIERIA DE TELECOMUNICACIONES

Curso Académico 2014/2015

Trabajo Fin de Máster

BUG SEEDING, IDENTIFICACION DEL ERROR Y LA IMPORTANCIA DEL COMMIT ANTECESOR

Autor : Gema Rodríguez Pérez

Tutor : Dr. Jesús María Gonzalez Barahona

Proyecto Fin de Carrera

FIXME: Título

Autor : Gema Rodríguez Pérez

Tutor : Dr. Jesús María Gonzalez Barahona

La defensa del presente Proyecto Fin de Carrera se realizó el día de de 2015, siendo calificada por el siguiente tribunal:

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Fuenlabrada, a de de 2015

Dedicado a

mi familia

II

Agradecimientos

Aquí vienen los agradecimientos... Aunque está bien acordarse de la pareja, no hay que olvidarse de dar las gracias a tu madre, que aunque a veces no lo parezca disfrutará tanto de tus logros como tú... Además, la pareja quizás no sea para siempre, pero tu madre sí.

Resumen

Estamos involucrados en un proyecto de investigación, y como en la mayoría de proyectos de este estilo, es esencial disponer de herramientas que nos faciliten el trabajo de una manera u otra. Por ese motivo, surgió la idea de este proyecto, cuyo objetivo final era desarrollar una herramienta enmarcada en el contexto anteriormente citado y que posteriormente sería validada con nuestro estudio de investigación.

La herramienta usa diversas tecnologías actuales, que en capítulos posteriores explicaremos, pero se basa principalmente en el uso de dos tecnologías, en el lado del servidor NodeJs y en el lado del cliente, JavaScript.

Aquí viene un resumen del proyecto. Ha de constar de tres o cuatro párrafos, donde se presente de manera clara y concisa de qué va el proyecto. Han de quedar respondidas las siguientes preguntas:

- ¿De qué va este proyecto? ¿Cuál es su objetivo principal?
- ¿Cómo se ha realizado? ¿Qué tecnologías están involucradas?
- ¿En qué contexto se ha realizado el proyecto? ¿Es un proyecto dentro de un marco general?

Lo mejor es escribir el resumen al final.

Summary

Here comes a translation of the “Resumen” into English. Please, double check it for correct grammar and spelling. As it is the translation of the “Resumen”, which is supposed to be written at the end, this as well should be filled out just before submitting.

Índice general

Índice de figuras

Capítulo 1

Introducción

En este proyecto presentamos una herramienta desarrollada con tecnologías actuales como son Node.js, JavaScript, Bootstrap, HTML5 ó CSS3 que nos ayudará posteriormente en un estudio de investigación, será el propio estudio quién valide la herramienta.

El estudio de investigación que estamos desarrollando actualmente será explicado con mayor detalle en la sección 6.2.2, ahora lo describiremos a grandes rasgos para poder entender el por qué del desarrollo de esta herramienta. Dentro del desarrollo software, el espíritu de las comunidades de software libre, ha permitido a los desarrolladores investigar como se lleva a cabo el proceso del desarrollo software. En la literatura actual, nuestra investiación se centra en el concepto de bug seeding commit [1]. Es decir, nuestro estudio se enfoca en descubrir el momento en el que un cambio en el código fuente provocó un error, y a partir de ahí, poder identificar al/los responsables de dicho error. Analizar el repositorio de errores de un proyecto, significa tener que repetir una metodología concreta que nos proporcione ciertas características que serán utilizadas posteriormente, obteniendo finalmente unas conclusiones y unos resultados.

En la primera fase del estudio, para poder definir correctamente una metodología, el análisis del repositorio se llevaba a cabo manualmente. Una vez que la metodología estaba acabada y contrastada, nos dimos cuenta que podríamos usar las tecnologías actuales para ayudarnos en nuestra investigación, automatizando una gran parte del análisis del repositorio. Por tanto, la herramienta que en este proyecto mostramos, se encarga de extraer características del repositorio de errores, proporcionar cierta información para tomar decisiones y guardar la información en otro repositorio. (Github)

[1] Momento en el que un error es introducido en el código

Para nuestro estudio, necesitamos analizar repositorios de código, de los cuales extraeremos una serie de características útiles para el estudio. La idea de desarrollar esta herramienta surgió con el fin de disminuir el tiempo de extracción de ciertas características.

Capítulo 2

Objetivos

2.1. Objetivo general

El objetivo general de este proyecto, es desarrollar una herramienta que posteriormente sea utilizada para el análisis de investigación que estamos llevando a cabo.

2.2. Objetivos específicos

La herramienta tiene que ser capaz de proporcionarnos mediante llamadas a su propia API, información extraída de los repositorios Launchpad y Gerrit. También, será necesario que el usuario tome decisiones a partir de la información que se presenta el la herramienta, por ello, será necesario que la herramienta proporcione una interfaz de usuario atractiva, agradable y sencilla, en la que el usuario pueda seleccionar la opción adecuada a partir de la información que en ella se despliega. Además, la herramienta debe ser capaz de albergar toda la información extraída para poder sacar posteriormente las gráficas y estadísticas correspondientes.

Debemos Hablar de que es OpenStack y como y donde guarda los tickets y la revision de código.

Creo que aquí deberíamos escribir que opciones tecnológicas tenemos para poder desarrollar el servidor, el cliente, la interfaz de usuario, la ”base de datos (github) . . . Y el porque hemos elegido cada una de las tecnologías.

Capítulo 3

Estado de la técnica

Descripción de las tecnologías que utilizas en tu trabajo. Con dos o tres párrafos por cada tecnología, vale.

Puedes citar libros, como el de Bonabeau et al. sobre procesos estigmérgicos [**?**].

LibreSoft[1].

3.1. OpenStack

OpenStack es el nombre asociado a un proyecto software, integramente .ºpen-Source.º código abierto, cuyo propósito es combinar una serie de proyectos para construir y manejar plataformas de computación en la nube, para nubes tanto publicas como privadas. Principalmente los usuarios lo desarrollan como una infraestructura como servicio, comúnmente conocido como IaaS (Infrastructure as a Service), permitiéndole orquestar y gestionar distintos aspectos . Se distribuye bajo la licencia de Apache y es software libre y distribuido.

Un aspecto que hace interesante a OpenStack es su capacidad de extensibilidad a través de sencillas APIs, este factor ha hecho posible que muchos proveedores de servicios usen OpenStack como elemento clave de su infraestructura. OpenStack se estaá convirtiendo en una referencia de implementación al construir nubes privadas y póblicas, se encuentra respaldado por grandes empresas como Cisco, Paypal, Webex, Ebay, NASA, Rackspace, Dell, HP, Comcast, Seagate, Intel, AT&T y NetApp.

OpenStack ha conseguido, en solo tres años, convertirse en el mayor proyecto libre de IaaS,

[1] `http://www.libresoft.es`

su éxito puede ser debido al poderoso framework que han creado, haciendo mas cómodo la infraestructura IaaS.

OpenStack se compone proyectos claramente identificados como parte del núcleo ó 'core', los cuales están relacionados entre sí y dependen de la versión que se esté utilizando. A continuación se detallan los componentes incluidos en la versión Juno (16 Octubre del 2014), ya que nuestro estudio estando siendo usada esa versión:

- Compute (Nova): Proporciona máquinas virtuales bajo demanda, también permite gestionar volúmenes de disco a través de uno de sus servicios. él es el controlador de la estructura básica del Cloud. Se encarga de iniciar las instancias de los usuarios y grupos.

- Object Store (Swift): Proporciona almacenamiento de objetos. Nos permite almacenar y recuperar ficheros, realizar copias de seguridad, almacenamiento de audio/vídeo en streamings, etc. Pero no podemos montar sistemas de ficheros basados en NFS. En Swift se incluyen un servidor proxy, un servidor de cuentas de usuario y ciertos procesos ejecutados periódicamente para limpiar datos.

- Block Storage (Cinder): Proporciona almacenamiento de bloques que se usan en las instancias de OpenStack. Complementa al almacenamiento que nos proporciona Swift.

- Networking (Neutron): Proporciona conectividad de red como servicio, de tal forma que permite alta flexibilidad a los usuarios finales para interconectar y crear sus propias redes.

- Dashboard (Horizon): Proporciona una aplicación web para el manejo de instancias y volúmenes. Permite la comunicación entre las distintas APIs de OpenStack de forma sencilla. No proporciona toda la funcionalidad que podríamos conseguir usando un interprete de comandos, pero todo lo que hace lo hace correcto.

- Identity (Keystone): Servicio usado para la autentificación entre el resto de componentes. Utiliza un sistema de autentificación basado en tokens.

- Image Service (Glance): Proporciona la búsqueda y recuperación de máquinas virtuales.

- Telemetry Service (Ceilometer) Permite recoger de forma fiable las mediciones de la utilización de recursos físicos y virtuales que comprenden las Clouds que se encuentran

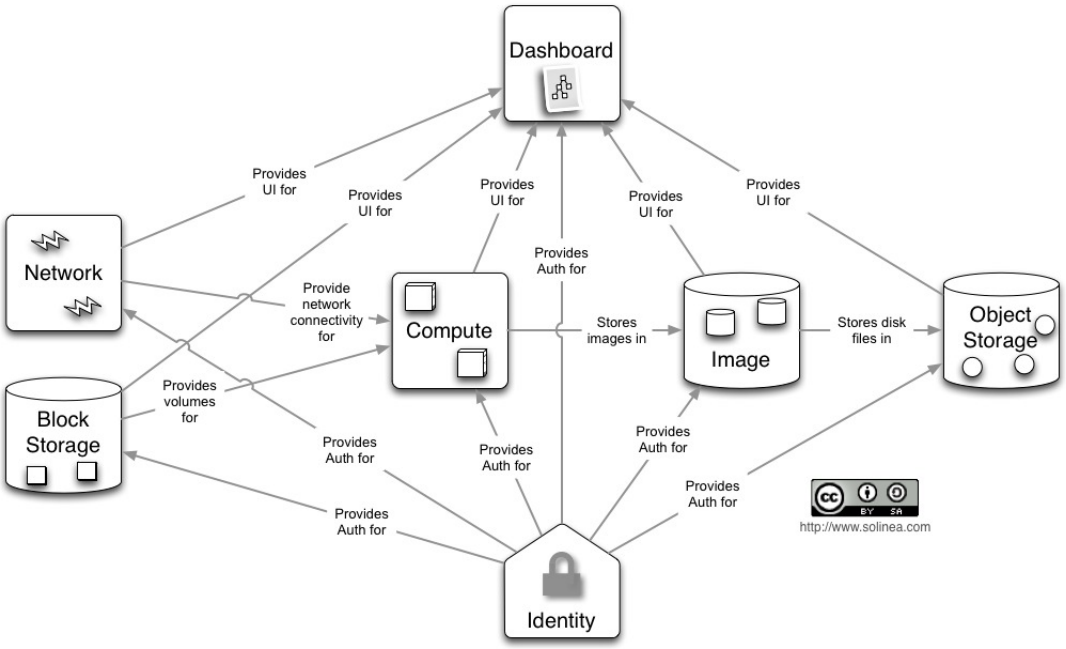

Figura 3.1: Relaciones entre los servicios de OpenStack

desplegadas. Se analizan los datos y se desencadena acciones cuando ciertos criterios definidos se cumplen.

