



MASTER EN INGENIERIA DE TELECOMUNICACIONES

Curso Académico 2015/2016

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

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

La defensa del presente Proyecto Fin de Máster se realizó el día de
de 2016, siendo calificada por el siguiente tribunal:

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Fuenlabrada, a de de 2016

*Dedicado a
mi abuelo, Ismael.*

Agradecimientos

En primer lugar, agradecer a mi familia todo el apoyo recibido, así como toda la confianza depositada en mi desde que decidí adentrarme en el mundo de las Telecomunicaciones. Quiero dejar constancia de lo agradecida que me siento con mis padres, ellos son los que me han sufrido en las épocas de estrés, y a pesar de ello siempre me han regalado palabras de ánimo y aún hoy, me siguen brindando grandes consejos. ¡Gracias Padres!. También quiero nombrar el agradecimiento que siento hacia mis abuelos, ellos han celebrado tanto como yo cada uno de mis aprobados, y han sufrido conmigo en los suspensos.

Me gustaría agradecer también a todas las personas que comparten mi día a día de un modo u otro, aportándome buenos momentos y haciéndome feliz, gracias a ellos el estudiar, trabajar o incluso el madrugar se hace mucho mejor.

Llegados a este punto, y mirando todo lo que las telecomunicaciones me ha ofrecido, si volviese a elegir una carrera, no dudo que sería la misma. En ella he encontrado grandes amigos, experiencias inolvidables, y como no, la importancia del esfuerzo constante. Además, he tenido la posibilidad de conocer a esos grandes profesores, profesores de vocación que hacen que este mundo del que ahora formo parte sea algo fascinante.

Por último, me gustaría agradecer a mi tutor, Jesús, toda la ayuda recibida durante el proyecto y la confianza que ha depositado en mi para que logre llevarlo a cabo.

«El crecimiento es un proceso de prueba y error: es una experimentación. Los experimentos fallidos forman parte del proceso en igual medida que el experimento que funciona bien» - Benjamin Franklin.

Resumen

Este proyecto se inscribe en el marco de la ingeniería del software, concretamente en el área dedicada a encontrar cómo y cuándo se introdujo un error no intencionado en el código fuente, en inglés, conocido como bug-seeding. Existen herramientas que permiten detectar si el código alberga errores o si es propenso a ello, analizando semánticamente y sintácticamente su estructura. A pesar de ello, el software se ve sometido a cambios que provocan que deje de funcionar, ocasionados por errores indetectables en las pruebas de test. Y es ahí dónde se contextualiza nuestro proyecto, aportando información sobre de qué manera y en qué momento el error fue introducido al código.

Partiendo de una teoría actualmente asumida por los desarrolladores que trabajan en este área, hemos llevado a cabo el estudio de investigación presentado en esta memoria. Dicho estudio aporta datos empíricos que muestran la validez del teorema aplicada al estudio de un proyecto en concreto. Nuestra intención no es otra que, ayudar a entender mejor el proceso involucrado en el bug-seeding, desde que aparece un error hasta que se soluciona, identificando el momento en el que dicho error fue introducido, poniendo atención a la importancia que ejerce el cambio previo en la insercción del error.

Además, en la memoria se presenta una herramienta desarrollada durante la investigación, cuyo objetivo es mecanizar la fase de análisis lo máximo posible, ayudándonos a ahorrar tiempo en la extracción de datos y mostrándo de manera atractiva la información que ayuda a los desarrolladores a tomar sus decisiones. Cabe mencionar, que la herramienta se encuentra validada por nuestro análisis, y ha sido diseñada para trabajar con el proyecto que estábamos investigando. Por tanto, actualmente no da soporte para analizar otros proyectos.

Summary

This project is focused on a specific area of software engineering, the study of how and when bug was inserted into source code, also known as bug seeding. We can find many tools that help us to prevent errors in the code analyzing its syntactical and semantical structure. But, sometimes the errors are undetectable and pass the tests, causing the system to fail. This project contribute to understand the procedure implicated in bug seeding, providing information about how and when the bug was introduced into the source code, also who previous change caused the bug.

We have conducted an observational study that englobes many tickets extracted from a free and open-source cloud computing software platform. The experiment was focused as a first approach to prove that the currently premise established does not hold for all cases. Understanding the importance of the previous commit in the bug-seeding.

Furthermore, in this project we showed a tool developed, which automates one part of the methodology used to distinguish between bug reports and other issues. The tool minimizes the time required to analyzing the data extracted, and shows us intuitively all data in the same interface.

Índice general

1. Introducción	1
1.1. El problema	2
1.2. La solución	3
1.3. Los objetivos	4
2. Estado del arte y contexto	7
2.1. Precedentes	7
2.2. Proyecto OpenStack	9
2.3. Herramientas utilizadas	13
2.3.1. Launchpad	16
2.3.2. Git	17
2.3.3. Gerrit	18
3. Tecnologías Usadas	21
3.1. Tecnologías usadas en el Navegador	21
3.1.1. HTML5	21
3.1.2. CSS3	22
3.1.3. Bootstrap	23
3.1.4. JavaScript	24
3.1.5. jQuery	27
3.1.6. GitHub.js	27
3.2. Tecnología usada en el servidor	28
3.2.1. NodeJS	28
3.3. Tecnologías de intercambio de datos	29

3.3.1.	JSON	29
3.3.2.	REST	30
3.3.3.	OAuth	31
4.	Estudio Realizado	33
4.1.	El estudio	33
4.2.	Primera fase ó filtrado	34
4.2.1.	Búsqueda y extracción	35
4.2.2.	Metodología	36
4.2.3.	Clasificación	38
4.3.	Segunda fase ó causa del error	40
4.3.1.	Análisis del código	40
4.3.2.	Metodología	43
4.3.3.	Clasificación	44
5.	Descripción de la Herramienta	47
5.1.	Arquitectura	47
5.2.	Requisitos	49
5.3.	API	50
5.4.	Autenticación	52
5.5.	Funcionamiento	52
5.5.1.	Analyze	53
5.5.2.	Statistics	57
5.5.3.	Modify	57
6.	Resultados	61
6.1.	Resultados del estudio	61
6.1.1.	Resultados primera fase	61
6.1.2.	Resultados segunda fase	63
6.2.	Resultados generales	67
7.	Conclusiones	69
7.1.	Consecución de objetivos	69

<i>ÍNDICE GENERAL</i>	XI
7.2. Aplicación de lo aprendido	70
7.3. Lecciones aprendidas	71
7.4. Trabajos futuros	72
7.5. Valoración personal	73
A. Artículo realizado	75
Bibliografía	85

Índice de figuras

2.1. Relaciones entre los servicios de OpenStack	11
2.2. Estadísticas de la versión Juno	12
2.3. Estadísticas de Cinder	14
2.4. flujo de trabajo en OpenStack	15
2.5. Aspecto de los tickets en Cinder	15
2.6. Aspecto de Gerrit para un ticket	16
2.7. Repositorio Cinder	17
2.8. Almacenamiento de la información como instantáneas a lo largo del tiempo . .	18
4.1. Ticket claramente clasificado como 'Es un error'	37
4.2. Ejemplo de cuando el cambio insertado en el commit previo,31f208423, causó el error.La versión del código mas reciente se encuentra a la izquierda (1) . . .	42
4.3. Ejemplo donde el commit previo, 0dc91bed, no causó el error. La versión del código mas reciente se encuentra a la izquierda (1)	43
4.4. Salida que muestra el comando diff entre dos ficheros	44
5.1. Estructura de la herramienta	48
5.2. Aspecto Web inicial	54
5.3. Obtención aleatoria y visualización de los tickets en la web	55
5.4. Datos Extraídos y comentarios del Desarrollador	56
5.5. Visualización de los datos tras pulsar el botón 'See Data'	58
5.6. Visualización de la Web tras pulsar la pestaña 'Analyze'	59
5.7. Visualización la Web tras pulsar la pestaña 'Modify'	60
6.1. Resultados de la primera fase	62

6.2. Resultados de la segunda fase	65
6.3. Grado de responsabilidad de cada uno de los commits en un ticket	67

Capítulo 1

Introducción

El software libre y de código abierto, *FLOSS*, nos ofrece la gran ventaja de mejorar continuamente su diseño; los desarrolladores estudian, modifican y mejoran el software gracias a la disponibilidad del código fuente. La evolución de la ingeniería del software libre, ha favorecido la aparición de repositorios de versiones, donde se puede observar la evolución a la que se somete el código, permitiendo a los desarrolladores investigar como se lleva a cabo todo el proceso de desarrollo.

El desarrollo en sistemas *FLOSS* puede llegar a ser algo complejo ya que en este tipo de sistemas se permite a todo el mundo contribuir, desarrolladores noveles y expertos cooperan para mejorar el software. Por tanto, es imposible tratar de prevenir las necesidades de todos ellos, así como los errores que aparecerán. Cientos de desarrolladores tendrán que permanecer atentos en las actividades que precisan mantenimiento; añadiendo nuevas características, mejorando las existentes o corrigiendo errores. Por este último motivo, los investigadores encontraron la necesidad de entender que cambios provocaron un fallo en el sistema para poder tratar de prevenirlos en el futuro. Así, surgieron los conceptos de bug-fixing, lo que en español se entiende como el proceso encargado de corregir un error, o bug-introducing/bug-seeding que se traduce como el momento en el que se insertó un error en el código fuente.

En los últimos años, se ha puesto mucho esfuerzo en entender el bug-seeding, investigando a partir de un error arreglado, quien lo causó. Además, una vez encontrado que cambio previo causó el error, se pueden sacar métricas de él que nos ayuden en este entendimiento. Así pues, se han desarrollado multitud de algoritmos y herramientas que ayudan a prevenir la insercción de errores en el código fuente. Estas herramientas y estudios son muy útiles para evitar insertar

errores en el software, pero todas ellas asimilan una misma idea que parece razonable pero que no ha sido probada empíricamente.

Nuestra aportación con este trabajo de investigación, se centra en aportar datos empíricos sobre la asunción que todos los estudios previos presentan. Nos centraremos en entender cómo y cuándo un error fue introducido en el código a través de un cambio, y analizando ese cambio podremos saber si causó el error o no.

Esta memoria se estructura de la siguiente manera; en la sección 1 se presenta una breve introducción sobre el tema a tratar en esta memoria, presentando la carencia encontrada actualmente en la literatura y la solución propuesta en las subsecciones 1.1 y 1.2. La sección 2, trata de introducir al lector en el estado de la técnica actual, subsección 2.1, y contextualizar el proyecto en las subsecciones 2.2 y 2.3. En la sección 3, se detallan las tecnologías usadas en el desarrollo de nuestra herramienta. En la sección 4 se detalla el estudio que hemos llevado a cabo. Dicho estudio consta de dos fases, explicadas en las secciones 4.2 y 4.3, durante las que se extraen y se analizan datos, obteniendo una clasificación final en cada una de ellas. En la realización de estas fases, nos dimos cuenta que ciertas partes de nuestra metodología se podían mecanizar, por ese motivo y para ahorrar tiempo, se desarrolló la herramienta descrita en el capítulo 5. Los resultados obtenidos tras finalizar el estudio se muestran en el capítulo 6. Las conclusiones aparecen en el capítulo 7, donde la primera sección 7.1, habla sobre los objetivos descritos en la sección 1.3 que han sido cumplidos. En la sección 7.4 aparecen algunas de las líneas de trabajo futuro. Además, para finalizar, en el anexo A se adjunta un documento que detalla todo el trabajo de investigación realizado.

En <http://gemarodri.GitHub.io/Presentacion/> podemos encontrar una web que reúne todos los recursos disponibles en el TFM como son, esta memoria, la herramienta desarrollada y su código fuente. Además, hemos añadido una demo en la que se presenta el funcionamiento de la herramienta.

1.1. El problema

Como se ha mencionado anteriormente, la falta de evidencia empírica que pruebe la actual idea concebida en la literatura actual sobre el bug-seeding es el problema del que parte esta memoria. Esta idea asume que el cambio anterior en las líneas que han sido arregladas pos-

teriormente causó la aparición del error en el código fuente. Esta idea tiene mucho sentido y señala al commit¹ anterior como el responsable de causar el error pero sin embargo no ha habido ningún estudio que demuestre empíricamente que es cierta, y creemos que hay otros cambios y otras circunstancias en las que el commit previo no causó ningún error.

Fue Zimmermann en uno de sus artículos [ZZ05], quien apuntó que el cambio anterior es el causante del posterior arreglo (*'This earlier change is the one that caused the later fixed'*). Otros autores como James Whitehead, ni siquiera indican en sus artículos que el commit anterior es el responsable, si no que ya parten de esa idea, lo tienen asumido. Al igual que sucede con J. Whitehead, esta idea preconcebida se encuentra tan generalizada que la mayoría de estudios de investigación relacionados con el bug-seeding parten de ella. Existen artículos recientes como [FWB⁺15], donde se presentan herramientas que detectan automáticamente los cambios que pueden ser futuros errores, pero estas herramientas también parten de la misma idea (*'We assume that a change/commit is buggy if its modifications has been later altered by a bug-fix commit'*). Hasta el momento no hemos descubierto ningún estudio o investigación empírica que trate de contribuir a este pensamiento generalizado aportando datos o porcentajes que muestren en cuantas ocasiones la literatura actual se cumple, y en cuantas ocasiones esta idea no es acorde con los datos.