- Orchestration (Heat): Proporciona ayuda para manejar la infraestructura que se necesita para un servicio de Cloud, para ello realiza peticiones REST y Query a la API.

- Database Service (Trove): Proporciona una base de datos que funciona como un servio de aprovisionamiento de bases de stand relacionales y no relacionales.

- Data Processing (Sahara): Proporciona un medio sencillo para la provisión de un cluster de aplicaciones de datos intensivos en la parte superior de OpenStack.

La gran comunidad de OpenStack es la encargada oficial de mantener a punto estos sistemas. Cada uno de ellos tiene su propia API para alcanzar una mejor integración. Además, se encuentra bien documentada, soportada por numerosos fabricantes y distribuidores de software.

En la imagen 3.1 extraída de `http://activity.openstack.org/dash/browser/repository.html?repository=https%3A__bugs.launchpad.net_cinder&ds=its&release=juno` se presenta una visión muy simplificada acerca de la arquitectura de OpenStack, la imagen no muestra la interacción con los consumidores del Cloud.

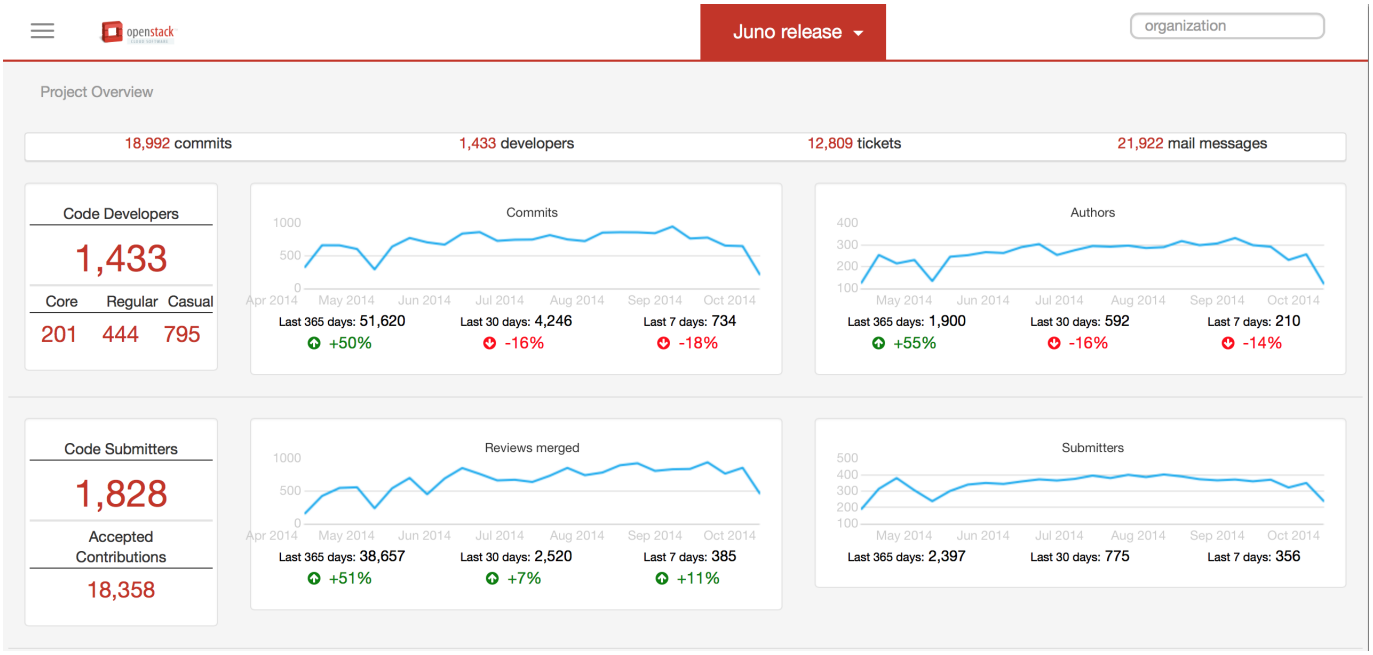

Figura 3.2: Estadisticas de la versión Juno

3.1.1. Cinder

Originariamente, el código de Cinder se albergaba en Nova bajo la etiqueta de 'nova-volume'. A partir de la version Folsom (27 de septiembre de 2012), se creó el componente Cinder. Este componente de OpenStack es el encargado de proporcionar dispositivos de almacenamiento para instancias de OpenStack. Gestiona el almacenamiento de bloques y manipulando volúmenes y 'snapshots'. El Dashboard que proporciona OpenStack permite a los usuarios gestionar sus propias necesidades de almacenamiento.

El almacenamiento de bloques es aconsejable usarlo cuando el rendimiento es delicado, por ejemplo en bases de datos, sistemas de archivo expandibles, etc. Cinder tiene su propia Base de Datos en la cual el estado de cada volúmen puede ser encontrado.

Cinder se compone de:

- Cinder API: Acepta los pedidos y los enruta para que sean procesados.
- Cinder Volume: Lee y escribe sobre la base de datos. Puede interactuar con otros procesos a través de la cola de mensajes ó directamente sobre el almacenamiento.
- Cinder Scheduler: Selecciona el nodo donde se almacenarán por bloques los volúmenes creados.

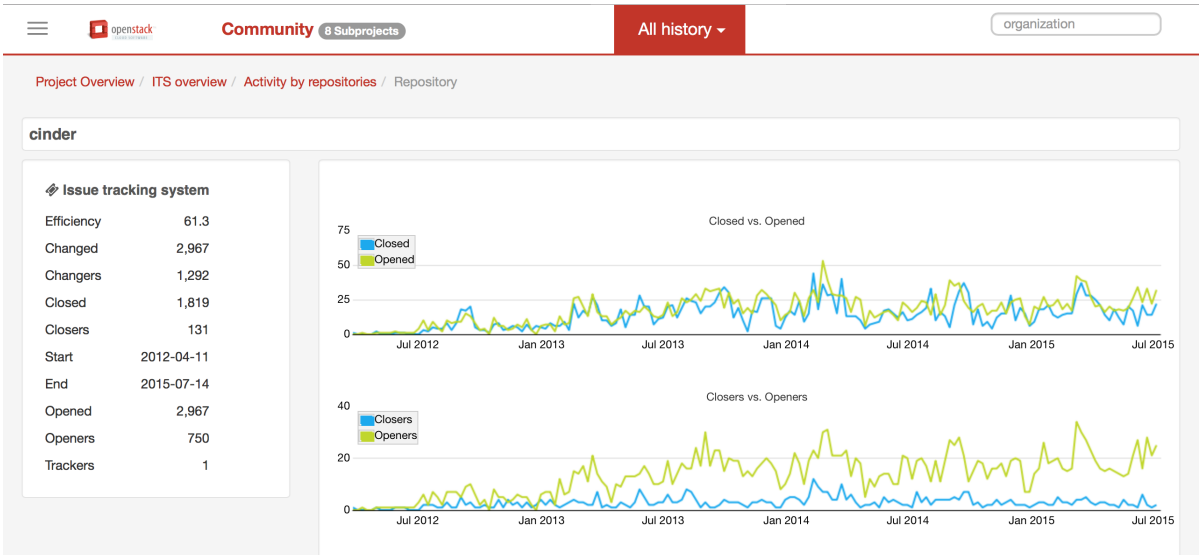

Figura 3.3: Estadísticas de Cinder

Decidimos analizar el repositorio de Cinder ya que este componente de OpenStack posee gran dinamismo, los errores se reportan en el repositorio que estaá habilitado para ello en Launchpad y los desarrolladores comienzan su labor para intentar solucionarlos.

En la figura 3.3 extraída de `http://activity.openstack.org/dash/browser/repository.html?repository=https%3A__bugs.launchpad.net_cinder&ds=its` se muestra diferentes estadísticas sobre la historia de Cinder.

3.2. Launchpad

Es una plataforma, lanzada por Canonical Ltd. en Enero de 2004, de desarrollo colaborativo de software libre. Es un servicio gratuito con una interfaz web desde la cual puedes ver reportes de errores, en el caso en que quisieras reportar, comentar o subir nuevos errores deberías registrarte como usuario. Nos referiremos a estos reportes en secciones posteriores como tickets.

Launchpad consta de varios componentes

- Code: Aloja el código fuente usando el sistema de control de versiones llamado Bazaar.
- Bugs: Permite un sistema de seguimiento de errores para informar sobre errores o bugs en diferentes distribuciones y productos.

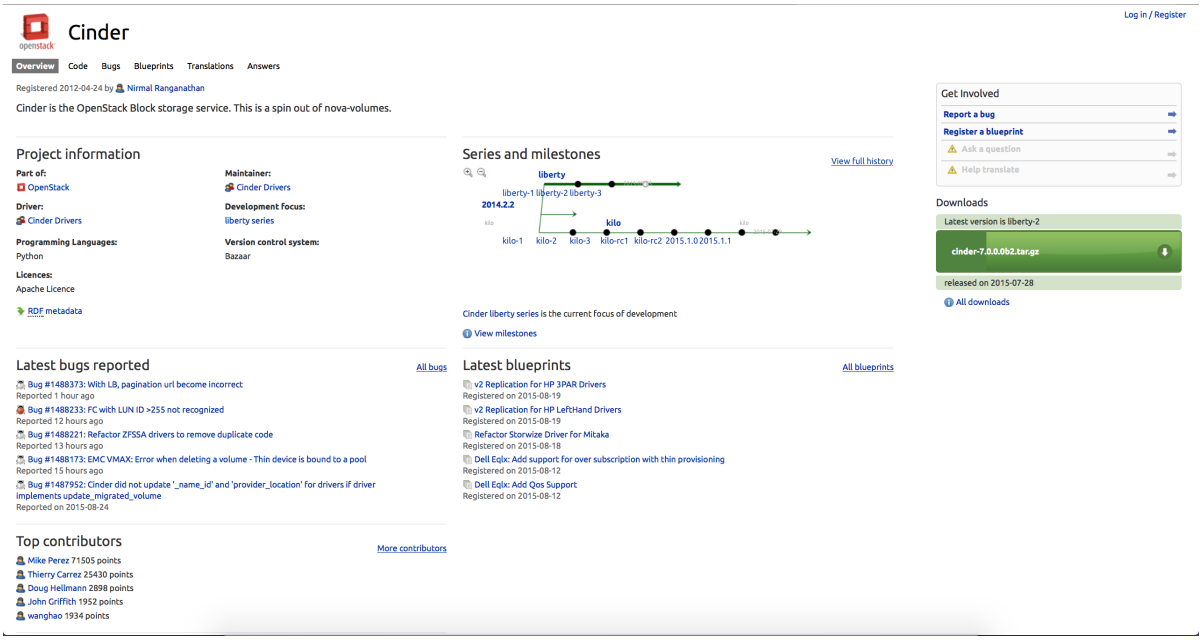

Figura 3.4: Repositorio Cinder

- Blueprints: Sistema de seguimiento usado para nuevas características, funcionalidad ó procesos.
- Translations: Sistema destinado a la traducción de aplicaciones.
- Answers: Foro destinado a la ayuda de la comunidad, en él se resuelven dudas planteadas.

Cada proyecto dentro de OpenStack tiene su propia página en la Launchpad, la figura 3.4 muestra la página que usa Cinder en Launchpad, cuya url es `https://launchpad.net/cinder`

3.3. Git

Git es un sistema de control de versiones distribuido escrito originalmente en C y desarrollado por Linus Torvalds y otraos colaboradores para manejar el kernel de Linux. Distribuido significa que no hay ninguna copia central del repositorio .

Git es un sistema abierto y de código libre diseñado para manejar proyectos de pequeña y gran envergadura con alta eficiencia y velocidad. Se diferencia de otros sistemas de control de versiones en la forma en que modela sus datos. Git modela los datos como si se tratase de un conjunto de instantáneas de un pequeño sistema de archivos. De tal forma que cada

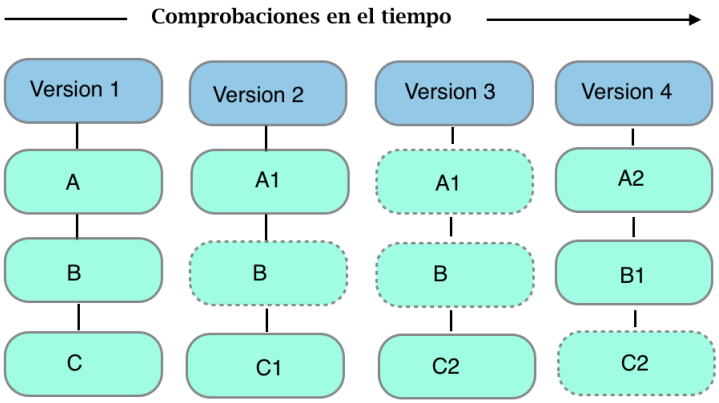

Figura 3.5: Almacenamiento de la información como instantáneas a lo largo del tiempo

vez que confirmas un cambio ó guardas el estado de un proyecto Git, internamente lo que se estaá realizando es una instantánea del aspecto que tiene todos sus archivos en ese momento, guardando una referencia a esa instantánea.

Consigue ser eficiente debido a que no almacena archivos nuevos si no se han sido modificados, sólo almacena un enlace al archivo idéntico de la versión anterior, que ya tiene almacenado ver figura 3.5. La rapidez que demuestra Git en su manejo se debe a que la mayoría de operaciones que realiza son locales, generalmente no necesita información de ningún otro ordenador de la red. De tal forma que para navegar por la historia del proyecto, no necesitamos salir al servidor para obtener la historia y mostrártela, si no que la lee de la base de datos local.

En Git, toda operación es verificada, antes de ser almacenada, a través de un 'checksum' ó suma de comprobación, esa operación es identificada con dicho checksum. Esto quiere decir que no se puede cambiar ningún contenido sin que Git se entere.

Un factor clave para nuestra investigación es que Git no se borra información. Esto se debe a que generalmente, las acciones de Git sólo añaden información a la base de datos. Es realmente difícil conseguir que el sistema haga algo que no se pueda revertir, al menos que los cambios no hayan sido confirmados. Es esto lo que queremos para nuestro estudio ya que nos interesa ver toda la evolución del código en las distintas versiones.

3.4. Gerrit

Cuando muchos desarrolladores trabajan en un mismo proyecto, el código se ve sometido a constantes cambios que pueden provocar que la build se rompa, este es uno de los problemas que tiene Git, ya que su desarrollo es en raíz o trunk, dónde pueden coexistir lineas de desarrollo denominadas como trunk (tronco) y branchs (ramas).

Git no permite un control centralizado, por tanto no permite bloquear cambios que pueden dañar la buid, y se recurre a sistemas como Github ó Gerrit para conseguir estas funcionalidades.

Gerrit es una herramienta gratuita de revision de código colaborativa a través de una Web, para los sistemas de control de versiones de Git, donde el propio Gerrit es un repositorio Git remoto que actúa como intermediario.De esta forma cada vez que se produzca un cambio, el código se subirá a Gerrit y no a Git, y es dentro de Gerrit donde se revisan los cambios propuestos, si cumplen las características y no rompen el build se aprueban, si sucede lo contrario, estos cambios serán rechazados.

Con Gerrit, tendríamos nuestros repositorios Git en local y un repositorio remoto albergado en sitios como Github, dónde haríamos push y pull para recoger o enviar los cambios.

Capítulo 4

Tecnologías Usadas

En esta sección hablaremos sobre las tecnologías que hemos usado en el desarrollo de nuestra herramienta. comentaremos sus caracteísticas he intentaremos dar una visión detallada sobre cada una de ellas.

4.1. JavaScript

JavaScript es un lenguaje de programación, lenguaje de scripting multiplataforma y orientado a objetos, utilizado principalmente en la creación de páginas web dinámicas. Ahora bien, ¿ Qué es una página web dinámica?. Es aquella que incorpora efectos como texto que aparecen y desaparecen, incluye animaciones, acciones activadas a través de botones y ventanas de avisos dirigidas hacia el usuario.

En el lenguaje JavaScript no es necesario compilar los programas, puesto que es un lenguaje de programación interpretado. Es decir, podemos usar cualquier navegador para probar nuestros programas, sin necesidad de procesos intermedios.

Principalmente, el uso de JavaScript se encuentra en el lado del cliente, permitiendo mejoras en la interfaz de usuario. En sus inicios solamente se podían realizar operaciones en el marco de aplicacón del cliente, pero actualmente y gracias a tecnologías como AJAX, puede enviar y recibir información del servidor.

4.1.1. Historia

En los años 90 debido a la baja velocidad de los módems y a la incrusión de formularios en las páginas webs surgió la necesidad de crear un lenguaje de programación que se ejecutase en el navegador del usuario reduciendo así el tiempo de espera en caso de error en el formulario.

La solución a este problema la encontó Brendan Eich, programador que trabajaba en NetScape, adaptando tecnologías existentes como ScripEase al navegador Netscape Navigator 2.0. Este lenguaje fue denominado inicialmente como LiveScript.

Posteriormente, Netscape firmó un acuerdo con Sun Microsystems para desarrollar un nuevo lenguaje de programación, al que llamaron JavaScript justo antes de ser lanzado. Razón meramente comercial ya que la palabra Java estaba de moda en aquella época.

La primera versión de JavaScript fué un éxito, tanto que Microsoft la copió bajo el nombre de Script y la incluyó en su navegador Internet Explorer 3.

En 1997, Netscape dedició estandarizar el lenguaje JavaScript y evitar de esta forma una guerra de tecnologías. ECMA (European Computer Manufacturers Association) creó el comité TC39 con el objetivo de estandarizar un lenguaje de script multiplataforma e independiente de cualquier empresa. El primer estándar se denominó ECMA-262, en el cual se definió por primera vez el lenguaje ECMAScript. Por tanto, JavaScript no es más que la implementación que realizó Netscape del estándar ECMAScript.

4.1.2. Características

Para integrar JavaScript en documentos XHTML solamente es necesario encerrar el código JavaScript entre las etiquetas ¡script¿e incluirlas en cualquier parte del documento. Para que la página XHTML sea válida, se necesita añdir el atributo type a la etiqueta ¡script¿, ya que los valores que incluye type se encuentran estandarizados, text/javascript es el valor correcto para el caso de JavaScript. También tenemos la posibilidad de definir el código JavaScript en un archivo externo y enlazarlo mediante la etiqueta ¡script¿en el XHTML, añadiendo el atributo src, indicando la ruta hacia el archivo, a la etiqueta ¡script¿.

Las principales características que posee JavaScript son:

- Tipado Dinámico: El tipo de la variable se encuentra asociado al valor y no a la variable ya que se trata de un lenguaje scripting. De esta forma podemos tener una variable, x,

ligada a una cadena de caracteres y posteriormente, relegada a un valor float.

- Objetual: JavaScript estaá formado mayoritariamente por objetos, Estos objetos son arrays asociativos, mejorados con la inclusión de prototipos. En tiempo de ejecución podemos crear, cambiar o eliminar las propiedades y sus valores.

- Evaluación en tiempo de ejecución: Existe una función eval que permite evaluar expresiones en tiempo de ejecución

- Funciones de primera clase: A las funciones se le suele conocer como ciudadanos de primera clase; son objetos en si mismos. Por ello, como objetos que son, posee propiedades y métodos. Existe la posibilidad de anidar funciones. Y aparece el término de clausura ligado a la invocación de funciones externas.

- Prototípos: En JavaScript se usan prototipos en lugar de clases para el uso de herencia.

- Funciones como constructores de objetos: Las funciones se pueden comportar como constructores, creando instancias de un prototipo, heredando propiedades y métodos del constructor.

- Entorno de ejecución: Normalmente, JavaScript depende del entorno en el que se ejecute para poder ofrecer objetos y métodos.

- Funciones como métodos: No existe diferencia entre la definición de función y la definición de método. Una función puede ser llamada como un método de un objeto, usando la palabra clave this como variable local a la función que representa al objeto que la invocó.

4.2. jQuery

jQuery es una de las biblioteca de JavaScript más utilizadas, creada el 26 de agosto de 2006 por John Resig, tiene como propósito facilitar la manera de interactuar con los documentos HTML, manipular el árbol DOM , manejar eventos y agregar la interacción con la técnica AJAX a páginas web .

JQuery es software libre y de código abierto, ofrece funcionalidades, .ªtajos", basadas en el lenguaje JavaScript, logrando grandes resultados en un menor tiempo y ocupando menos espacio de código.

Consiste en un único fichero JavaScript con las funcionalidad comunes del DOM, eventos, efectos y AJAX. Su característica principal es que a través de peticiones AJAX y manipulando el árbol DOM, podemos lograr que el contenido de la web cambie sin necesidad de recargar la página.

La forma de interactuar con la página es usando la función jQuery(), normalmente se identifica con el alias $(). Esta función nos permite acceder a los elementos del DOM, recibe como parámetro el nombre de una etiqueta HTML ó una expresión CSS y devuelve todos los elementos del DOM que cumplan la expresión. Las funciones que proporciona la biblioteca pueden ser aplicadas a cualquiera de los nodos que nos devuelva la función.