1.2. La solución

Debido a esta carencia en la literatura actual, decidimos desarrollar un estudio de investigación, enfocado en descubrir y analizar el momento en el que un cambio en el código fuente insertó un error, y a partir de ese momento, poder identificar al/los causantes de dicho error. Consiguiendo finalmente una evidencia empírica sobre el número de errores introducidos por el commit anterior, y aquellos errores introducidos en los que el commit anterior no causó el posterior arreglo.

Para realizar dicho estudio, nos centramos en analizar el proyecto OpenStack, concretamente uno de sus componentes, Cinder. OpenStack se ayuda de herramientas como Launchpad, Gerrit y Git, para funcionar. Launchpad es un sistema de seguimientos de errores donde los

¹acción de comprometer, cambio que indica que cierto conjunto de operaciones realizadas sobre una base de datos, ha culminado de efectuarse correctamente.

desarrolladores dejan constancia de las especificaciones que requiere el sistema, usando lo que se conoce como *blueprint*. A través de los *blueprint* se describe una idea, ya sea una característica o un proceso, que llevará asignados metadatos (*identificador, proyecto al que pertenece o importancia*) que permiten que sea rastreado. Nosotros estamos interesados en aquellos *blueprint* que se han etiquetado en el sistema como errores. Una vez que dichos errores han sido descritos en Launchpad, y mediante su identificador único, la evolución del error se puede seguir en Gerrit, que no es más que un sistema de revisión de código donde se discute los posibles arreglos para que el software vuelva a funcionar. Cuando los arreglos propuestos son aceptados, Git se encarga de introducirlos al código fuente a través un commit, arreglando así el error.

Gracias a esta conexión en las herramientas, podemos rastrear un error y encontrar el momento en el que fue introducido, puesto que tenemos a nuestra disposición todo el historial de cambios que se han realizado en el software. Este proceso no es único de OpenStack, si no que es relativamente típico en otros proyectos que usan las mismas herramientas, como Mahara².

Una vez que sabemos como rastrear un error obteniendo el momento en el que se introdujo, podemos averiguar quien causó el error, y si el commit previo fue quien provocó dicho fallo. Todo este proceso se realizó manualmente hasta obtener una metodología adecuada, muchos de los pasos descritos en la metodología se podían automatizar ya que se repetían constantemente y es por eso que decidimos diseñar la aplicación web, para facilitarnos el trabajo reduciendo aproximadamente de 5 a 10 minutos el tiempo de análisis de cada ticket.

1.3. Los objetivos

Los objetivos propuestos abarcan dos ámbitos claramente diferenciados. Por un lado, el estudio de investigación, y por otro lado, la herramienta desarrollada.

La obtención de datos fiables a través del análisis del proyecto, ayudará a generar diversas gráficas, aportando de forma visual los datos obtenidos. La tendencia que pueden mostrar las gráficas y nuestras expectativas al respecto, marcarán un punto de inflexión en la investigación, viéndonos en la necesidad de pararnos a pensar y debatir sobre el futuro de la investigación.

Por tanto, los objetivos a cumplir en el proyecto de investigación son:

- Objetivo 1: Entender el proceso requerido para rastrear un error. Es decir, partiendo del

²<https://mahara.org>

error, saber cómo y dónde buscamos para encontrar el momento en el que se introdujo el error.

- Objetivo 2: Escribir una metodología adecuada para identificar al responsable, de manera que que nuestra investigación sea reproducible.
- Objetivo 3: Obtener porcentajes fiables que nos permitan conocer las veces que la teoría actual se cumple en nuestro caso de estudio.
- Objetivo 4: Obtener porcentajes fiables que nos permitan conocer las veces que la teoría actual no se cumple en nuestro caso de estudio.
- Objetivo 5: Aportar gráficas en las que se relacionen los datos extraídos con la teoría actual.

Por otro lado, y tras ver que nuestra metodología no era solamente reproducible si no que podía ser automatizada, planteamos una serie de objetivos a cumplir, que se encuentran listados a continuación. Seguiremos con la numeración anterior para luego poder referenciarlos en las conclusiones.

- Objetivo 6: La metodología consta de dos fases, por tanto trataremos de automatizar lo máximo posible ambas fases.
- Objetivo 7: La herramienta es una ayuda en nuestra investigación, por tanto la información que se presentar en ella debe ser lo más atractiva posible, y de la misma manera ser intuitiva para el usuario.
- Objetivo 8: Para que el proyecto llegue a tener gran envergadura, será necesario generalizar la herramienta lo máximo posible.
- Objetivo 9: Uno de los objetivos principales para desarrollar la herramienta, es que en realidad nos ahorre tiempo en el análisis, por tanto deberemos comprobar que esto se cumple.

Capítulo 2

Estado del arte y contexto

Este capítulo se encuentra dividido en tres secciones. La primera sección está dedicada a explicar las motivaciones que nos han llevado a realizar el estudio de investigación. Donde partimos de un gran proyecto como es OpenStack para realizar el análisis de una población de tickets pertenecientes a uno de sus componentes, concretamente Cinder. Por tanto, la segunda sección se centra en explicar el proyecto OpenStack, profundizando en entender de manera superficial qué es y cómo funciona, pero centrándose más en la explicación de Cinder, ya que este proyecto es nuestro caso de estudio.

Como ya se mencionó en la introducción, OpenStack usa Launchpad como herramienta de mejora continua, notificando la posibilidad de nuevas características, errores encontrados y mejoras en el funcionamiento. Gerrit proporciona a OpenStack un repositorio de revisión de código, donde todos los cambios propuestos se debaten y se analizan para que no interfieran en el buen funcionamiento del sistema. Por último, Git es la herramienta encargada de añadir los cambios aceptados al código fuente. Estas tres herramientas se explican en mayor detalle en la tercera sección de este capítulo.

2.1. Precedentes

La literatura actual acerca de la inserción de errores en el código fuente, asienta sus bases en la idea preconcebida de que el error ha sido causado por el commit inmediatamente anterior, puesto que fue el último cambio que modificó las líneas que se vieron involucradas en el posterior arreglo. A partir de esta premisa, se desarrollan multitud de estudios y experimen-

tos relacionados con el bug-seeding, bug-fixing, o cualquier tema que involucre el concepto de encontrar al responsable de introducir el error en el código.

La idea principal en [PKWJ09] es la búsqueda de patrones de errores resueltos en el código fuente, analizando el código de los ficheros en las diferentes revisiones que nos proporciona un sistema SCM, centrándose en la semántica del código fuente.

Estudios como [KWJZ08], presentan una técnica usada para la predicción de errores latentes en los códigos, separa aquellos cambios en los que el código indica que son errores de aquellos que no son errores. Para conseguirlo, usan la ayuda del machine learning, identificando de este modo si un nuevo cambio en el código es más *buggy* o mas *clean*, es decir, es más propenso a errores o no.

Mientras que en [ICGB11] se investiga que porcentaje de desarrolladores arreglan sus propios bugs, asumiendo que los commits inmediatamente anteriores son los causantes del error. Del mismo modo, en [SZZ05], tratan al commit previo como el causante de introducir el error al código.

El afán por prevenir los errores en el código ha dado estudios como [ZZWD05] , donde se aplica la minería de datos, *Data Mining*, a las versiones de un repositorio. Con el objetivo de guiar a los programadores en los cambios realizados, sugiriendo lo que otros desarrolladores han hecho anteriormente. De esta manera consiguen prevenir los errores debidos a cambios incompletos.

Hasta el momento, no hemos encontrado ningún estudio que intente conocer si el commit anterior es el verdadero causante del error o no, si no que, todos los estudios analizados en esta memoria señalan como causante al commit previo. Por lo tanto, nuestro estudio se focaliza en ayudar a comprender mejor la causa de fallo en el código fuente, aportando a la literatura actual un posible nuevo enfoque. Nuestro objetivo no es otro que comprobar empíricamente en cuantos casos la literatura actual acerca del causante de un error se cumple y en cuantos casos la literatura podría no ser del todo correcta. Y para ello, nos centramos en el análisis de el/los commit/s inmediatamente anteriores al error introducido.

En el análisis, se tendrán en cuenta ciertas características de los commits que nos ayudan a tomar decisiones. Esta misma idea se encuentra en [SZZ05], donde al tener en cuenta ciertas propiedades sobre los cambios que conducían a posteriores arreglos, como por ejemplo el día de realización de los cambios, se encontró que los viernes se introducen más fallos en el código.

En las referencias anteriores, así como en nuestro estudio de investigación, el uso de la herramienta *Diff* es de gran utilidad. Puesto que lo usamos para identificar cambios entre diferentes versiones de un mismo fichero. Es más, los autores del algoritmo SZZ hacen uso de *Diff*. Dicho algoritmo, descrito en [SZZ05], tiene como propósito determinar el origen de un error identificando el commit que lo arregló, enlazando el número del ticket en el que se reporta el fallo con el commit que arregla ese error. Para ello, asumen que las líneas que han sido eliminadas o modificadas en el commit que arregló el error son las causantes de dicho error. *Diff* presenta un pequeño problema que envuelve a las líneas que no se modifican, si no que se cambian de lugar, y esto puede provocar un error en las características extraídas de los commits.

En [ZKZWJ06] presentan un estudio a nivel de línea, identificando las líneas que cambian y cómo lo hacen a lo largo de las versiones. Los autores eliminan el concepto de número de línea y lo sustituyen por nodos en un grafo, un nodo por cada revisión, obteniendo así una mayor información de la vida de las líneas, ya que actualmente todos los sistemas de control de versiones nos devuelven la última modificación realizada, perdiendo información que podría ser útil.

Al igual que nosotros, [KWJZ08] y [HGH08] entienden que un cambio se puede clasificar dentro de una o varias categorías dependiendo de la intención de dichos commits.

2.2. Proyecto OpenStack

OpenStack es el nombre asociado a un proyecto software, íntegramente *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 públicas como privadas. Principalmente los usuarios lo desarrollan para conseguir una infraestructura como servicio, comúnmente conocido como IaaS (Infrastructure as a Service), permitiéndole dirigir 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 está convirtiendo en una referencia de implementación en la construcción de nubes privadas y públicas. 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 convertirse en el mayor proyecto libre de IaaS, su éxito puede ser debido al poderoso framework que han creado, haciendo mas cómodo la infraestructura IaaS.

OpenStack se compone de proyectos claramente identificados como parte del núcleo o *core*, que se encuentran relacionados entre sí y dependen de la versión utilizanda. A continuación se detallan los componentes incluidos en la versión Juno (16 Octubre del 2014), ya que en nuestro estudio la vesión que estaba siendo usada era esta:

- Compute, *Nova*: Proporciona máquinas virtuales bajo demanda, también permite gestionar volúmenes de disco a través de uno de sus servicios. 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 autenticación entre el resto de componentes. Utiliza un sistema de autenticación 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

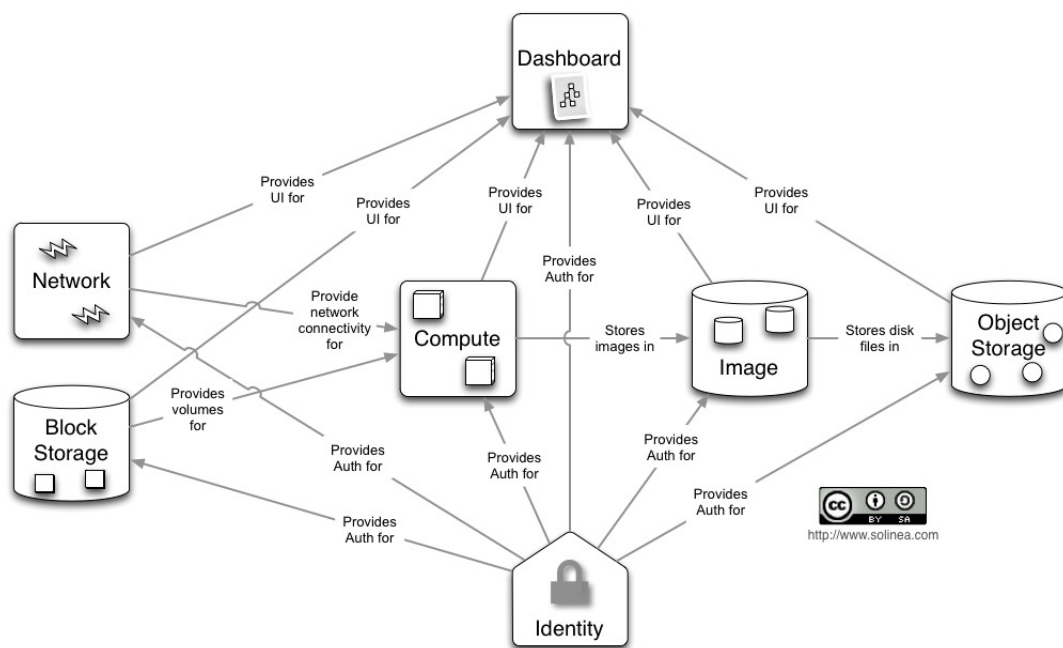


Figura 2.1: Relaciones entre los servicios de OpenStack

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 servicio de aprovisionamiento de bases de datos relacionales y no relacionales.
- **Data Processing, *Sahara***: Proporciona un medio sencillo para la provisión de un clúster 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 2.1 extraída de la siguiente url¹, se presenta una visión muy simplificada acerca de la arquitectura de OpenStack, la imagen no muestra la interacción con los consumidores del Cloud.

¹<https://wiki.openstack.org/wiki/Documentation/HowTo#Introduction>

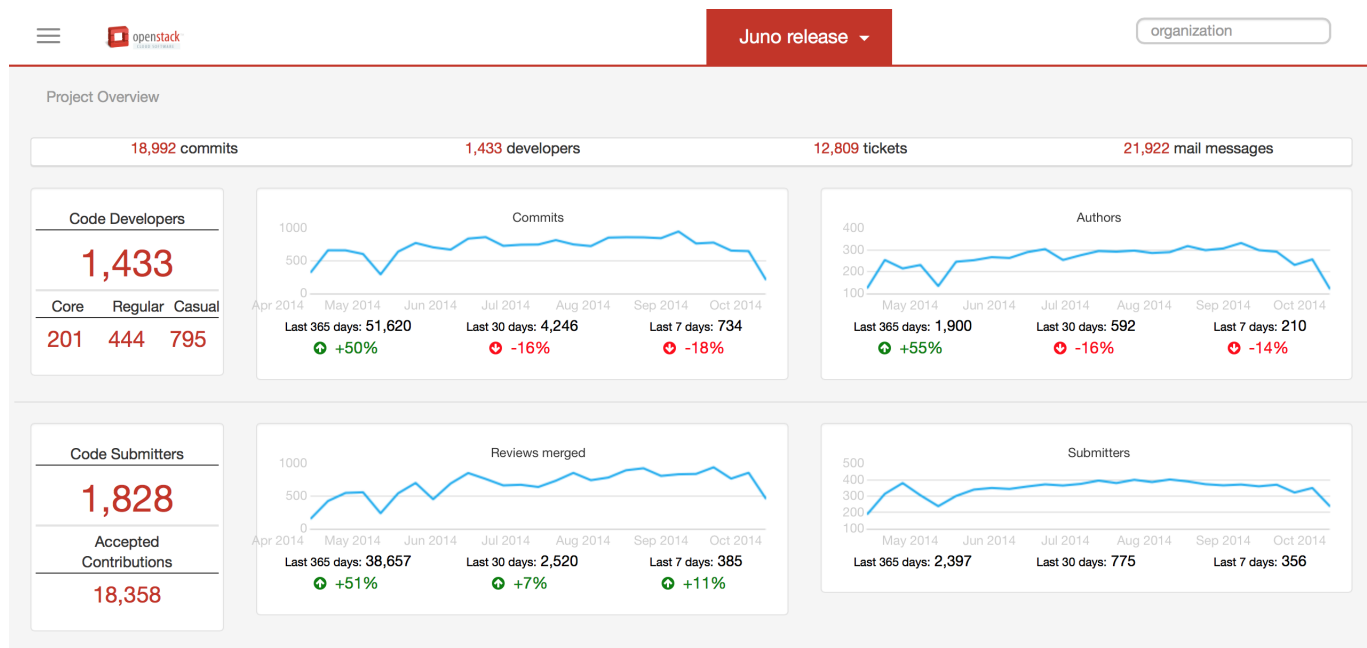


Figura 2.2: Estadísticas de la versión Juno

Como podemos comprobar, OpenStack es un gran proyecto, complejo y que se encuentra en continuo desarrollo. Los datos que lo avalan se pueden ver en la figura 2.2, extraída usando la herramienta Bicho, disponible en la url ². En la imagen podemos ver una captura de la web donde encontramos gráficas que muestran las estadísticas relacionadas con la 'release' de Juno. Podemos observar como el elevado número de tickets, de desarrolladores y de commits hacen factible la elección de este gran proyecto como caso de estudio.

Concretamente, nos centraremos en analizar uno de los componentes de OpenStack, Cinder. Nos pareció más adecuado empezar por analizar uno de sus componentes para, quizás en un futuro, acabar analizando el proyecto entero.

Cinder

Originariamente, el código de Cinder se albergaba en Nova bajo la etiqueta de '*nova-volume*'. A partir de la versión 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, manipulando volúmenes y copias instantáneas de ellos conocidas como '*snapshots*'. El Dashboard que

²<http://activity.openstack.org/dash/browser>

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 que se encuentra el estado de cada volumen.

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 o directamente sobre el almacenamiento.
- Cinder Scheduler: Selecciona el nodo donde se almacenarán por bloques los volúmenes creados.

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

En la figura 2.3 extraída de la url³ se muestra diferentes estadísticas sobre la historia de Cinder.

2.3. Herramientas utilizadas

Esta sección es la encargada de especificar de que manera OpenStack hace uso de herramientas como son Launchpad, Gerrit y Git. Antes de entrar en detalle en la explicación de cada una de las herramientas, empezaremos por una primera aproximación, relacionando OpenStack con Launchpad, Gerrit y Git.

La comunidad de desarrolladores de OpenStack utiliza Launchpad como herramienta para notificar mejoras en el programa, errores en el código, la necesidad de nuevas características, etc. Estos posts o notificaciones se denotarán como tickets a lo largo de esta memoria. Gerrit proporciona un seguimiento personalizado para cada ticket, esta herramienta nos permite poder visualizar los diferentes estados por los que pasan los ficheros involucrados en el ticket hasta

³http://activity.openstack.org/dash/browser/repository.html?repository=https\%3A__bugs.launchpad.net_cinder&ds=its

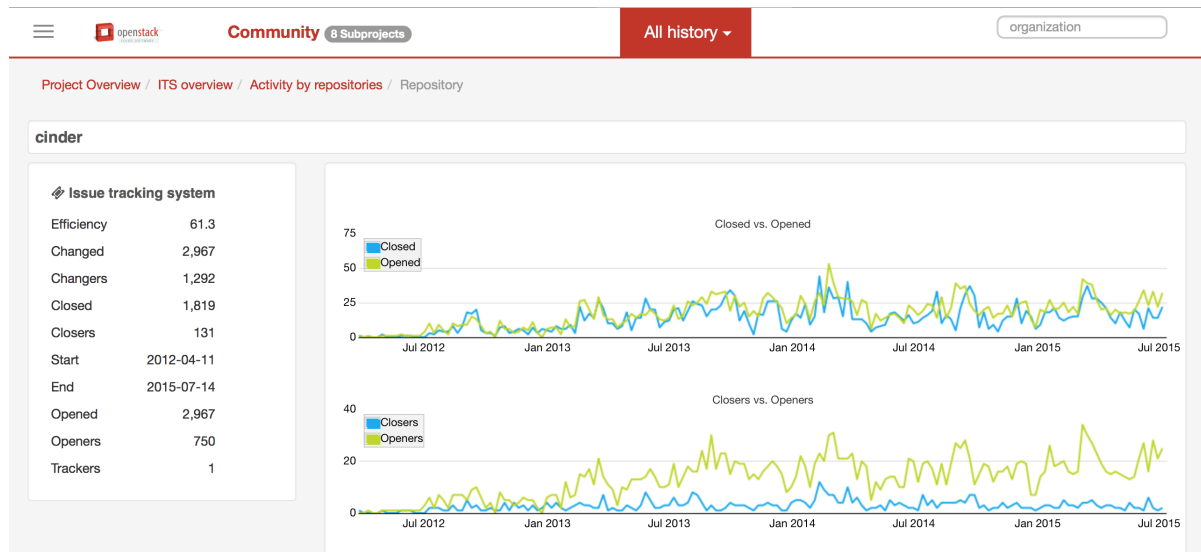


Figura 2.3: Estadísticas de Cinder

obtener la solución correcta. Por último, Git es la herramienta que proporciona un repositorio de almacenamiento de código fuente a OpenStack y es la encargada de añadir los cambios al código.

El proceso que sigue un ticket desde que se abre hasta que se cierra, se encuentra descrito a continuación:

1. Un ticket es creado atendiendo a la necesidad de cierta persona/desarrollador que cree necesario notificar una mejora, un bug, una nueva funcionalidad o algo con cierta relevancia. Ver imagen 2.5
2. Una serie de desarrolladores son los encargados de rechazar o aceptar el ticket. Si el ticket se acepta, ciertos valores asociados a él, como 'status' o 'importance', entre otros, cambian.
3. Aquel desarrollador que crea saber como se puede solucionar el problema/necesidad que se describe en el ticket, se asigna como encargado de resolver dicho ticket. En ese momento aparece Gerrit, proporcionando un hilo de seguimiento para el ticket.
4. En Gerrit, el encargado de solucionar el problema añade dicho parche, que no es más que código añadido al código fuente. Este código debe pasar unos tests y ser aprobado por varios desarrolladores encargados de supervisar que el cambio propuesto no dañe al programa. Ver imagen 2.6

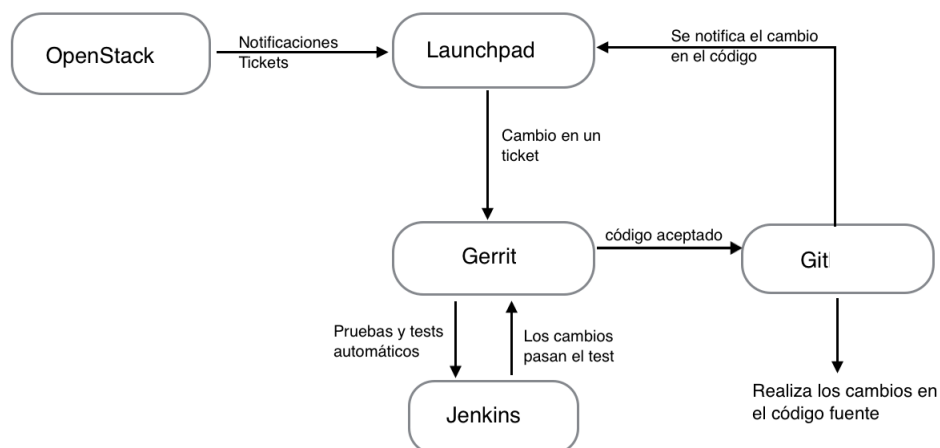


Figura 2.4: flujo de trabajo en OpenStack

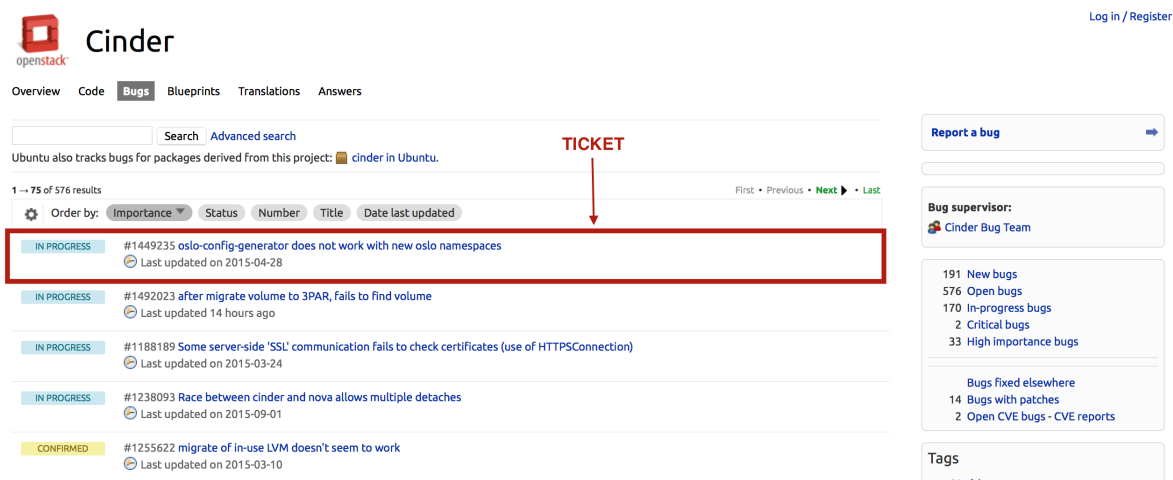


Figura 2.5: Aspecto de los tickets en Cinder

- Una vez que el código es aceptado por los desarrolladores y pasa las pruebas de test, se incorpora al código almacenado usando Git.

Puede suceder también que en algunos casos, los tickets abiertos no sean aceptados, sean abandonados o sean asignados a varias personas a lo largo del ciclo del ticket.

En la figura 2.4 aparece el flujo de trabajo que usa OpenStack entre cada una de las herramientas.