Para usar jQuery lo único que debemos hacer es añadir la librería de jQuery a nuestra página, la librería puede ser descargada previamente ó incluida desde uno de los servidores de Google que nos la ofrece.

4.3. GitHub.js

Github.js es una librería de JavaScript que nos proporciona un .enVoltorio"de alto nivel entorno a los comandos de git. Cuenta con una API diseñada para manipular los repositorios de GitHub a nivel de archivo.

Si queremos trabajar con esta biblioteca, debemos proporcionarle acceso a la API HTTP de GitHub mediante un protocolo de autenticación. Para ello, se requiere generar un token que será utilizado junto con el mecanismo de autenticación OAuth, mecanismo soportado por esta biblioteca.

4.4. JSON

JavaScript Object Notation ó JSON, es una alternativa a XML en AJAX, que permite representar estructuras de datos en formato ligero para el intercambio de datos. Al requerir menos caracteres para representar la misma información, consume menos ancho de banda.

Una de las grandes ventajas que aporta este formato, a parte de consumir menos ancho de banda, es que parsearlo es mucho mas sencillo. Es decir, escribir un analizador sintáctico requiere menos esfuerzo si usamos un texto JSON, es más, existen funciones como 'eval()' en JavaScript que sirven para dicho propósito, analizar textos JSON.

4.5. NodeJS

Google desarrolló un interprete ultra rápido escrito en C++ para Chrome, pudiendo descargar e incorporar el motor JavaScript V8 a cualquier aplicación que deseemos, ya que se ejecuta en el navegador. Node hace uso de este motor V8 JavaScript dándole otro propósito para ser usado en el servidor.

Node es una plataforma reciente, basada en ejecutar JavaScript en el lado del servidor. Tiene el concepto de agregar módulos a su mismo núcleo. Hay cientos de módulos que se pueden agregar a Node, y si no existe, podríamos desarrollarlo.

En el lado del servidor, la programación es orientada a eventos, estos eventos no se inician de la misma forma que en el lado del cliente ya que no se pulsan botones o se inserta texto, pero a un nivel superior, lo que se está ejecutando son eventos. En el lado del servidor, los eventos se inician con "streams.º flujo de datos entrantes, cuando se realiza una conexión, cuando se está recibiendo datos a través de esa conexión o cuando se dejan de recibir datos.

4.5.1. Características

Node cambia la forma en la que una conexión es realizada con el servidor, de tal forma que en vez de generar un nuevo hilo para cada conexión, lo que se dispara en cada conexión es un evento dentro del proceso del motor de Node. Gracias a esto, Node puede asegurar que soporta decenas de miles de conexiones concurrentes, y nunca se quedará en un punto muerto ya que no permite bloqueos.

Node ha sido diseñado para soportar gran cantidad de tráfico, con una lógica y procesamiento requerido que no necesita ser demasiado grande en la parte del servidor.

Node es perfecto para ofrecer:

- API RESTful: Podemos implementar un servicio Web que facilita una API RESTful,

de la cual toma unos parámetros y los interpreta, devolviendo la respuesta al usuario, normalmente una cantidad pequeña de texto. No requiere una cantidad de lógica,. por tanto pude construirse para dar servicio a multitud conexiones.

Node implementa los protocolos de comunicación más habituales en redes, tales como HTTP,DNS, TLS, SSL, etc.

Algunos ejemplos de empresas que usan Node son LinkedIn, la empresa ha reducido el número de servidores que tenían funcionando al pasar a usar NodeJs. Otras de las empresas que han optado por usar Node son Microsoft, eBay ó la red social Geekli.st.

4.5.2. Tecnologías y frameworks basados en NodeJS

Existen diversos proyectos muy interesantes cuyo funcionamiento esta basado en Node, a continuación los comentaremos:

- Meteor JS: Es un framework destinado a crear aplicaciones Web programado en JavaScript, se ejecuta sobre el motor de NodeJs.
- SocketStream: Es un framework diseñado para soportar aplicaciones de una Web en tiempo real ó Realtime SPAs (Single Padge Apps). Más enfocado hacia el cliente, permite usar plantillas y módulos en el lado del cliente.
- Yeoman: Es una herramienta que simplifica la tarea de crear proyectos, ofrece multitud de utilidades basadas en librerías y frameworks habituales como Bootstrap, BackboneJs ...

4.6. HTML5

Es la quinta versión de del lenguaje HTML, lenguaje utilizado para la creación de páginas web que se encuentra regulado por el Consorcio W3C.

En las webs actuales que usan HTML5, se pueden encontrar etiquetas y atributos nuevos, así como nuevas funcionalidades como son audio y video. Algunas etiquetas han sido eliminadas y otras como la etiqueta ¡canvas¿ha sido mejorada, permitiendo renderizar elementos 3D en los navegadores Chrome, Firefox, Opera, Safari e Internet Explorer.

La versión 5 de HTML es completamente compatible con la versión anterior, HTML4, de tal manera que las webs que usan HTML4 segurán funcionado en HTML5. Antes de HTML5 se usaba la tecnología Flash para proporcionar al navegador de audio, video, webcams, animaciones vectoriales . . . Ahora con HTML5 no serán necesarios más plugins.

4.6.1. Novedades

- Incorpora etiquetas con codecs para mostrar los contenidos multimedia.
- Incorpora mejoras en los formularios, creando nuevos tipos de datos como (eMail, number, url, datetime . . .)
- Incorpora una nueva funcionalidad Drag & Drop que permite arrastrar objetos como si fuesen imágenes.
- Incorpora etiquetas para manejar grandes conjuntos de datos.
- Añade etiquetas como ¡header¿, ¡footer¿, ¡article¿... para manejar la Web Semántica.
- Incorpora una nueva API de Geolocalización para los dispositivos que la soporten.
- Incorpora una API Srorage que nos da la posibilidad de almacenamiento persistente en local.
- Incorpora una API para la comunicación bidireccional entre páginas, llamada WebSockets.

4.7. CSS3

Es la tercera versión de las hojas de estilo en cascada o CSS, mediante estas hojas podemos definir las reglas y los estilos de representación en diferentes dispositivos capaces de mostrar contenidos web. El manejo de CSS es imprescindible para cualquier desarrollador web, ya que le permite definir y diseñar de manera eficiente la representación de la página. El Objetivo principal de CSS es separar el contenido de la forma.

A partir del año 2005 se comenzó a definir CSS3, y actualmente esta versión nos ofrece una amplia variedad de opciones que cubren las necesidades de cualquier diseñador web actual.

Actualmente, CSS3 aporta más control sobre los elementos de la página, introduce nuevos mecanismos con los cuales dejamos de recurrir a trucos que normalmente complicaban el código de la web.

En la url `http://www.css3.info/preview/` se encuentra detalladas todas las características que nos ofrece CSS3, a continuacón listaremos brevemente algunas de ellas.

4.7.1. Novedades

- Bordes: Aparece la posibilidad de definir colores a los bordes, añadirles a los bordes imágenes, diseñar bordes redondeados ó crear sombras a los elementos de la página.
- Fondos: Aparece la posibilidad de decidir la posición de la imagen de fondo con respecto al borde ó conseguir que un elemento tenga varias imágenes de fondo a la vez.
- Color: Aparecen los colores RGBA y la opacidad.
- Texto: Aparece la posibilidad de aplicar sombras ó distintos efectos en los textos y usar cualquier tipografía en una página web.
- Interfaz: Aparece la posibilidad de cambiar el modelo de caja por defecto ó desplegar el menú según tus necesidades.
- Selectores: Aparecen los selectores por atributos
- Degradado: Aparece la posibilidad de insertar degradados, ya sean lineales, radiales ...
- Animaciones: Aparece la posibilidad de realizar efectos que sólo estaban disponibles con otros tipos de tecnologías.

4.8. Bootstrap

Fué desarrollado inicialmente en el año 2011 por el equipo de ingeniería de Twitter, para resolver las inconsistencias en el desarrollo web. Fomentando así la utilización del mismo framework para minimizar inconsistencias, minimizando de esta manera los fallos y aumentando tanto el flujo de trabajo como la eficacia.

En Agosto del 2011 se lanzó al público como un proyecto Open Source en Github, lo que siguió fue una enorme colaboración por parte de miles de desarrolladores, convirtiéndose así en uno de los proyecto Open Source más activos a nivel mundial. Día tras día, Bootstrap gana notoriedad hasta tal punto que ha llegado a convertirse en uno de los framework CSS má famoso y más utilizado.

Bootstrap es una colección de elementos y funciones web como HTML, CSS y JavaScript que pueden personalizarse, están empaquetadas y son accesibles a través de una herramienta. Nos permite crear interfaces de usuario adaptables a cualquier dispositivo y pantalla.

Actualmente se encuentra en la version 3.3.5, pero desde la version Bootstrap 3 lanzada en Agosto del 2013, se ha mejorado el diseño, personalización y la gestión de errores. Es totalmente compatible con los navegadores Google Chrome, Safari, Mozilla Firefox, Internet Explorer y Opera.

4.8.1. BootStrap 3

Esta versión nos aporta una gran variedad de nuevas características como son:

- Soporte casi completo con HTML5 y CSS3.
- Sistema Grid que permite diseñar la web usando un Grid donde el contenido es plasmado.
- Media Queries que permiten de forma fácil variar la web dependiendo del tamaño del dispositivo.
- Insertar imágenes que pueden adaptan su tamaño.

Capítulo 5

Aplicaciones

La literatura actual acerca de la inserción de errores en código abierto, asienta sus bases en la idea preconcebida de que el error ha sido causado por el commit inmediatamente anterior. A partir de esta asunción, se ha desarrollado numerosos estudios acerca de que factores contribuyen a un mayor aumento de la inserción de errores, se han desarrollado diferentes herramienta que clasifican dichos errores ... [añadir literatura Whitewead Zimmerbeg..].[Hablar sobre las conclusiones obtenidas dependiendo de la comunidad que se estudia, analizando los repositorios disponibles (OpenStack, Bugzilla...) en cada estudio/paper]

Hasta el momento, no hemos encontrado ningún estudio que trate de demostrar lo anterior partiendo de que no todos los commits inmediatamente anteriores podrían ser los causantes del error introducido. Es en este punto en el cual se centra nuestro estudio, aportando a la literatura actual un posible nuevo enfoque. Nuestro objetivo no es otro que comprobar empíricamente en cuantos casos la literatura es cierta y en cuantos casos la literatura podría no ser del todo correcta, centrándonos en la responsabilidad que ejerce el/los commit/s inmediatamente anteriores al error introducido.

Capítulo 6

Herramienta y estudio que la valide

6.1. Herramienta

La herramienta consta de dos partes claramente diferencias, el Back-End y el Front-End, ambas partes se comunican a través un API RESTful que nosotros mismos hemos creado atendiendo a nuestras necesidades, de esta forma obtenemos los recursos que necesitamos en cada momento.

Nos referimos a Front-End como las estructuras ”HTML”, Estilos ÇSS.[ei]nteracciones ”JavaScript.[en]cargadas de transformar todo .[el] diseño”web mediante código que no necesita ser procesado para ejecutarse. Actualmente herramientas como AJAX y Websockets nos permiten comunicarnos con la parte del Back-End.

Se conoce como Back-End al encargado implementar la capa de datos, trabajando con lenguajes y gestores base de datos como ”PHP, ASP, JAVAƶ ”MySQL, Postgres, SQL Server, MongoDB”, respectivamente. Para ello usa frameworks como Django y Ruby on Rails o interpretes como NodeJS, que permiten consultas a base de datos remotos, en los que se realizan envío de formularios, inicios de sesión, registros, etc. y se transmite la información a través del código realizado por el Front-End al lado del cliente.

En nuestra herramienta, el Back-End usa el interprete de NodeJs, actuará como servidor, encargandose de realizar peticiones a otras APIs, concretamente a la API de Launchpad `https://api.launchpad.net/1.0.html` y la API de Gerrit `https://gerrit-review.googlesource.com/Documentation/rest-api.html`. También se encargará de atender las peticiones de su propia API, que llegan desde el cliente, enviando toda la información

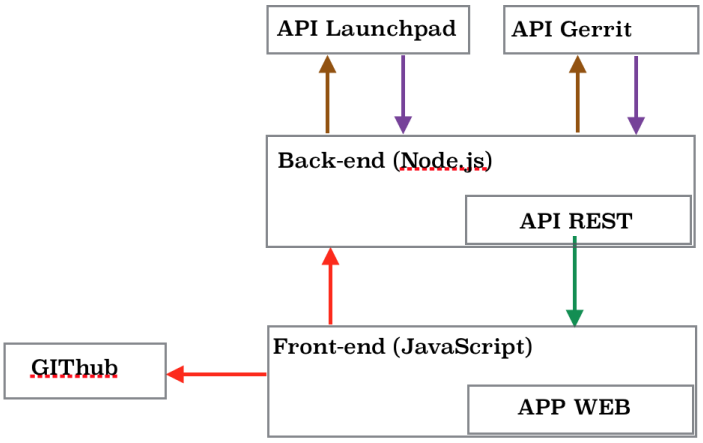

Figura 6.1: Estructura de la herramienta

requerida al Front-End en formato JSON.

Mientras que el frontend tiene varias tecnologías coexistiendo, el estilo y estructura de la página viene definido por Bootstrap, HTML5 y CSS3, mientras que la funcionalidad de la herramienta es trabajo de JavaScript y sus bibliotecas JQuery y Githug.js . Los Request son enviados a través de eventos programados en los botones de los que el usuario dispone, de esta forma, la interfaz de usuario es mas amigable e intuitiva, o por lo menos esa es nuestra idea.

[Hablar del repositorio GIT asociado a la herramienta]

En la figura 6.1 se muestra la estructura de nuestra aplicación web (APP WEB)

6.2. Estudio

En esta seción explicaremos en que se basa nuestro estudio, la descripción será más detallada que en las secciones previas, pero sin llegar a profundizar demasiado, ya que éste no es el objetivo del proyecto. Si queremos saber más acerca del estudio, en los anexos, podemos encontrar un paper en el que se explica el estudio empírico que estamos llevando a cabo. Empezaremos introduciendo algunos de los conceptos y familiarizándonos con palabras que serán nombradas continuamente a lo largo de este proyecto.

El objetivo principal es estudiar el número de casos en los cuales el error ha sido causado por el commit inmediatamente anterior. Entendiendo por commit un cambio realizado en el código

fuente que lleva asociado un identificador único, [hablar sobre repositorios y la evolución de los commits]. A causa de la falta de evidencia empírica que encontramos en la literatura actual acerca del tema que queremos tratar, hemos realizado un estudio observacional que engloba el análisis de 100 tickets extraídos del repositorio de Cinder. Un ticket es el nombre que se da a la evolución escrita sobre un error encontrado en el código, el cual se reporta con un identificador único en un repositorio destinado para ello, en nuestro caso, Launchpad. [hablar sobre Launchpad, alguna captura de pantalla estaría guay].

El estudio consta de dos fases, en la primera de ellas clasificamos los tickets en tres categorías: 'Error', 'No error', ó 'No lo sabemos'. La segunda fase, se centra en el estudio de los tickets que han sido categorizados como 'Error'. Clasificando nuevamente los tickets acorde con de la responsabilidad del commit o commits anteriores en tres nuevos grupos: 'Es responsable', 'No es responsable' y 'No lo podemos saber'.

6.2.1. Primera Fase

En OpenStack los tickets pueden encontrarse en el Launchpad de ese proyecto, en nuestro caso, hemos extraído los 100 tickets de `https://bugs.launchpad.net/cinder`. Los tickets poseen diferentes estados dependiendo de su evolución, desde que se reporta un error hasta que alguien arregla el error, el ticket pasa por diferentes estados: [Listar los diferentes estados].

Nosotros solamente estamos interesados en aquellos tickets cuyo estado sea 'Fix Committed' or 'Fix Released', ya que solamente en esos casos los cambios realizados en el código son visibles. Como ya hemos comentado anteriormente, cada ticket posee un identificador único, será con ese identificador con el cual reconoceremos a ese ticket en el Launchpad. El ticket contiene toda la información relativa al error que se ha reportado [mostrar captura de imagen], pero nosotros solamente usamos cierta información para obtener la clasificación de tickets en esta primera etapa. [Mencionar la información que usamos y el porque la usamos].

Esta primera fase tiene gran relevancia puesto que es aquí donde separamos los tickets que son error de aquellos que no lo son. Esto es muy importante, ya que nuestro estudio empírico se centra en analizar si el commit inmediatamente anterior ha sido el responsable del error. Por tanto, si lo que analizamos no es un ticket en el que se notifica un error, el estudio sería erróneo. Por este motivo, nos interesa detallar que información es la extraída y que nos aporta

esa información para obtener la primera clasificación.

Normalmente, un buen ticket debería tener un titulo claro, una descripción detallada sobre lo que actualmente esta fallando, y una descripción, que iría ligada al commit, en la que se detallase cual ha sido la solución que ha areglado el error. [Captura de pantalla donde aparezcan estos tres campos]. Cuando nos encontramos con tickets que cumplan lo anteriormente mencionado, podemos estar seguros de que el ticket realmente notifica un verdadero error. Suele ser común encontrar que la descripción del ticket y la descripción del commit sea la misma, en esos casos, tendremos que revisar detalladamente el código en el estado antes y después de que el posible error fuera solucionado, para ello será necesario obtener el identificador del commit realiza el parche y el su commit padre. [Hablar un poco sobre como localizar versiones en git moviendose a través de commits].

A continuación detallaremos como obtener la información, como ya hemos dicho, cada ticket posee un identificador único en Launchpad, nos referiremos a él como 'n_ticket'. Con este id, podemos obtener toda la información acerca del error en `https://bugs.launchpad.net/cinder/+bug/n_ticket`. El titulo y el identificador del commit a veces puede encontrarse en la misma página, en un comentario cuyo titulo sea 'Fix merged to cinder (master)'. [Poner captura de pantalla]

OpenStack nos proporciona una interfaz web donde encontramos Gerrit Code Review system, ahí podemos encontrar los cambios propuestos y revisarlos en `https://review.openstack.org/`. Concretamente, podemos encontrar la revisión del código para cada 'n_ticket' que queremos analizar en at `https://review.openstack.org/#/c/n_review`, donde 'n_review' es el identificador de revision único para cada 'n_ticket'. En la revision del código se muestra toda la información acerca del parche que arregló el error.

[Debemos hablar o explicar mas detalladamente el porque necesitamos usar unos criterios o en que casos podemos tener dudas, podria pensar incluso en aïadir un ejemplo con codigo] Necesitamos definir ciertos criterios, los cuales seguiremos en caso de duda. Las siguientes preguntas responden al procedimiento que debemos seguir cuando nos encontremos en esas situaciones.

[Comentar los tipos de ficheros,]

- ¿ El commit sólamente ha modificado ficheros de tests ? No es un error. Uno de nuestros criterios marca que no debemos analizar los ficheros de test, ya que comprendemos que

cada modificatción que se realiza en el código debe llevar asociada su correspondiente comprobación, es decir, código que compruebe que funciona, de esta manera se le da consistencia al programa. Para ello, Cinder tiene unos ficheros bajo el nombre de test, donde realiza estas comprobaciones. Por tanto, entendemos que si algo falla solamente en los ficheros de test, se debe a que el error se encuentra en el código del fichero de test y no en los ficheros del programa.

- ¿ Si el ticket notifica una nueva funcionalidad ? No es un error. No consideraremos las nuevas funcionalidades como errores, bajo nuestro criterio no es un fallo del programa si no una mejora. Algunas personas podrían pensar que la falta de algunas funcionalidades en el programa implica un error, por eso lo notifican en un ticket, pero esa no es es nuestra creencia y por tanto no lo consideraremos como un error. Ademas de las nuevas funcionalidad, todos aquellos tickets que notifiquen optimización del código ó eliminación de código muerto estarían bajo los mismos criterios que las nuevas funcionalidades.

- ¿ El título del ticket describe un comportamiento inesperado ? Es un error. Ocasionalmente, el ticket describe un comportamiento inesperado en funcionalidades que no afectan al comportamiento general del programa. Por este motivo podríamos pensar inicialmente que si no afecta al funcionamiento principal del programa no es un error, pero hemos decidido que lo consideraremos como un error, puesto que se se ha llegado a implementar algo y después no funciona como se espera, es un error, a pesar de que no afecte al funcionamineto general del programa.

- ¿ El título del ticket describe que se requieren actualizaciones ? Es un error. La naturaleza del código es evolucionar, por tanto lo mas normal es que el código necesite ser actualizado, evolucionar en versiones, etc. En esos casos, consideraremos que el ticket está notificando un error ya que si no se actualiza, el programa no funcionará adecuadamente y causará errores.

En ocasiones no seremos capaces de decidir si un ticket es un error o no lo es, por ese motivo tenemos el tercer grupo llamado 'No lo sabemos', en este grupo se encontrarán todos aquellos tickets que son difíciles de clasificar, debido a que no tenemos toda la información necesaria ó que no somos expertos conocedores del código.

6.2.2. Segunda Fase

En la segunda fase, solo nos centraremos en los ticket que hayan sido clasificados dentro del frappe 'Es un Error', ya que de esta forma nos estamos asegurando que todo el análisis posterior es válido. Además, asumimos el error que estamos cometiendo al no tener en cuenta los ticket del grupo ' No lo sabemos' , ya que puede suceder que hayamos clasificado ticket que notificaban errores de verdad en el grupo 'No lo sabemos'. Aún así, decidimos obviar a ese grupo ya que el porcentaje de tickets hallados en él es bajo, entorno al 16 %, por tanto no sería de suma importancia tenerlo en cuenta en los resultados finales y asumimos estar cometiendo un error.