Reference Version:

Patch Set 1: 15241959a560d0f17bea3d00ea31deab2a7e9c (github)

Patch Set 2: 23c964b8da4e3c0f1e3ecb2f2588095b3034bb (github)

Patch Set 3: ddf3900ad8fa4a5ed2cdaae1c158b016190a502a (github)

Author: Jay S. Bryant <jsbryant@us.ibm.com> Apr 27, 2015 10:28 PM

Committer: Jay S. Bryant <jsbryant@us.ibm.com> Apr 27, 2015 11:47 PM

Parent(s): 2a6c1786d58f06c146d96711e454e9397a574d15f Merge "Windows: Improve vhdutils error messages"

Download: [checkout](#) | [pull](#) | [cherry-pick](#) | [patch](#) | [Anonymous HTTP](#)

git fetch https://review.openstack.org/openstack/cinder refs/changes/57/177957/3 && git checkout FETCH_HEAD

File Path	Comments	Size	Diff	
Commit Message			Side-by-Side	Unified
M tools/config/generate_sample.sh		+16, -4	Side-by-Side	Unified
		+16, -4	All Side-by-Side	All Unified

Comments: Expand Recent | Expand All | Collapse All

Jay Bryant Uploaded patch set 1. Apr 27 10:33 PM

Sean McGinnis Patch Set 1: Code-Review+1 (1 comment) Just had to point out the ... Apr 27 10:47 PM

Jay Bryant Uploaded patch set 2. Apr 27 11:38 PM

Jay Bryant Uploaded patch set 3. Apr 27 11:47 PM

Vilobh Meshram Patch Set 3: Code-Review+1 Apr 28 6:19 AM

Marcus V R Nascimento Patch Set 3: recheck HDS Apr 28 12:50 PM

Elastic Recheck Patch Set 3: I noticed jenkins failed, I think you hit bug(s): - ... Apr 28 1:51 PM

Jay Bryant Patch Set 3: recheck bug 1412513 Apr 28 4:50 PM

Sean McGinnis Patch Set 3: Code-Review+1 May 3 4:35 AM

Vipin Balachandran Patch Set 3: Code-Review+1 May 4 1:43 PM

Michal Jura Patch Set 3: Code-Review+1 Looks Good To Me Jun 17 4:22 PM

Kendall Nelson Patch Set 3: Code-Review+1 Jun 17 4:53 PM

Sean McGinnis Patch Set 3: Code-Review+2 Jun 17 5:11 PM

Yuriy Nesenenko Patch Set 3: Code-Review+1 Jun 17 6:00 PM

Gorka Egulleor Patch Set 3: Code-Review+1 Jun 17 9:34 PM

Mike Perez Patch Set 3: Code-Review+2 Workflow+1 Jun 17 10:13 PM

Jenkins Jun 17 11:08 PM

Change has been successfully merged into the git repository.

Figura 2.6: Aspecto de Gerrit para un ticket

2.3.1. 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 que permite ver reportes de errores, en caso de que quisieras reportar, comentar o subir nuevos errores deberías registrarte como usuario. Nos referiremos a estos reportes en secciones posteriores como tickets.

Launchpad⁴ consta de varios componentes

- Code: Aloja el código fuente usando un sistema de control de versiones.
- Bugs: Permite un sistema de seguimiento de errores para informar sobre errores o bugs en diferentes distribuciones y productos.
- Blueprints: Sistema de seguimiento usado para nuevas características, funcionalidad o procesos.
- Translations: Sistema destinado a la traducción de aplicaciones.
- Answers: Foro destinado a la ayuda de la comunidad, en él se resuelven dudas planteadas.

⁴<https://launchpad.net>

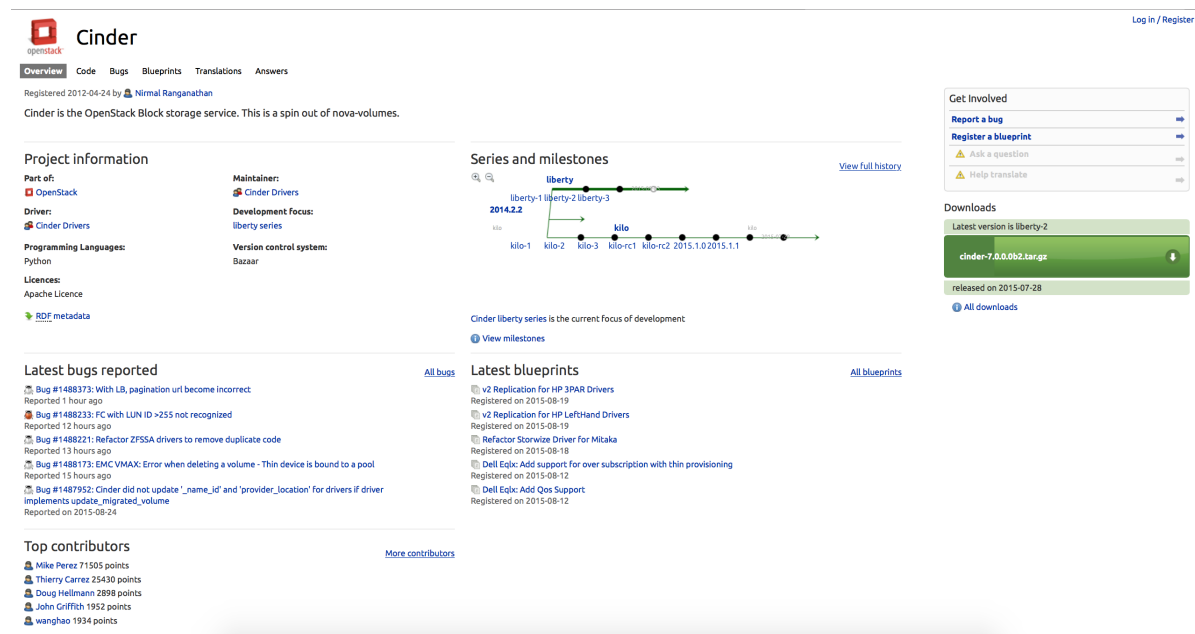


Figura 2.7: Repositorio Cinder

Cada proyecto dentro de OpenStack tiene su propia página en Launchpad, la figura 2.7 muestra la página que usa Cinder en Launchpad⁵

2.3.2. Git

Git⁶ es un sistema de control de versiones distribuido escrito originalmente en C y desarrollado por Linus Torvalds y otros 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 vez que confirmas un cambio o guardas el estado de un proyecto Git, internamente lo que se está 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 han sido modificados, sólo almacena un enlace al archivo idéntico de la versión anterior, que ya tiene almacenado ver

⁵<https://launchpad.net/cinder>

⁶<https://git-scm.com>

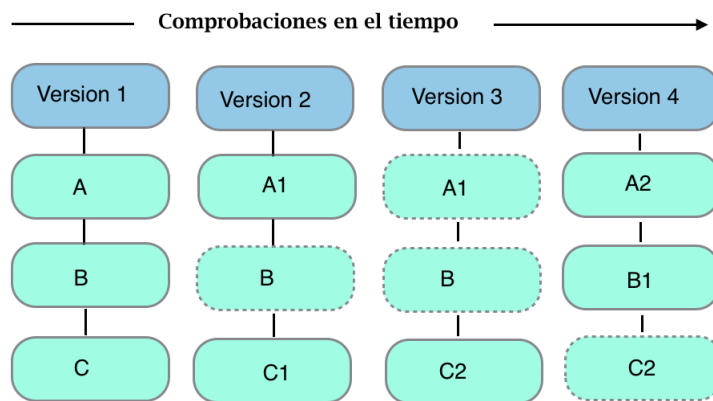


Figura 2.8: Almacenamiento de la información como instantáneas a lo largo del tiempo

figura 2.8. 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 mostrarla, 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'* o 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 en 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.

2.3.3. 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 líneas de desarrollo denominadas como *trunk* (tronco) y *branches* (ramas).

Git no permite un control centralizado por tanto, no permite bloquear cambios que puedan

dañar la *build*, y se recurre a sistemas como GitHub⁷ o Gerrit⁸ para conseguir estas funcionalidades.

Gerrit es una herramienta gratuita de revisión 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, desde dónde haríamos push y pull para recoger o enviar los cambios.

⁷<https://GitHub.com>

⁸<https://code.google.com/p/gerrit/>

Capítulo 3

Tecnologías Usadas

Esta sección se encuentra dedicada a las tecnologías que se han visto envueltas en la creación de nuestra herramienta. La herramienta hace uso de dos tecnologías principales, la primera del lado del cliente es HTML5 que agrupa las nuevas tecnologías de desarrollo de aplicaciones web como HTML5, CSS3, Bootstrap y nuevas capacidades de JavaScript, y la segunda tecnología es Node.js usada como servidor.

3.1. Tecnologías usadas en el Navegador

A continuación se describen cada una de las tecnologías que son usadas en la herramienta desde el navegador y que forman lo que se conoce como el lado del cliente.

3.1.1. 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 HTML5¹, 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

¹<https://developer.mozilla.org/es/docs/HTML/HTML5>

tal manera que las webs que usan HTML4 seguirán funcionando 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.

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 Storage que nos da la posibilidad de almacenamiento persistente en local.
- Incorpora una API para la comunicación bidireccional entre páginas, llamada WebSockets.

3.1.2. 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. 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 url² se encuentra detalladas todas las características que nos ofrece CSS3, a continuación listaremos brevemente algunas de ellas.

Novedades

- Bordes: Aparece la posibilidad de definir colores a los bordes, añadirles a los bordes imágenes, diseñar bordes redondeados o 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 o 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 o 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 o 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.

3.1.3. 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

²<http://www.css3.info/preview/>

notoriedad hasta tal punto que ha llegado a convertirse en uno de los framework CSS más famosos 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 versión 3.3.5, pero desde la versión 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*.

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.

3.1.4. JavaScript

JavaScript³ 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.

³<https://developer.mozilla.org/es/docs/Web/JavaScript>

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 aplicación del cliente, pero actualmente y gracias a tecnologías como AJAX, puede enviar y recibir información del servidor.

Historia

En los años 90 debido a la baja velocidad de los módems y a la incursión 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 encontró Brendan Eich, programador que trabajaba en Netscape, adaptando tecnologías existentes como ScriptEase 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.

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 añadir 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 está 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.

3.1.5. jQuery

jQuery⁴ 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 o 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 o incluida desde uno de los servidores de Google que nos la ofrece.

3.1.6. 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<https://jquery.com>

3.2. Tecnología usada en el servidor

En esta sección se detalla la tecnología Node.js usada como servidor por la herramienta.

3.2.1. 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.

Node⁵ 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.

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,

⁵<https://nodejs.org/en/docs/>

normalmente una cantidad pequeña de texto. No requiere una cantidad de lógica, por tanto puede 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* o la red social *Geekli.st*.

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 o Realtime SPAs, *Single Page 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...

3.3. Tecnologías de intercambio de datos

En esta sección se detalla las tecnologías que son usadas como forma de intercambio entre el cliente y el servidor.

3.3.1. JSON

JavaScript Object Notation o 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.

3.3.2. REST

La transferencia de Estado Representaciones se conoce como REST, es un estilo de arquitectura software para sistemas hipermedia distribuidos, que son aquellos que permiten contenidos como texto, imagen o video interactuando con los usuarios.

En REST, lo que se publica son recursos. Cada recurso posee un identificador que lo distingue entre cualquier otro. Poseen un estado interno, al que no se puede acceder desde el exterior directamente. Podemos acceder a representaciones de dicho estado desde el exterior, entendiendo por representación un formato de datos concreto para la transferencia de una copia del estado entre el cliente y el servidor. La implementación del recurso decide que información es visible o no desde el exterior, y que representaciones se soportan. Las representaciones pueden ser documentos XML, HTML o JSON.

En REST todos los recursos comparten una interfaz única y constante. Todos tienen las mismas operaciones, que nos permiten manipular el estado publico del recurso. La clave de REST es que es un servicio sin estado, lo que quiere decir que entre dos llamadas cualesquiera, el servicio pierde todos sus datos, el estado lo debe mantener el cliente y es él quien debe encargarse de pasárselo en cada llamada.

En la herramienta se ha implementado un APIRestFul, mediante la cual, a través de eventos programados en el lado del cliente se realizan una serie de llamadas, usando funciones y parámetros, a la API de la herramienta.

Uno de los principales motivos por los cuales desarrollamos una APIRestful es el problema del *'Cross Domain Call'*. Un Cross Domain Call, es un mecanismo de seguridad de comunicaciones entre los navegadores actuales, evita que un script o una aplicación de una página web pueda acceder a un servidor web diferente del que residen. Debido a que nuestra aplicación necesita extraer información de otros servidores diferentes al que reside, nos vimos obligados a crear la esta APIRestful ,mediante la cual hacemos peticiones a nuestras API y es el servidor quien se encarga de formular las peticiones a las APIs de las webs que no residen en nuestro

servidor.

3.3.3. OAuth

OAuth es un protocolo que permite flujos de autorización para aplicaciones web. Permite a un proveedor de servicios compartir su información con el consumidor, sin la necesidad de compartir toda su identidad.

Es un método que permite interactuar con datos protegidos y publicarlos, proporciona a los usuarios un acceso a sus datos al mismo tiempo que protege las credenciales de su cuenta. Eliminando la necesidad de crear usuario y contraseña para acceder a una web, confiando en que la web use tus credenciales sabiamente y queden protegidas de hackers. OAuth evita este problema, ya que da acceso solamente a lo que el desarrollador permita acceder.

Capítulo 4

Estudio Realizado

En este capítulo nos centramos en describir el estudio de investigación que hemos llevado a cabo, detallando cada una de las dos fases de las que consta. La primera fase denominada como fase de filtrado, se centra en descartar los tickets, que bajo nuestro criterio no son considerados como tickets de error, de aquellos que describen un error. La segunda fase denominada como causa del error, analiza si el commit antecesor provocó la aparición posterior del error.

4.1. El estudio

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, y se reporta con un identificador único en el repositorio destinado para ello, en nuestro caso, Launchpad. Cuando hablamos de commit, nos referimos al identificador que muestra un cambio realizado en el estado de ficheros de un repositorio.

El estudio consta de dos fases, la primera fase tiene como objetivo la obtención de una clasificación de los tickets analizados, dicha clasificación consta de tres categorías: 'Es un error', 'No es un error', y 'No podemos saberlo'. Cada uno de los tickets pertenecerá a una de estas categorías en base a los criterios establecidos y a la opinión personal de los desarrolladores. La segunda fase, se centra en el estudio de aquellos tickets que han sido categorizados como 'Es un error'. Obteniendo finalmente una nueva clasificación de estos tickets acorde con la responsabilidad del commit o commits antecesores. Dicha clasificación también tiene tres nuevas categorías: 'Es responsable', 'No es responsable' y 'No podemos saberlo'. Es esta última fase

la que nos proporcionará los datos necesarios para poder demostrar empíricamente en cuantos casos el commit antecesor es la causa del error en el proyecto de Cinder.

Si queremos saber más acerca del estudio, en el anexo A, podemos encontrar un artículo explicando el estudio que estamos llevando a cabo.

4.2. Primera fase o filtrado

En esta primera fase, colaboran dos desarrolladores realizando el mismo análisis paralelamente, de esta forma la clasificación de cada ticket no se ve involucrada por la opinión o comentarios de la otra persona. Dicho análisis consta de la extracción aleatoria de 100 tickets residentes en la Launchpad de Cinder cuya url se encuentra disponible en ¹. Los tickets poseen diferentes estados dependiendo de su evolución, desde que se reporta un error hasta que alguien arregla el error:

- **New:** Es cuando un ticket acaba de ser notificado en la Launchpad.
- **Incomplete:** Es cuando el informe del error esta incompleto y necesita mas información para poder ser clasificado dependiendo de sus prioridades.
- **Invalid:** Es cuando la notificación del error describe un comportamiento normal del software, o cualquier otro motivo que haga inválido al ticket.
- **Confirmed:** Es cuando un miembro de la comunidad distinto a quien reportó el ticket, acepta la descripción del ticket, en este momento cualquier desarrollador podría trabajar en una solución.
- **In progress:** Es cuando un desarrollador ha tomado la responsabilidad del corregir el error y está trabajando en ello.
- **Fix Committed:** Es cuando un desarrollador ha realizado un commit con la solución al código base del proyecto.
- **Fix Released:** Es cuando una nueva versión del software, con la corrección del error, ha sido lanzada.

¹<https://bugs.launchpad.net/cinder>

4.2.1. Búsqueda y extracción

Nosotros estamos interesados en aquellos tickets cuyo estado sea *'Fix Committed'* o *'Fix Released'*, ya que solamente en esos casos es cuando el error ha sido arreglado y los cambios realizados en el código son visibles, por tanto restringiremos la búsqueda de los tickets aleatorios a aquellos cuyo estado sea *'Fix Committed'* o *'Fix Released'*. Como ya hemos comentado anteriormente, cada ticket posee un identificador único, con el que reconoceremos al ticket en Launchpad. El ticket contiene toda la información relativa al error que se ha reportado, como por ejemplo el identificador del error en Launchpad, la descripción del error, la persona encargada de arreglar el error o una serie de comentarios que permiten ver el proceso de evolución del ticket. En la mayoría de los casos, obtenemos además la descripción del commit, el identificador del commit y la url de Gerrit, donde tenemos disponible la revisión del código.

A pesar de toda la información de la que disponemos en Launchpad, para nuestro análisis en esta primera fase, solamente necesitamos conocer el título y descripción del ticket y la url de Gerrit asociada a este ticket, para extraer de ella toda la información relacionada con el commit que arregló el error como es el identificador del commit, el identificador del commit padre, los ficheros implicados y la descripción del commit. Con toda esta información los desarrolladores deben de ser capaces de clasificar el ticket en una de las tres categorías.

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 se centra en analizar si el commit inmediatamente anterior ha sido el responsable del error. Por tanto, si lo que analizamos es un ticket en el que no se notifica un error, el estudio será erróneo. Por este motivo, a continuación detallaremos cada uno de los campos de información extraídos para ayudarnos a tomar la decisión en ésta primera clasificación.

Normalmente, un buen ticket debería tener un título claro, una descripción detallada sobre lo que actualmente está fallando, y una descripción, que iría ligada al commit, en la que se detallase cual ha sido la solución que ha arreglado el error. Cuando nos encontramos con tickets que cumplan lo anteriormente mencionado, podemos estar seguros de que el ticket realmente notifica un verdadero error. En la imagen 4.1 se puede apreciar la visualmente la buena descripción de un ticket de error. Ya que se observa que en la descripción del título aparece una palabra que nos puede indicar que se trata de un error, pero es en la descripción cuando efectivamente corroboramos que se trata de un error puesto que describe detalladamente que

es lo que está fallando y además, propone una solución. Pero no siempre es tan sencillo como lo descrito anteriormente, ya que suele ser común encontrarnos la misma descripción para el ticket y para la descripción del commit, en esos casos, tendremos que revisar detalladamente el código, analizando el estado de los ficheros implicados antes y después de que el posible error fuera solucionado. Para ello, será necesario obtener el identificador del commit que realiza el parche y el identificador de su commit padre. El commit padre no es el commit previo, puesto que esté se refiere a la última modificación realizada en las líneas que supuestamente contienen el error antes de su arreglo. Cuando hablamos de commit padre, nos referimos al último commit realizado que involucra cualquier línea de un fichero antes de arreglar el error, por tanto no tiene porqué coincidir con el commit previo.

4.2.2. Metodología

A continuación detallaremos como obtener la información manualmente, a estas alturas ya sabemos que cada ticket posee un identificador único en Launchpad, nos referiremos a él como *'n_ticket'*. Con este identificador, podemos obtener toda la información acerca del error en la web ². El título, el identificador del commit y la url de Gerrit asociada a ese ticket, a veces puede encontrarse dentro de un comentario en la misma página bajo el título de *'Fix merged to cinder (master)'*, *'Fix proposed to cinder (master)'* o *'Fix merged to cinder ({Branch})'*, el título depende del estado del commit dentro del repositorio. En alguna ocasión, el problema descrito en el ticket afecta a varios componentes de OpenStack y no siempre se añaden los parches a todos los repositorios de dichos componentes, por tanto podemos encontrarnos con tickets que no nos proporcionen la url de Gerrit en Cinder.

OpenStack nos proporciona una interfaz web donde encontramos el Gerrit Code Review system, ahí podemos encontrar los cambios propuestos y revisarlos en ³. Concretamente, podemos encontrar la revisión del código para cada *'n_ticket'* que queremos analizar en ⁴, donde *'n_review'* es el identificador de revisión único para cada *'n_ticket'*. En la revisión del código se muestra toda la información acerca del parche que arregló el error.

²https://bugs.launchpad.net/cinder/+bug/n_ticket

³<https://review.openstack.org/>

⁴https://review.openstack.org/#/c/n_review



The screenshot shows an OpenStack bug report for the 'Cinder' project. The title 'TgtAdm is broken' is highlighted with a red box and labeled 'título'. Below the title, it says 'Bug #1038062 reported by Akira Yoshiyama on 2012-08-17'. A red arrow points to the title. Below the title, it says 'This bug affects 4 people' and '36' votes. A table lists the affected projects, status, importance, assigned to, and milestone. The table has two rows: one for 'Cinder' and one for 'OpenStack Compute (nova)'. Both are marked as 'Fix Released' with a 'Critical' importance. Below the table, there are links to 'Also affects project', 'Also affects distribution/package', and 'Nominate for series'. The 'Bug Description' section is highlighted with a red box and contains a 'Summary' and 'Reason' section. A red arrow points to the 'Reason' section with the label 'Explicación del fallo'. The 'Solutions' section is also highlighted with a red box and contains two numbered solutions. A red arrow points to the 'Solutions' section with the label 'Solución propuesta'.

Cinder

Overview Code **Bugs** Blueprints Translations Answers

TgtAdm is broken ← título

Bug #1038062 reported by Akira Yoshiyama on 2012-08-17

This bug affects 4 people 36

Affects	Status	Importance	Assigned to	Milestone
Cinder	Fix Released	Critical	John Griffith	Cinder 2012.2 "folsom"
OpenStack Compute (nova)	Fix Released	Critical	Chuck Short	OpenStack Compute (nova) 2012.2 "folsom"

Also affects project ? Also affects distribution/package Nominate for series

Bug Description

Summary
=====

A patch for Bug #1011159 (<https://review.openstack.org/10417>) breaks ISCSIDriver with TgtAdm.

a) New volumes won't be exported.
b) Deleting volumes will be failed.

Reason
=====

ISCSIDriver expects tid of a volume is specified properly when it was created and the driver use tid to delete it. The driver checks that tid exists before deleting it and doesn't free it if not found. But scsi-target-utils 1.0.17 doesn't have an option to specify tid in <target xxx> tag. Tid is numbered serially from 1. So b) happens.

Currently, I can't find the reason of a). It looks exported but Nova can't mount it.

I tried to revert this patch and Nova and Cinder worked perfectly.

Solutions
=====

There are 2 solutions for b).

1. Use newer scsi-target-utils. It has an option "controller_tid" to specify tid for a target.
2. Add an argument for show_target() to specify ign and use it in TgtAdm. There is no way for tgtadm/tgt-admin to check a target with the ign but we can do "tgt-admin -s | grep ign...".

Figura 4.1: Ticket claramente clasificado como 'Es un error'

4.2.3. Clasificación

Este estudio puede ser considerado como un estudio empírico a baja escala, puesto que solamente se analizan 100 tickets, uno de los principales motivos para no realizar el estudio con un número elevado de tickets fue el carácter manual que tenía la investigación en sus inicios. Los desarrolladores eran los encargados de analizar los tickets manualmente hasta encontrar la metodología apropiada, a pesar de que solamente se han analizado 100 tickets, se encontraron muchas diferencias entre ellos en cuanto a extraer datos o incluso tomar decisiones. Debido a este motivo, nos vimos obligados a definir unos criterios básicos, que dan respuesta a posibles situaciones analizando un ticket.

A continuación se describe como actuar si nos encontramos ante alguna de estas situaciones, explicando el motivo que nos ha llevado a definir ese criterio.

- ¿El commit solamente 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 modificación que se realiza en el código debe llevar asociada su correspondiente comprobación, es decir, código que compruebe que funciona, dándole consistencia al programa. El repositorio de Cinder, existe un directorio con el nombre de 'tests' y es bajo ese directorio donde se encuentran los ficheros con el código que realiza las 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. Por esta misma razón, los ficheros de test que se encuentren involucrados en un commit no serán analizados y por tanto no los incluiremos como parte de la investigación.
- ¿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 nuestra creencia, y por tanto no lo consideraremos como un error. Además de las nuevas funcionalidad, todos aquellos tickets que notifiquen optimización del código o 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 funcionamiento 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, y debido quizás a la falta de experiencia, falta de conocimientos sobre el tema descrito en el ticket o la escasez de información obtenida, no seremos capaces de decidir si un ticket es un error o no lo es, por ese motivo tenemos el tercer grupo llamado 'No podemos saberlo', en este grupo se encontrarán todos aquellos tickets que son difíciles de clasificar.

Una vez que ambos desarrolladores acabaron de analizar y clasificar cada uno de los 100 tickets, los resultados deben ser comparados y comentados por ambas partes. Ambas clasificaciones puede tener tres posibles estados de similitud referente a cada ticket, estos estados son:

- Estado de similitud idéntica: Se da cuando ambos desarrolladores coinciden en su clasificación sobre un ticket.
- Estado de similitud moderada: Se da cuando uno de los desarrolladores ha clasificado el ticket bajo la etiqueta 'No podemos saberlo', mientras que el otro ha optado por una de las otras dos categorías.
- Estado de similitud crítica: Se da cuando ambas clasificaciones para un mismo ticket son contrarias, es decir, un desarrollador lo clasifico como 'Es un error' mientras el otro optó por clasificarlo como 'No es un error'.

Uno de los objetivos de este estudio es obtener resultados fiables, por tanto decidimos que la clasificación final fuese obtenida comparando las dos clasificaciones anteriores. De esta forma, en los tickets con similitud moderada y con similitud crítica, ambos desarrolladores deben

debatir argumentando sus posiciones y llegar de esta manera a una conclusión final. Si por el contrario, no eran capaces de llegar a ningún acuerdo en alguno de los tickets, se propuso albergar ese ticket en la categoría 'No podemos saberlo'.