A partir de este momento, la parte más difícil comienza. Nos centraremos en analizar los commits involucrados e identificar quien ha sido el responsable del error. Para ello tenemos que identificar cuales son los ficheros involucrados y dentro de ellos, cuales son las líneas que han sido modificadas, eliminadas ó añadidas.

Cuando hablamos de líneas añadidas, entendemos que son lineas que antes de solucionar el error no se encontraban en el/los fichero/s pero para arreglar el error, han tenido que ser añadidas. Lo mismo sucede con las lineas eliminadas, para solucionar el error han tenido que eliminar lineas del código. Hasta este momento no hay ninguna duda para poder identificar estas líneas, la cosa se complica al identificar líneas que han sido modificadas. En el código pueden realizarse multitud de cambios que impliquen una modificación de código, por eso debemos dejar claro que es lo que nosotros entendemos por código modificado. Para nosotros, el código modificado es aquellas líneas en las que cambia algo, el nombre de una variable, las condiciones del bucle, los argumentos de la función, ... etc. Pero no entendemos que el código haya sido modificado en aquellos casos en los que se borra código de un lado para añadirlo en otro sitio, ó incluso cuando se elimina código para volverlo a escribir idénticamente.

[Hablar sobre los tres grupos de clasificación que se han llevado a cabo, si es responsable, no es responsable, no lo sabemos] Al igual que en la primera fase, en esta también hemos seguido ciertos criterios para identificar si el commit previo es el responsable de introducir el error o no. A continuación aparecen algunos casos en los que podríamos encontrarnos antes de analizar el commit, por tanto el criterio a seguir es el siguiente.

- ¿ El ticket notificaba una actualización ? No hay responsable. Entendemos que la natura-

leza del código es evolucionar, por tanto si se requiere una actualización nadie puede ser el responsable.

- ¿ El commit que arregla en error no se ha llevado a cabo en el repositorio de Cinder ? No podemos identificar al responsable, ya que el parche se ha ejecutado en otro repositorio que no es el de Cinder, por tanto no tenemos acceso al identificador del commit. En ocasiones esto sucede debido a que el error afecta a varios componentes de OpenStack, por tanto el error puede ser arreglado en cualquiera de los repositorios

Si el ticket no se encuentra en ninguna de las situaciones anteriores, tendremos que analizar los commits implicados y el código en detalle. Para analizar el tipo de código usado en el parche y las carpetas que están involucradas, hacemos uso de GIT, sistema de control de versiones distribuido que nos proporciona commandos útiles para el análisis del código.

Para empezar nuestro análisis, lo primero que hemos hecho es clonarnos el repositorio Cinder localmente usando el comando 'git clone'. Después, hemos elegido uno de los tickets y hemos extraído el identificador del commit que arregló el error, y el identificador del commit padre, es decir, el identificador del commit antes de que el error fuese arreglado. También hemos extraído el nombre de los ficheros que han sido involucrados. Usamos el comando 'git checkout id_commit' para cambiar entre los diferentes estados de los ficheros, empezamos en el estado actual, cuando el error ya ha sido arreglado, para obtener el/los identificador/es del commit/s previos al cambio en cada fichero. Para ello, usamos el comando 'git blame file', así obtenemos el commit y auto que ha modificado cada una de las líneas del fichero por última vez, y guardamos la salida obtenida en un fichero de texto. Luego, nos cambiamos al estado del fichero antes de que el error fuese arreglado y realizamos el mismo procedimiento para posteriormente poder comparar los ficheros que hemos guardado.

Una vez que hemos guardado los ficheros tenemos que compararlos para saber en que líneas se encuentran las diferencias entre ambos, y así saber que commits previos son los presuntos responsables del error. Usamos el comando 'diff -B fichero1 fichero2' para mostrar lineaa-linea las diferencias entre los dos ficheros, la optición ˝B' significa que las líneas en blanco se ignorarán. Diff es un algoritmo extensamente explicado en [Ukkonen, 1985, Miller and Myers, 1985, Myers, 1986] el cual nos provides de multitude de source code management systems. Esta herramienta examina ambos ficheros y nos devuelve los cambios que necesita realizar el primer

fichero para coincidir con el segundo fichero, es decir, nos está devolviendo las diferencias entre ellos, mostrándo el número de lineas.

En este momento tenemos una lista de commits con los posibles responsables del error. El siguiente paso es analizar cada uno de los commits y clasificarlos dependiendo de su responsabilidad en uno de estos tres grupos:

1. Es el responsable
2. No es el responsable
3. No sabemos si es responsable

Como es normal, un parche puede tocar varios ficheros y en cada fichero padría estar implicado un commit previo diferente. Por tanto, podemos tener situaciones en las que el responsable del error sea mas de un commit previo.

Una vez que estamos analizando los commits implicados, podríamos encontrarnos en las siguientes situaciones, por ello también dejamos claro cual es el procedimiento a seguir.

- ¿ El commit previo es un commit fork ? No podemos identificar al responsable. [Hablar sobre que es un commit fork] En este caso, no podemos identificar al responsable a menos que nos traslademos de repositorio y sigamos investigando.
- ¿ El commit solo ha modificado comentarios, nombre de las versiones ó a añadido líneas en blanco? No es responsable. Estos son algunos de los ejemplos mas claros sobre cuando un commit no es responsable del error, ya que modificar comentarios o añadir líneas en blanco no alteran el comportamiento del programa.

Capítulo 7

Descripción de la Herramienta

La herramienta nos sirve de gran ayuda para nuestro estudio, nos ahorra tiempo de análisis y nos proporciona la integración de varios procesos en una misma aplicación. La herramienta debe cumplir una serie de requisitos indispensables para que sea de gran utilidad y a su vez para que la experiencia del desarrollador sea agradable.

A continuación se enumeraran los requisitos que debe cumplir nuestra herramienta:

1. Extraer información de un ticket concreto.
2. Obtener el identificador de los tickets de manera aleatoria.
3. Recursos que permitan al desarrollador expresar su opiniïn acerca del análisis.
4. Guardar la información obtenida y analizada sobre el ticket.
5. Visualizar la información almacenada sobre un ticket.
6. Modificar la información almacenada sobre un ticket.
7. Visualizar los resultados.

Estas necesidades se encuentran solventadas en las tres pestañas, 'Analyse', 'Statistics' y 'Modify', de las que dispone la aplicación. Vamos a describir la herramienta desde el lado del cliente, es decir, describiendo cada una de las funcionalidades que nos ofrecen los distintos botones de los que disponemos en interfaz de usuario, iremos poniendo imágenes que muestren visualmente dichas funcionalidades.

La imágen 7.1 muestra el aspecto inicial de la aplicación Web.