En la comparativa de los resultados de clasificación, se dieron 80 estados de similitud idénticos, 16 estados de similitud moderada y 4 de similitud crítica. Después de debatir sobre aquellos tickets en los que ambos desarrolladores diferían, se llegó a un acuerdo en 13 de los casos, mientras que para 7 de ellos las opiniones de cada desarrollador seguían siendo diferentes.

Tras esta fase, obtenemos una primera clasificación, cuyos datos se presentan y se comentan en la sección 6.1.1

4.3. Segunda fase o causa del error

En la segunda fase, solo nos centraremos en los ticket que hayan sido clasificados dentro del grupo '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 podemos saberlo', ya que puede suceder que hayamos clasificado tickets que notificaban errores de verdad en este grupo. Aún así, decidimos obviar a ese grupo ya que el porcentaje de tickets pertenecientes a él es bajo, entorno al 16 %, por tanto creemos que no sería de suma importancia tenerlos en cuenta en los resultados finales pero asumimos estar cometiendo un error.

4.3.1. Análisis del código

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

Cuando hablamos de líneas añadidas, entendemos que son líneas que antes de solucionar el error no se encontraban en el/los fichero/s pero con el fin del arreglar el error, han tenido que ser añadidas. Lo mismo sucede con las líneas eliminadas, para solucionar el error, los desarrolladores encargados del ticket han tenido que eliminar líneas del código. Hasta este momento no hay ninguna duda para poder identificar estas líneas, el problema comienza al intentar identificar líneas que han sido modificadas, ya que el concepto de línea modificada puede ser confuso.

En el código pueden realizarse multitud de cambios que impliquen una modificación, por eso debemos dejar claro que es lo que nosotros entendemos por código modificado en este estudio. Para nosotros, el código modificado engloba 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, o incluso cuando se elimina código para volverlo a escribir idénticamente. A pesar de nuestro pensamiento acerca de lo que entendemos por código modificado, el uso de la herramienta *Diff*, nos complica demasiado poder diferenciar el código modificado según nuestro concepto, ya que considera los últimos casos descritos anteriormente como código modificado, por tanto, en nuestro análisis, tendremos que considerar como código modificado aquellas líneas en las que aparezca código eliminado y código añadido conjuntamente.

El análisis que se ha llevado a cabo en esta fase, se centra en definir el número de ficheros implicados, el tipo de código que se ha encontrado en dichos ficheros y el número de commits anteriores que se han encontrado. A pesar de que el objetivo es averiguar cuantos de los commits previos implicados en un cambio fueron causantes de insertar el error, hemos decidido analizar las demás características para obtener más información, de esta manera, podríamos tener en cuenta otras variables que nos ayuden en futuros análisis.

Al igual que en la primera fase, en esta también hemos definido ciertos criterios a seguir para identificar la responsabilidad del commit previo ante la inserción de un error. 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 naturaleza del código es evolucionar, por tanto si se requiere una actualización nadie puede ser el responsable. Y los commits previos que se hayan encontrado quedan exentos de responsabilidad.
- ¿El commit que arregla el error no se ha llevado a cabo en el repositorio de Cinder? No sabemos si es 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, y el cambio

After Fix Bug		Fix-inducing (Before fix Bug)		Before fix-inducing	
31f208423 711)	if rescue_auto_disk_config is None:	31f208423 711)	if rescue_auto_disk_config is None:	31f208423 701)	if rescue_auto_disk_config is None:
e30b45f69 712)	LOG.debug("auto_disk_config value not found in"	e30b45f69 712)	LOG.debug("auto_disk_config value not found in"	e30b45f69 702)	LOG.debug("auto_disk_config value not found in"
e30b45f69 713)	"rescue_image_properties. Setting value to %s",	e30b45f69 713)	"rescue_image_properties. Setting value to %s",	e30b45f69 703)	"rescue_image_properties. Setting value to %s",
e30b45f69 714)	auto_disk_config, instance=instance)	e30b45f69 714)	auto_disk_config, instance=instance)	e30b45f69 704)	auto_disk_config, instance=instance)
31f208423 715)	else:	31f208423 715)	else:		
20847c25a 716)	auto_disk_config = strutils.bool_from_string(31f208423 716)	auto_disk_config = rescue_auto_disk_config		
20847c25a 717)	rescue_auto_disk_config)				
(1)		(2)		(3)	

Figura 4.2: Ejemplo de cuando el cambio insertado en el commit previo, 31f208423, causó el error. La versión del código mas reciente se encuentra a la izquierda (1)

que arregla el error se ha ejecutado en uno de los repositorios que pertenecen a esos componentes.

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 los ficheros que están involucrados, hacemos uso de Git, sistema de control de versiones distribuido que nos proporciona comandos útiles para el análisis del código.

A continuación se muestran dos claros ejemplos que explican cuando el commit previo insertó el error en la línea 4.2 y cuando el commit previo no causó el error 4.3:

En la imagen 4.2 se observa el estado del fichero en tres versiones diferentes, la primera (1) es una vez que el error ha sido arreglado, la segunda (2) es donde se encuentra la línea con el error, y finalmente la tercera versión (3) muestra el estado del fichero antes de que el error se insertase. En la versión (3) se muestra el commit padre del commit previo 31f208423 al arreglo del error, de esta manera podemos asegurarnos de que en ese commit se insertó el error, puesto que usó un string como variable cuando debería haber usado un Booleano para mantener la concordancia con el resto del código.

En la imagen 4.3 se observa el estado del fichero en tres versiones diferentes, la primera (1) es una vez que el error ha sido arreglado, la segunda (2) es donde se encuentra la supuesta línea que contiene el error, y finalmente la tercera versión (3) muestra el estado del fichero antes de insertar la supuesta línea con el error. En esta ocasión, tras analizar los comentarios y el código, no podemos atribuir el commit previo como responsable del error, puesto que en este caso después de actualizar cierta versión en el código, el nombre del argumento de una

After Fix Bug	Fix-inducing (Before fix Bug)	Before fix-inducing
<pre> 0dc91bed 318) if VERSIONS.active < 3: 0dc91bed 319) user = manager.create(name, password, email, project, enabled) 0dc91bed 320) return VERSIONS.upgrade_v2_user(user) 0dc91bed 321) else: 0dc91bed 322) return manager.create(name, password=password, email=email, 49f9d154 323) default_project=project, enabled=enabled, cbd63f27 324) domain=domain, description=description) </pre>	<pre> 0dc91bed 318) if VERSIONS.active < 3: 0dc91bed 319) user = manager.create(name, password, email, project, enabled) 0dc91bed 320) return VERSIONS.upgrade_v2_user(user) 0dc91bed 321) else: 0dc91bed 322) return manager.create(name, password=password, email=email, 0dc91bed 323) project=project, enabled=enabled, </pre>	<pre> 68a55e3f 303) if VERSIONS.active < 3: 68a55e3f 304) user = manager.create(name, password, email, enabled) 68a55e3f 305) return VERSIONS.upgrade_v2_user(user) 68a55e3f 306) else: 68a55e3f 307) return manager.create(name, password=password, email=email, 68a55e3f 308) enabled=enabled) cbd63f27 324) domain=domain, description=description) </pre>
(1)	(2)	(3)

Figura 4.3: Ejemplo donde el commit previo, 0dc91bed, no causó el error. La versión del código mas reciente se encuentra a la izquierda (1)

variable cambió, provocando de esta manera el fallo en el código. Pero este fallo no se debe a que se insertase ningún error en la línea 323, puesto que antes de actualizar la versión el código funcionaba perfectamente, es decir, en la version (2) el código no tenía ningún error en su contexto. Por tanto, este ejemplo muestra cuando un error es causa de nuevos requisitos en el software y no podemos atribuir al commit previo como causante del error.

4.3.2. Metodología

Para empezar nuestro análisis, lo primero que hemos hecho es clonar el repositorio de 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 anterior 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 para cada línea afectada en el fichero. Para ello, usamos el comando `'git blame file'`, así obtenemos el commit y el nombre del autor que ha modificado cada una de las líneas del fichero por última vez, finalmente guardamos la salida obtenida en un fichero de texto. En segundo lugar, debemos cambiar al estado del fichero antes de que el error fuese arreglado, realizando otro `'git checkout id_commitPadre'` y realizamos el mismo procedimiento para acabar comparando los ficheros que hemos guardado con la salida obtenida tras el `'git`

```

Last login: Tue Sep  8 16:18:25 on ttys001
Gema:~ gerope$ diff ~/Desktop/sinError1.txt ~/Desktop/conError1.txt | tee ~/Desktop/diffError1.txt
576c576
< 663eadfb (Mitsuhiro Tanino    2014-06-24 19:40:36 -0400 576)      message = (_("Destination Volume Group %s does not exist") %)
> b6999068 (John Griffith      2013-07-22 10:39:54 -0600 576)      message = ("Destination Volume Group %s does not exist" %)
578c578
< 663eadfb (Mitsuhiro Tanino    2014-06-24 19:40:36 -0400 578)      LOG.error(message)
> b6999068 (John Griffith      2013-07-22 10:39:54 -0600 578)      LOG.error(_('s'), message)
Gema:~ gerope$

```

Figura 4.4: Salida que muestra el comando diff entre dos ficheros

blame file’.

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 fichero1 fichero2’* para mostrar *línea-a-línea* las diferencias entre los dos ficheros,. Diff es un algoritmo extensamente explicado teóricamente en [Ukk85] y [Mye86], el cual nos proporciona sistemas de gestión de gran cantidad de código fuente. 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 líneas.

La imagen 4.4 muestra la salida del comando diff al comparar ambos ficheros, en esta salida se puede ver tres partes diferenciadas. En los recuadros coloreados de azul, aparece el número de línea del primer fichero una letra *‘c’* y el número de línea del segundo fichero. La letra *‘c’* indica que se debe realizar un cambio en dichas líneas de los ficheros. El cambio que se debe hacer en el primer fichero es eliminar las líneas recuadradas en color verde he introducir las líneas recuadradas en color rojo. Por tanto, ya podemos identificar a los posibles commits antecesores responsables del error, en este caso, para ambas líneas es el mismo commit *‘663eadfb’*.

4.3.3. Clasificación

Tras seguir el procedimiento descrito anteriormente, obtenemos una lista con los posibles identificadores de commit causantes del error. El siguiente paso es analizar cada uno de los commits y clasificarlos en uno de estos tres grupos, dependiendo de su responsabilidad en la aparición del error. Teniendo en cuenta siempre el momento, y el contexto del proyecto, en el

que la supuesta linea que contiene el error fué añadida/modificada al código fuente.

1. Es el responsable : Cuando tenemos claro que el commit ha provocado el error. Una condición mal escrita, el mal uso de las variables, etc.
2. No es el responsable: Cuando tenemos claro que el commit no ha provocado el error. Como por ejemplo cambios en las llamadas a la API, el cambio de nombre a una variable, etc.
3. No podemos saberlo : Cuando no somos capaces de identificar si el commit ha provocado el error o no. En algunas ocasiones hay demasiados commits previos y la descripción es demasiado vaga como para identificar cual de los cambios causó el error, en otras ocasiones el no ser experto del lenguaje usado y del proyecto nos impide saber si el commit previo insertó o no el error.

Como es normal, un parche realizado en el código para solventar el error, puede tocar varios ficheros y en cada fichero podría estar implicado mas de 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 debemos dejar claro cual es el procedimiento a seguir.

- ¿El commit previo es un fork? No podemos saberlo, ya que no podemos identificarle. Un fork es cuando se realiza una copia exacta de un repositorio, el cual podremos usar a partir de ese momento como un repositorio git normal. En otras palabras, se puede entender como una bifurcación de un repositorio, en el cual hasta el fork tienen el mismo estado pero a partir de ahí cada repositorio evoluciona de manera independiente. La razón por la cual no podemos identificar al responsable, se debe a que cuando se realiza un fork, todos los archivos del repositorio son añadidos al nuevo repositorio con un identificador de commit diferente al que tenían. Ese es el motivo por el cual no podemos identificar al responsable a menos que nos cambiemos al repositorio 'padre' y sigamos investigando en él.
- ¿El commit solo ha modificado comentarios, nombre de las versiones o 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.

Al igual que en la primera fase, los dos desarrolladores han ido trabajando en paralelo analizando los tickets de error. Una vez finalizado su trabajo, el procedimiento seguido fué el mismo que en la primera fase, ambos desarrolladores compararon sus resultados, obteniendo una clasificación final, aquellos tickets que no eran similares en ambas clasificaciones fueron debatidos y se llegó a un acuerdo.

En esta fase el número de discrepancias antes del análisis fue más elevado, obteniendo 19 diferencias entre ambas clasificaciones. En 11 de ellas se produjo una similitud moderada y 8 de similitud crítica. Teniendo en cuenta que en esta fase la similitud moderada se da cuando opinan lo mismo acerca de la responsabilidad de los commits implicados, pero en una de las clasificaciones aparece algún commit más. Mientras que la similitud crítica se da cuando ambos desarrolladores opinan diferente en cuanto a la responsabilidad de los commits o cuando el identificador de dichos commits difiere.

Tras debatir ambas discrepancias y volver a revisar el código, en aquellos casos de similitud crítica, se llegó a un acuerdo unánime por ambas partes, y como consecuencia, se obtuvo una clasificación final. Después de obtener dicha clasificación, podemos generar diferentes gráficas jugando con las variables que hemos analizado. Estas gráficas se muestran y se explican en la sección 6.1.2

Capítulo 5

Descripción de la Herramienta

En este capítulo va dirigido a la descripción de la herramienta. En la primera sección se describe la arquitectura de la web. La segunda sección está dedicada a enumerar detalladamente los requisitos de la herramienta. La tercera sección, detalla la API que ha sido implementada, y que permite la conexión entre el navegador y el servidor. En la cuarta sección se detalla como se realiza la autenticación en la aplicación. Para acabar, la ultima sección ofrece de forma detallada el funcionamiento de la web.

5.1. Arquitectura

La herramienta implementada consta de dos partes claramente diferenciadas, 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, el cual ha sido descrito en la sección 5.3, de esta forma obtenemos los recursos que necesitamos en cada momento.

Nos referimos a Front-End como las estructuras HTML, los estilos CSS e interacciones JavaScript que se encargan 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, JAVA y MySQL, Postgres, SQL Server, MongoDB, respectivamente. Para ello usa 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

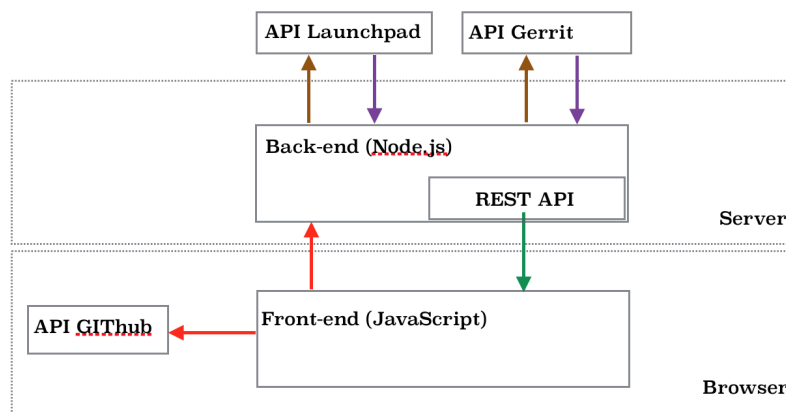


Figura 5.1: Estructura de la herramienta

se transmite la información a través del código realizado por el Front-End al lado del cliente.

En nuestra aplicación, el Back-End usa el interprete de NodeJs, actuando como servidor encargando de realizar peticiones a otras APIs, concretamente a la API de Launchpad¹ y la API de Gerrit². También se encargará de atender las peticiones de su propia API, que llegan desde el cliente, enviando toda la información requerida al Front-End en formato JSON. La elección de esta tecnología y no otra como Django o Ruby on Rails recae en mí, ya que me desenvuelvo mejor en el lenguaje javaScript que en python o ruby.

El Front-End de nuestra aplicación tiene varias tecnologías coexistiendo, ya que el estilo y la 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 GitHub.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 más amigable e intuitiva, o por lo menos esa es nuestra idea.

La herramienta no dispone de base de datos, pero en su defecto, dispone de un repositorio Git asociado a la aplicación, se encuentra albergado en GitHub. Gracias al uso de algunas librerías de javaScript y tras crear y configurar permisos para que nuestra aplicación use GitHub, disponemos de una 'base de datos' online, ofreciéndonos una gran ventaja, la alta disponibilidad para ver los datos siempre que lo necesitemos.

En la figura 5.1 se muestra la estructura de nuestra aplicación web (APP WEB)

¹<https://api.launchpad.net/1.0.html>

²<https://gerrit-review.googlesource.com/Documentation/rest-api.html>

5.2. Requisitos

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. Pero 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: La información es aquella obtenida manualmente en el estudio anterior y que nos ayuda a clasificar los tickets.
2. Obtener el identificador de los tickets de manera aleatoria: De este modo el estudio sigue teniendo un carácter aleatorio sin poseer ningún sesgo previa.
3. Recursos que permitan al desarrollador expresar su opinión durante el análisis: Para que los desarrolladores puedan tener acceso a las opiniones de otros desarrolladores en relación al análisis de un ticket.
4. Guardar la información obtenida y analizada sobre el ticket: Es imprescindible poder guardar los datos de los tickets que acaban de ser analizados.
5. Visualizar la información almacenada sobre un ticket: La aplicación debe mostrar los datos sobre un ticket que previamente ha sido guardado.
6. Modificar la información almacenada sobre un ticket: La aplicación debe ofrecer la opción de modificar el análisis de un ticket ya que el desarrollador podría haber cometido un error.
7. Visualizar los resultados de clasificación: Esta funcionalidad es requerida para proporcionar al desarrollador una visión general de la primera clasificación de los tickets.

La aplicación no es mas que una web dinámica en la que disponemos de tres pestañas '*Analyze*', '*Statistics*' y '*Modify*'. Son en estas pestañas donde se encuentran solventadas las necesidades enumeradas anteriormente.

5.3. API

A continuación pasaremos a describir cada una de las funcionalidades programadas en el lado del cliente que nos proporcionan la información que el usuario necesita.

- `get(/random)`: Se usa para la obtención de tickets aleatorios, para ello, el servidor realiza una petición url a la página de Launchpad, la parsea y extrae de ella los primeros 75 tickets, que devuelve en formato JSON.
- `get(/random/moreTickets)`: Se usa para la obtención de los siguientes tickets aleatorios, funciona igual que la anterior, pero obteniendo diferentes tickets.
- `get(/ticket/{idTicket})`: Se usa para obtener datos referentes a un ticket como son la descripción, el título, los comentarios y el link a la Launchpad. Al llegar al servidor, éste se encarga de formular una petición a la API de launchpad usando el identificador del ticket obtenido como parámetro anteriormente.
- `get(/ticket/{idTicket}/messages/)` : Se usa para obtener todos los comentarios e los que dispone un ticket. Al llegar al servidor, éste se encarga de formular una petición a la API de launchpad usando el identificador del ticket obtenido como parámetro anteriormente.
- `get(/ticket/{idTicket}/SeeData)`: Se usa para obtener los datos que han sido almacenados sobre un ticket en un repositorio concreto de GitHub, haciendo uso de la API de GitHub al llegar al servidor.
- `get(/tickets/{nameOfFile}/statistics)`: Se usa para obtener el parámetro que hace referencia a la clasificación del ticket que ya ha sido analizado y se encuentra almacenado en el repositorio de GitHub. Para ello, al igual que en el caso anterior el servidor usa la API de GitHub para obtener la información.
- `get(/review/{numReview})`: Se usa para obtener la información relativa a la revisión de código de un ticket, como son la descripción de cada uno de los cambios propuestos y el identificador del cambio. Al llegar al servidor, éste se encarga de formular una petición a la API de gerrit usando el identificador dl la revisión de código de un ticket, obtenido como parámetro anteriormente.

- `get(/commit/{idCommit}/files)`: Se usa para obtener información sobre los ficheros que han estado involucrados en el commit. Para ello, el servidor hace una petición a la API de Gerrit usando algunos parámetros obtenidos en esta llamada.
- `get(/commit/{idCommit}/aboutcommit)`: Se usa para obtener información sobre algunas características relevantes del commit implicado, como son la descripción del mismo o el identificador del commit padre. Para ello, el servidor hace una petición a la API de Gerrit usando algunos parámetros obtenidos en esta llamada.

API Launchpad

La API de launchpad se encuentra detallada en³, a continuación solo se mencionarán aquellas llamadas que hemos usado en nuestra aplicación.

- `https://api.launchpad.net/1.0/bugs/{numBug}` : Obtiene en formato JSON todas las características relacionadas con el ticket a través del identificador del ticket 'numBug'
- `https://api.launchpad.net/1.0/bugs/{numBug}/messages`: Obtiene en formato JSON cada uno de los comentarios relacionados con el ticket a través del identificador del ticket 'numBug'

API Gerrit

La API de Gerrit se encuentra detallada en⁴, a continuación solamente detallaremos aquellas llamadas que han sido usadas en nuestra aplicación.

- `http://review.openstack.org/changes/{idGerrit}/detail` : Obtiene en formato JSON todas las características relacionadas con la revisión del cambio a través del identificador asociado a la revision 'idGerrit' de un ticket.
- `http://http://review.openstack.org/changes/{idGerrit}/revisions/{idPatch}/commit`: Obtiene en formato JSON todas las características del commit asociado a un cambio 'idPatch' a través del identificador de la revision 'idGerrit' de un ticket.

³<https://api.launchpad.net/1.0.html>

⁴<https://gerrit-review.googlesource.com/Documentation/rest-api.html>

- <http://review.openstack.org/changes/{idGerrit}/revisions/{idPatch}/files>: Obtiene en formato JSON todas las características de los ficheros que han estado involucrados en el commit asociado a un cambio 'idPatch' a través del identificador de la revision 'idGerrit' de un ticket.

5.4. Autenticación

En nuestra aplicación es necesario logearnos, y para ello necesitamos tener cuenta en GitHub, otorgando permisos a la aplicación sobre esa cuenta. Necesitamos una clave publica y una clave privada para permitir que nuestra web sea una aplicación de GitHub, así se crea la conexión entre el consumidor y el proveedor. Pero en realidad, esta conexión no es directa, ya que en la aplicación usamos una biblioteca llama '*hello.js*', que permite el uso del protocolo OAuth a través de una proxy de autenticación. Para ello generamos los identificadores del cliente y el identificador secreto para la aplicación en GitHub.

Esta biblioteca hace uso de un servicio web definido como *oauth proxy*, realiza la labor de un intermediario, permitiendo que la clave secreta no viaje en el browser si no que se consulte en la base de datos y permita la negociación entre cliente y proveedor. Por tanto, nuestra web pasará la información al proxy para que él permita la comunicación con GitHub y el usuario pueda logearse en nuestra aplicación.

Cabe destacar que lo primero que se ejecuta cuando entramos en la web es la autenticación de la aplicación en GitHub, permitiendo de esta manera a los desarrolladores, con cuenta en GitHub, visualizar y guardar los datos extraídos. Si el usuario no dispone de cuenta en GitHub, la autenticación falla pero puede extraer toda la información usando la herramienta, el inconveniente es que no puede usar todas aquellas funcionalidades que dependan de GitHub, como guardar la información, modificarla, consultarla o extraer las estadísticas actuales

5.5. Funcionamiento

En las siguientes secciones, vamos a describir la aplicación web desde cada una de las pestañas disponibles en ella. Es decir, describiendo cada una de las funcionalidades que nos ofrecen los distintos botones que podemos encontrar en la interfaz de usuario. Además de la

descripción, se proporcionarán capturas de pantalla realizadas a la aplicación web en las cuales se que muestran visualmente las funcionalidades que se describen.

La primera captura de pantalla que nos encontramos, imáagen 5.2, muestra el aspecto inicial de nuestra aplicación Web. En ella podemos observar en la parte superior a la derecha, las tres pestañas que hemos comentado anteriormente. Por defecto la pestaña '*Analyze*' se muestra al abrir la aplicación.

5.5.1. Analyze

En esta sección detallaremos cada una de las funcionalidades que nos ofrecen los botones, así como el contenido que aparece en los formularios disponibles en la pestaña '*Analyze*' de nuestra aplicación web.

La funcionalidad primordial de la herramienta web desarrollada, se aloja en esta pestaña. Nos vemos en la necesidad de analizar una gran cantidad de tickets para poder proporcionar datos fiables. Para ello, los tickets necesariamente tienen que tener carácter aleatorio, de esta forma eliminamos el posible sesgo que pueda tener los resultados de la investigación debido a nuestra elección.

Para poder cumplir este objetivo, la aplicación dispone del botón '*Start*'. al clicar sobre dicho botón, a herramienta consulta la página de la Launchpad y extrae de ella los primeros 75 tickets, de los cuales se mostrará el identificador de cada uno al usuario en tres columnas desplegables. Además, internamente se está realizando una consulta al repositorio de GitHub que tenemos asociado a la investigación, en el que se almacenan los tickets que han sido analizados, para mostrarnos que tickets han sido analizados previamente, coloreando el identificador del ticket en verde.

En este punto, el desarrollador tiene tres opciones. La primera, analizar un ticket que no ha sido analizado previamente, es decir, aquellos que no están coloreados en verde. La segunda, visualizar los datos que han sido guardados sobre uno de los tickets coloreado en verde. La tercera opción, volver a realizar el análisis de un ticket que ya ha sido analizado previamente.

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*', información extraída de Launchpad acerca de ese ticket. Para obtener la información relativa a la revisión del código, el desarrolla-

Analizing Tickets

GET SOME INFORMATION

ANALYZESTATISTICSMODIFY

FILLING THE BLANKS

Ticket ID

Enter Id Ticket

Get Info

ANALIZING RANDOM TICKET

Start

Click on start button to analyse a random ticket, you can save and see the data.