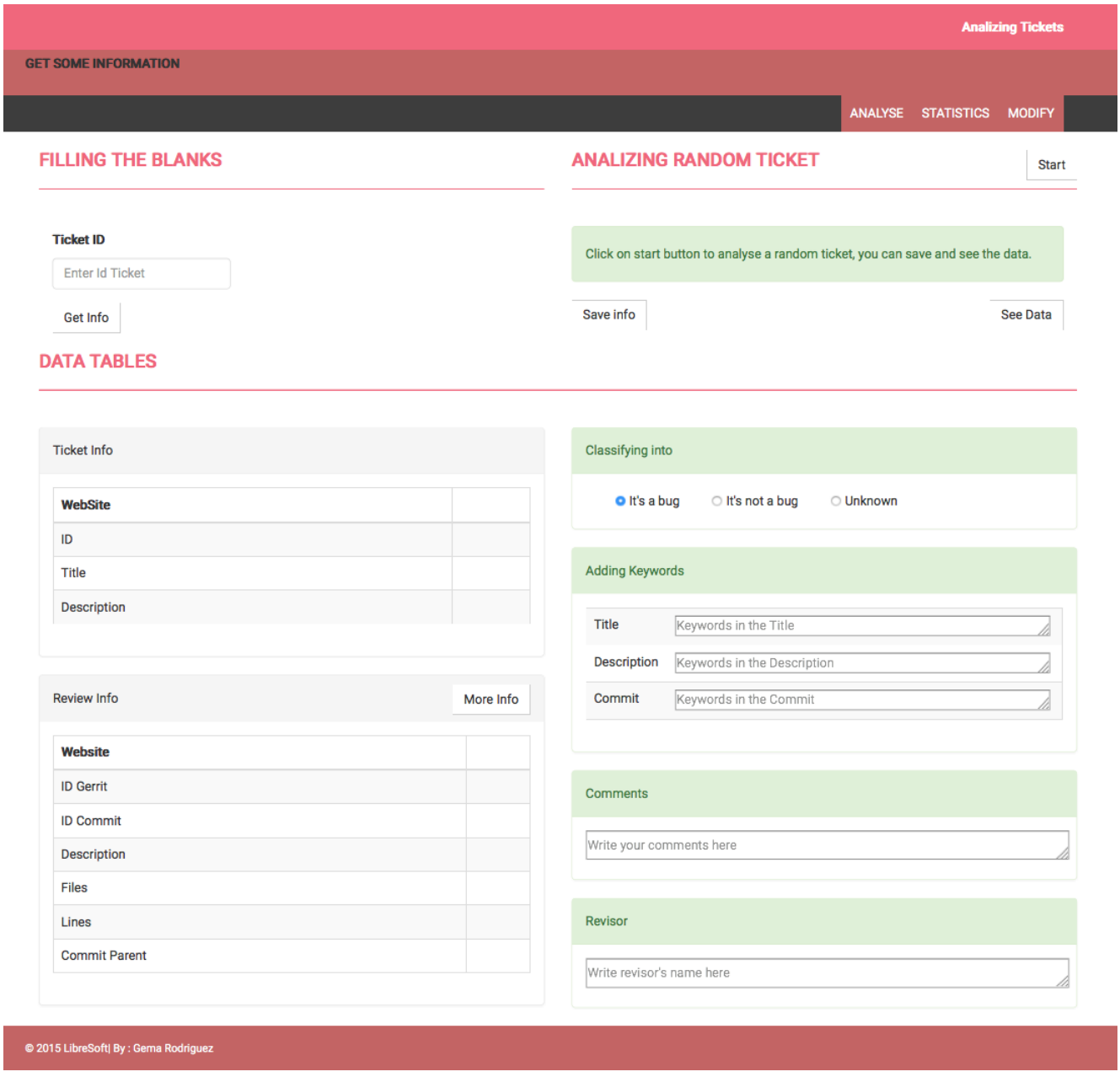

Figura 7.1: Aspecto Web inicial

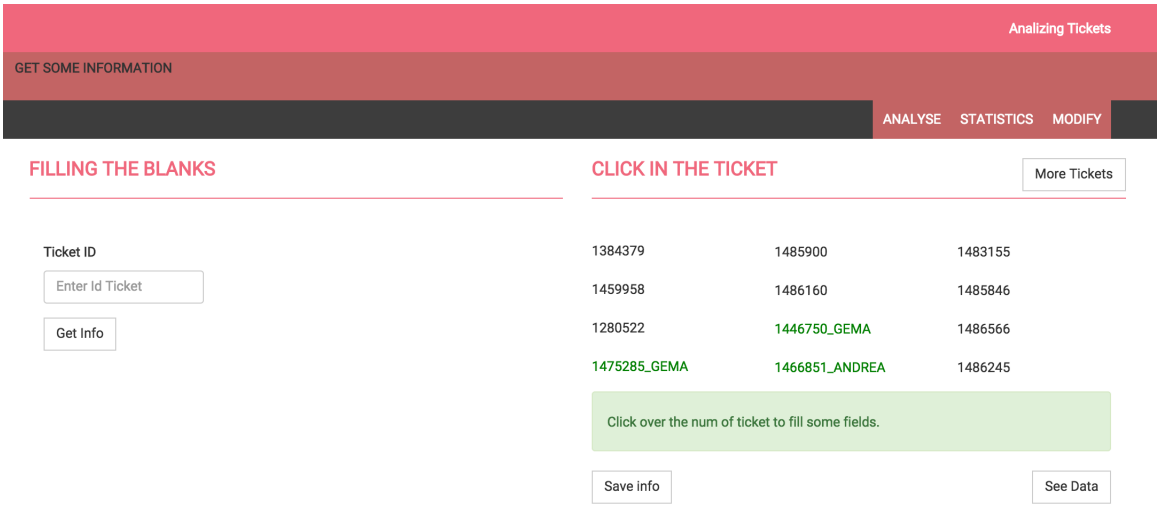

Figura 7.2: Obtención aleatoria y visualización de los tickets en la web

7.1. ANALYSE

En esta sección comentaremos cada uno de los botones y tablas que aparecen en la web al pulsar sobre la pestaña 'ANALYSE'.

La funcionalidad priomordial de la aplicaciïn web que hemos desarrollado se centra en esta pestaña. Nosotros necesitamos analizar una gran cantidad de tickets para poder proporcionar datos concisos y fiables. Para ello, los tickets necesariamente tienen que tener carácter aleatorio, para que nuestra eleción no influya en los resultados finales. Para poder conseguir tickets aleatorios, disponemos del botón 'START'.

Al clicar sobre dicho botón, aparecen tres columnas 'scrollables' con el identificador que los tickets poseen el el Launchpad. Además, se realiza una consulta al repositorio GIT que tenemos asociado para mostrarnos que tickets han sido analizados previamente, coloreando el identificador del ticket en verde y añadiendo el nombre del desarrollador que lo analizó.

De esta forma el desarrollador sabe cuales son los tickets que han sido analizados y decidir si quiere visualizarlos o seguir analizando otros tickets.

Si el desarrollador decide seguir analizando otros tickets, debe pinchar sobre el identificador del ticket que desee analizar, éste pasará a colorearse en un tono ámbar, y se mostrará la información relevante acerca de ese ticket en la tabla 'Ticket Info', informaciïn extraída de Launchpad. Para obtener la información relativa a la revisión del código, el desarrollador debe pulsar el botón 'More Info', y la tabla 'Review Info' se rellenará automáticamente con la

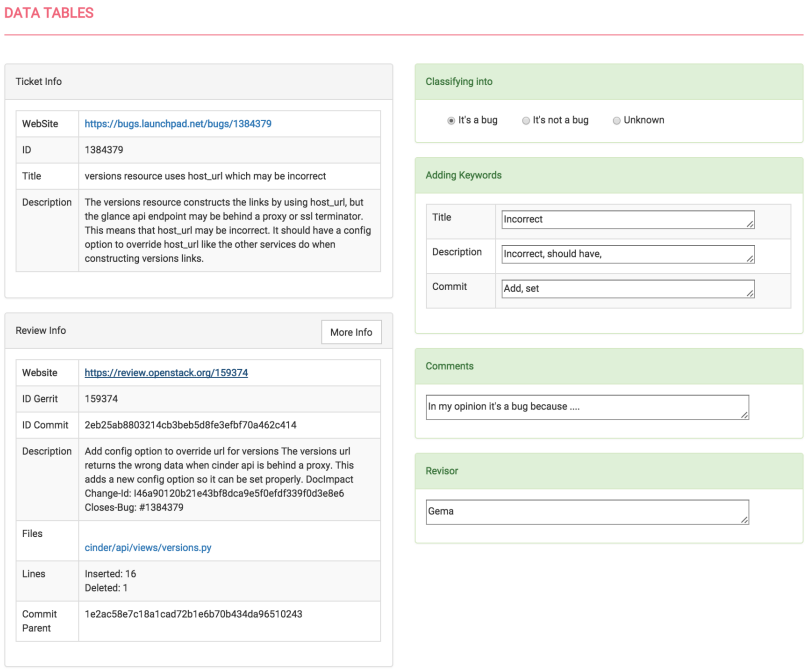

Figura 7.3: Datos Extraídos y comentarios del Desarrollador

información extraída de 'Gerrit'.

Es en este momento es cuando el desarrollador puede categorizar el ticket en uno de los grupos, 'Es un Error', 'No es un Error' ó 'No estamos seguros', añadir las palabras claves y expresar su opinión, para ello, la herramienta dispone de unas 'cajas' y un selector de categoría. En las tablas aparecen enlaces a las páginas tanto de Launchpad como de Gerrit, así como al código que ha sido modificado entre la versión actual y la versión anterior de cada uno de los ficheros implicados. Gracias a esto, el desarrollador tiene a su alcance todo lo que necesita para poder decidir como categoriza el ticket que está analizando.

Las palabras clave o 'keywords' serán útiles en un futuro. Analizaremos su presencia en los tickets, usando algoritmos de machine learning, con el objetivo de conseguir clasificar automáticamente los tickets basándose en las palabras que aparecen en el titulo, la descripción del error y la descripción del parche.

La figura 7.3 muestra como sería el aspecto de las tablas tras el auto-relleno y la opinión del desarrollador.

El último paso para acabar con el análisis del ticket, sería proceder a guardar la información. Para ello se encuentra habilitado el botón 'SAVE', que transforma todos datos a un formato JSON y lo almacena en el repositorio albergado en GIThub. Añadiendo un archivo de

texto, cuyo contenido son los datos en formato JSON y cuyo nombre es 'NombreDelRevisor_NumDelTicket'. De esta forma, sabemos quién es el revisor y el ticket que se ha revisado.

En esta pestaña se añade la posibilidad de analizar un ticket del cual sabemos su identificador, para ello deberemos rellenar el campo etiquetado como 'Ticket ID' y pulsar el botón 'GET INFO' ver figura 7.2.

Internamente, lo que se estaá ejecutando tras pulsar sobre el identificador del ticket o en el botón 'GET INFO', es un evento programado. Este evento hace su primera petición request a nuestra API, obteniendo los campos agrupados en la tabla 'Ticket INFO'. La segunda petición request a nuestra a API se realiza cuando pulsamos el botón 'MORE INFO', rellenando así los campos correspondientes a la información que obtenemos del code review (GERRIT).

La otra opción que tiene el desarrollador es pulsar sobre uno de los tickets coloreados en verde, ticket que ya ha sido analizado, para poder visualizar la información que fue guardada. Para ello, tras pulsar en el ticket, debemos pulsar el botón 'SEE DATA', y aparecerá en la parte derecha de la web una tabla con toda la información almacenada en el ticket.

7.2. STATISTICS

Al pulsar sobre esta pestaña, aparecen unos selectores que muestran el nombre de los revisores que han contribuido en el análisis de la primera fase. Al pulsar sobre cada uno de los selectores, se rellena una tabla de datos en la cual se muestran los resultados de la primera etapa de clasificación, es decir, las estadísticas de clasificación de los tickets para ese revisor en la primera fase.

En la tabla se puede visualizar el número de tickets que componen cada uno de los tres grupos de clasificación, 'Es un error, 'No es un error', ' Unknown', así como el porcentaje que se ha obtenido en cada grupo.

Además, disponemos de la posibilidad de dibujar un gáfico de barras en el que se muestra visualmente el porcentaje de tickets para cada grupo. Como se muestra en la figura 7.5.

Para conseguir esto, lo único que debe hacer el usuario es pulsar sobre el botón 'DRAW'. A la derecha del gáfico disponemos de un menú no desplegado que nos permite descargarnos la imagen del gáfico con diferentes extensiones.

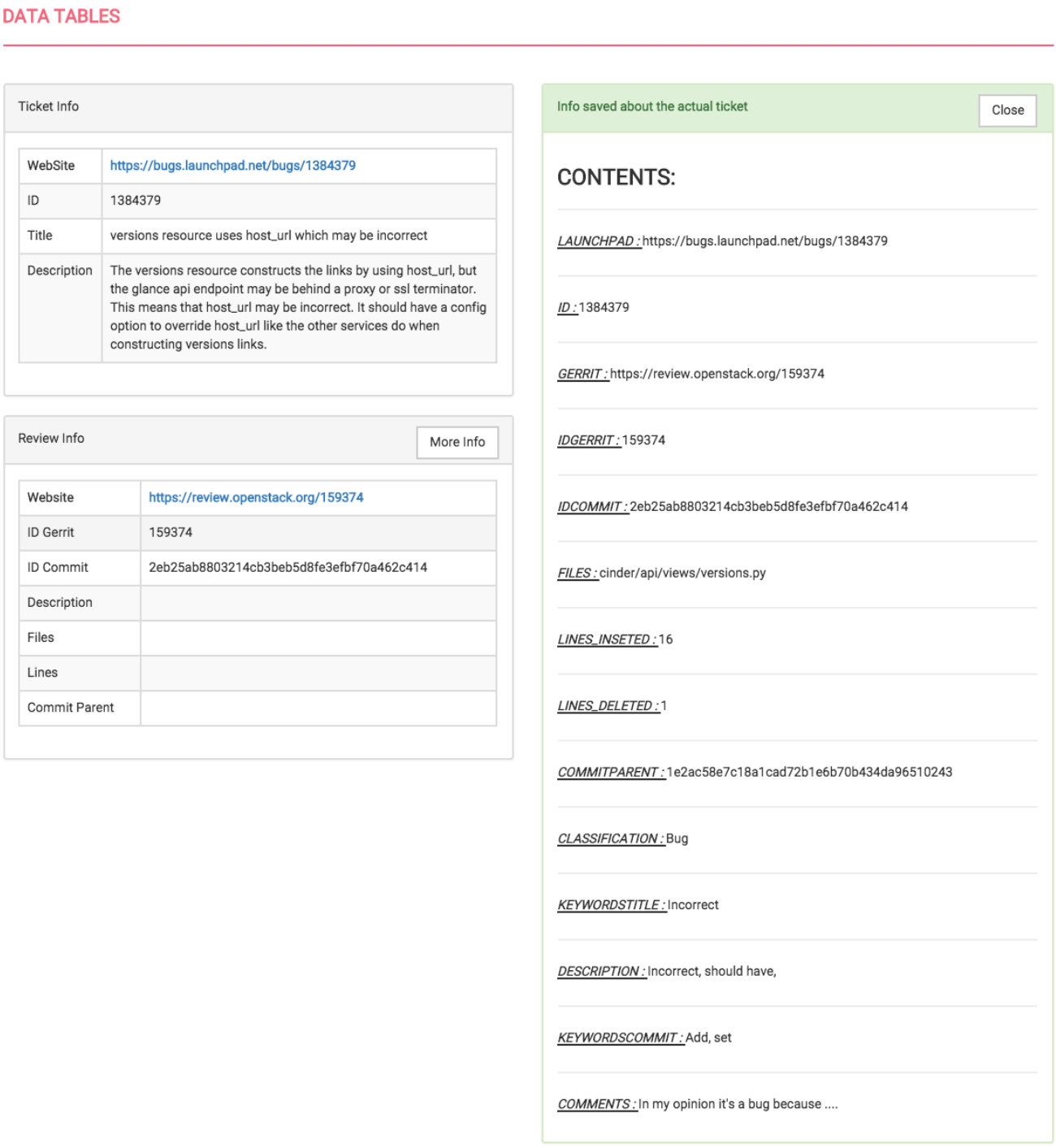

Figura 7.4: Visualización de los datos tras pulsar el botón 'See Data'

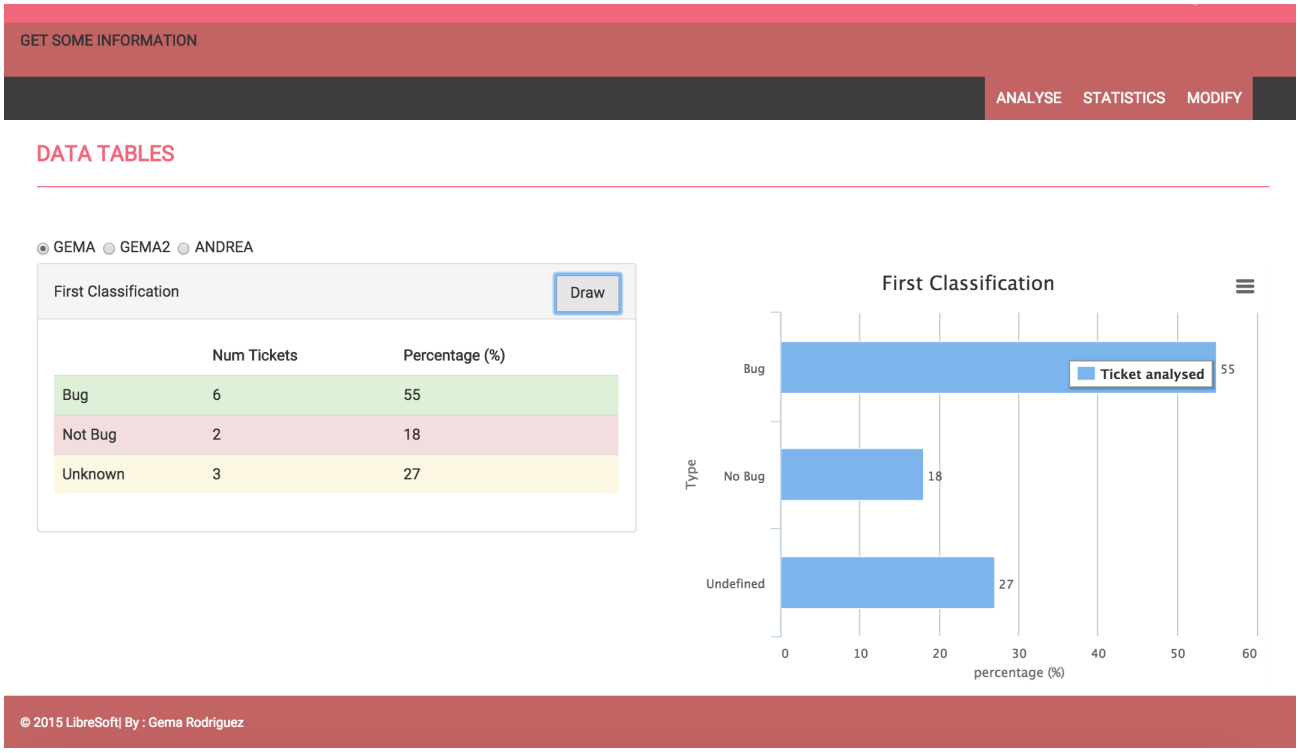

Figura 7.5: Visualización la Web tras pulsar la pestaña 'ANALYSE'

7.3. MODIFY

Tras pulsar sobre la pestaña 'MODIFY', se nos proporciona en la parte izquierda un listado con todos los tickets que han sido analizados y se encuentran guardados en el repositorio GIT asociado a la web, y en la parte derecha un formulario que se autocompletará tras clicar sobre un ticket.

Esta funcionalidad estaá diseñada con el fin de proporcionar al desarrollador un lugar en el que pueda modificar aquellos tickets que fueron guardados previamente. Ya sea porque cometió un fallo, porque quiere añadir mas información o simplemente porque ha cambiado de opinión tras volverlo a revisar.

En el formulario todos los campos que se rellenan pueden ser modificados a excepción del nombre del desarrollador, ya que para los resultados necesitamos saber quien ha sido el desarrollador.

El aspecto que posee la web tras haber pinchado en la pestaña 'MODIFY' aparece en la figura 7.6

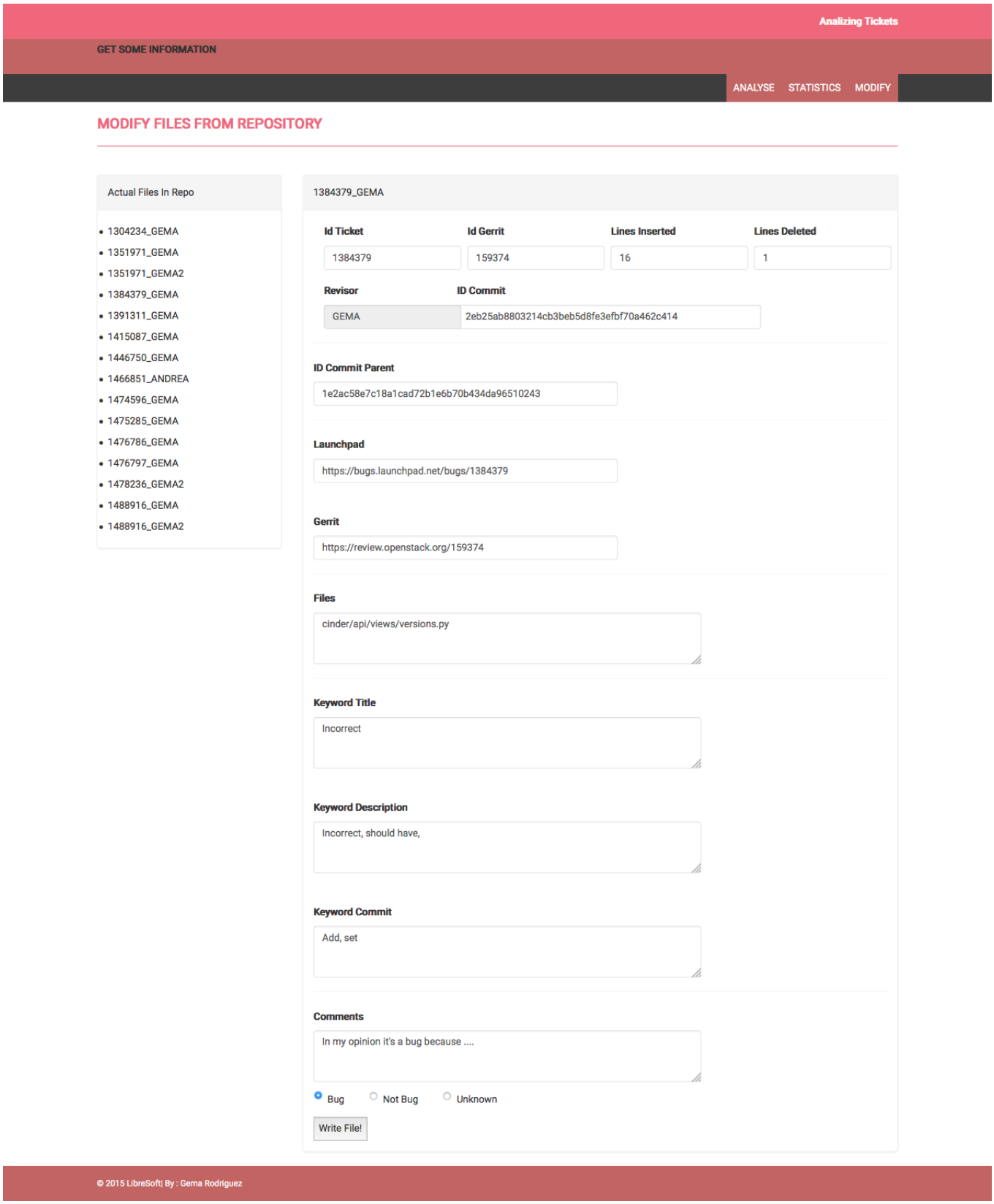

Figura 7.6: Visualización la Web tras pulsar la pestaña 'MODIFY'

Capítulo 8

Gráficas

Capítulo 9

Resultados

Capítulo 10

Conclusiones

10.1. Consecución de objetivos

Esta sección es la sección espejo de las dos primeras del capítulo de objetivos, donde se planteaba el objetivo general y se elaboraban los específicos.

Es aquí donde hay que debatir qué se ha conseguido y qué no. Cuando algo no se ha conseguido, se ha de justificar, en términos de qué problemas se han encontrado y qué medidas se han tomado para mitigar esos problemas.

10.2. Aplicación de lo aprendido

Aquí viene lo que has aprendido durante el Grado/Máster y que has aplicado en el TFG/TFM. Una buena idea es poner las asignaturas más relacionadas y comentar en un párrafo los conocimientos y habilidades puestos en práctica.

1. a
2. b

10.3. Lecciones aprendidas

Aquí viene lo que has aprendido en el Trabajo Fin de Grado/Máster.

1. a

2. b

10.4. Trabajos futuros

Ningún software se termina, así que aquí vienen ideas y funcionalidades que estaría bien tener implementadas en el futuro.

Es un apartado que sirve para dar ideas de cara a futuros TFGs/TFMs.

10.5. Valoración personal

Finalmente (y de manera opcional), hay gente que se anima a dar su punto de vista sobre el proyecto, lo que ha aprendido, lo que le gustaría haber aprendido, las tecnologías utilizadas y demás.

Apéndice A

Manual de usuario