Save infoSee Data

DATA TABLES

Ticket Info

WebSite	
ID	
Title	
Description	

Review Info

More Info

Website	
ID Gerrit	
ID Commit	
Description	
Files	
Lines	
Commit Parent	

Classifying into

☒ It's a bug☐ It's not a bug☐ Unknown

Adding Keywords

Title	Keywords in the Title
Description	Keywords in the Description
Commit	Keywords in the Commit

Comments

Write your comments here

Revisor

Write revisor's name here

© 2015 LibreSoft| By : Gema Rodríguez

Figura 5.2: Aspecto Web inicial

The screenshot shows a web application for analyzing tickets. At the top, there's a pink header with the text 'Analyzing Tickets'. Below it is a dark red bar with the text 'GET SOME INFORMATION'. The main content area is divided into two sections: 'FILLING THE BLANKS' and 'CLICK IN THE TICKET'.

FILLING THE BLANKS

Ticket ID

Enter Id Ticket

Get Info

CLICK IN THE TICKET

More Tickets

1466851	1495758	1494459
1361360	968696	1479530
1490845	1490641	1473001
1494574	1479897	1479342

Click over the num of ticket to fill some fields.

Save info

See Data

Figura 5.3: Obtención aleatoria y visualización de los tickets en la web

El usuario debe pulsar el botón *'More Info'*, y la tabla *'Review Info'* se rellenará automáticamente con la información extraída de Gerrit sobre dicho ticket.

Es en este momento cuando el desarrollador puede categorizar el ticket en uno de los tres grupos que disponemos, *'Es un error'*, *'No es un error'* y *'No estamos seguros'*. Además puede añadir las palabras claves que ha encontrado en el título, en la descripción del ticket y en la descripción del commit, así como expresar su opinión. Para ello, la herramienta dispone de unas cajas de texto donde el usuario puede escribir y un selector de categoría.

En los formularios que han sido rellenados automáticamente, aparecen enlaces a páginas de interés como son la página de Launchpad o la página de Gerrit. Los enlaces aparecen para proporcionar al usuario mayor nivel de información, por si la mostrada le pareciese escasa o incluso porque necesita visualizar el código que ha sido modificado entre la versión actual y la versión anterior de cada uno de los ficheros implicados para poder llegar a una conclusión. 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* debemos elegirlas cuidadosamente, ya que nos serán útiles en un futuro. En esta fase, de momento nos sirve como una recolección de muestras, las cuales analizaremos en fases más avanzadas. Determinando como influye su presencia en los tickets, para ellos haremos uso de algoritmos de machine learning, con el objetivo de obtener una clasifica-

DATA TABLES

Ticket Info

WebSite	https://bugs.launchpad.net/bugs/1384379
ID	1384379
Title	versions resource uses host_url which may be incorrect
Description	The versions resource constructs the links by using host_url, but the glance api endpoint may be behind a proxy or ssl terminator. This means that host_url may be incorrect. It should have a config option to override host_url like the other services do when constructing versions links.

Classifying into

☒ It's a bug
 ☐ It's not a bug
 ☐ Unknown

Adding Keywords

Title	<input type="text" value="Incorrect"/>
Description	<input type="text" value="Incorrect, should have,"/>
Commit	<input type="text" value="Add, set"/>

Comments

Revisor

Review Info More Info

Website	https://review.openstack.org/159374
ID Gerrit	159374
ID Commit	2eb25ab8803214cb3beb5d8fe3efb70a462c414
Description	Add config option to override url for versions The versions url returns the wrong data when cinder api is behind a proxy. This adds a new config option so it can be set properly. DocImpact Change-Id: I45a90120b21e43bf8dca9e5f0efdf339f0d3e8e6 Closes-Bug: #1384379
Files	cinder/api/views/versions.py
Lines	Inserted: 16 Deleted: 1
Commit Parent	1e2ac58e7c18a1cad72b1e6b70b43da96510243

Figura 5.4: Datos Extraídos y comentarios del Desarrollador

ción automática de los tickets basándonos en la semántica descrita en el título, en la descripción del ticket y en la descripción del cambio.

La figura 5.4 muestra el aspecto final que debería tener la herramienta tras el auto-relleno de los formularios 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 los datos situados en los campos de información de la web a un formato JSON, y lo almacena en el repositorio destinado para nuestro análisis albergado en GitHub. Añadiendo un archivo de texto, cuyo contenido son los datos en formato JSON y cuyo nombre es *NumDelTicket_NombreDelRevisor*. De esta forma, sabemos quién es ha sido el revisor y cual ha sido ticket que se ha revisado.

En esta pestaña se añade además, 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', situados en la parte izquierda de la aplicación, ver figura 5.3.

Internamente, lo que se está ejecutando tras pulsar sobre el identificador de un ticket obtenido aleatoriamente o sobre el botón 'Get info', es un evento Java Script 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).

Ahora, solo falta comentar la otra opción que tiene el desarrollador y de la cual hemos hablado al principio y es pulsar sobre uno de los tickets coloreados en verde. Tras pulsar en el identificador del ticket, podemos volver a realizar el proceso que acabamos de describir para obtener un segundo análisis del ticket, o podemos visualizar la información guardada sobre ese ticket. Si optamos por lo segundo, lo único que debemos hacer es pulsar el botón '*See Data*', y se mostrará en la parte derecha de la web, una tabla con toda la información almacenada en el ticket, como podemos comprobar en la imagen 5.5.

5.5.2. Statistics

Al pulsar sobre esta pestaña, aparecen unos selectores que muestran el nombre de cada uno de los revisores que ha contribuido en el análisis de los tickets en su primera fase. Al pulsar sobre los selectores, se rellena una tabla de datos con información relativa a los resultados de clasificación de la fase de filtrado. Podemos entenderlo como las estadísticas de clasificación de los tickets para un 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' y 'Unknown', así como el porcentaje en '%' que se ha obtenido en cada grupo.

Además, disponemos de la posibilidad de dibujar un gráfico de barras en el que se muestre visualmente el porcentaje de tickets para cada grupo. Como se puede observar en la imagen 5.6.

Para conseguir esto, lo único que debe hacer el usuario es pulsar sobre el botón '*Draw*'. A la derecha del gráfico disponemos de un menú desplegable que nos permite descargar la imagen, mostrada en gráfico de barras, con diferentes extensiones.

5.5.3. Modify

Tras pulsar sobre la última 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 aplicación, en la parte derecha aparece un formulario vacío, el cual se

DATA TABLES

Ticket Info

WebSite	https://bugs.launchpad.net/bugs/1384379
ID	1384379
Title	versions resource uses host_url which may be incorrect
Description	The versions resource constructs the links by using host_url, but the glance api endpoint may be behind a proxy or ssl terminator. This means that host_url may be incorrect. It should have a config option to override host_url like the other services do when constructing versions links.

Review Info

More Info

Website	https://review.openstack.org/159374
ID Gerrit	159374
ID Commit	2eb25ab8803214cb3beb5d8fe3efbf70a462c414
Description	
Files	
Lines	
Commit Parent	

Info saved about the actual ticket

Close

CONTENTS:

LAUNCHPAD : <https://bugs.launchpad.net/bugs/1384379>

ID : 1384379

GERRIT : <https://review.openstack.org/159374>

IDGERRIT : 159374

IDCOMMIT : 2eb25ab8803214cb3beb5d8fe3efbf70a462c414

FILES : cinder/api/views/versions.py

LINES_INSERTED : 16

LINES_DELETED : 1

COMMITPARENT : 1e2ac58e7c18a1cad72b1e6b70b434da96510243

CLASSIFICATION : Bug

KEYWORDSTITLE : Incorrect

DESCRIPTION : Incorrect, should have,

KEYWORDSCOMMIT : Add, set

COMMENTS : In my opinion it's a bug because

Figura 5.5: Visualización de los datos tras pulsar el botón 'See Data'

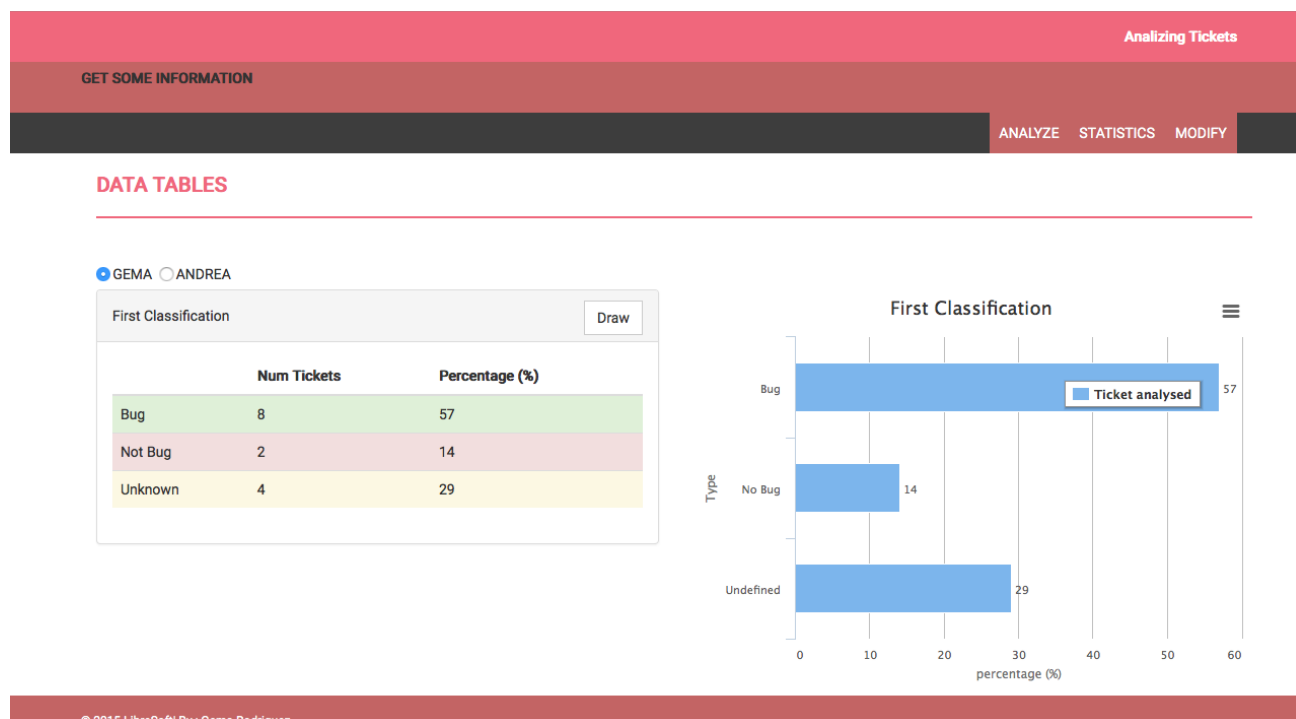


Figura 5.6: Visualización de la Web tras pulsar la pestaña 'Analyze'

autocompletará tras clicar sobre un ticket.

Esta funcionalidad está diseñada con el fin de proporcionar al desarrollador un lugar en el que pueda modificar aquellos tickets que ya han sido analizados y 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 y no nos parece apropiado que se pueda cambiar el nombre del desarrollador. Podría suceder que el propio desarrollador se equivocase al introducir su nombre, en ese caso, habría que eliminar el ticket desde el repositorio de GitHub.

El aspecto que posee la web tras haber pinchado en la pestaña 'Modify' aparece en la figura 5.7

Analizing Tickets

GET SOME INFORMATION

ANALYSESTATISTICSMODIFY

MODIFY FILES FROM REPOSITORY

Actual Files In Repo

- 1304234_GEMA
- 1351971_GEMA
- 1351971_GEMA2
- 1384379_GEMA
- 1391311_GEMA
- 1415087_GEMA
- 1446750_GEMA
- 1466851_ANDREA
- 1474596_GEMA
- 1475285_GEMA
- 1476786_GEMA
- 1476797_GEMA
- 1478236_GEMA2
- 1488916_GEMA
- 1488916_GEMA2

1384379_GEMA

Id Ticket	Id Gerrit	Lines Inserted	Lines Deleted
1384379	159374	16	1

Revisor

GEMA

ID Commit

2eb25ab8803214cb3beb5d8fe3efbf70a462c414

ID Commit Parent

1e2ac58e7c18a1cad72b1e6b70b434da96510243

Launchpad

https://bugs.launchpad.net/bugs/1384379

Gerrit

https://review.openstack.org/159374

Files

cinder/api/views/versions.py

Keyword Title

Incorrect

Keyword Description

Incorrect, should have,

Keyword Commit

Add, set

Comments

In my opinion it's a bug because

☒ Bug ☐ Not Bug ☐ Unknown

Write File!

© 2015 LibreSoft| By : Gema Rodríguez

Figura 5.7: Visualización la Web tras pulsar la pestaña 'Modify'

Capítulo 6

Resultados

Este capítulo se divide en dos secciones. La primera de ellas referente a los resultados obtenidos durante el estudio, en sus dos fases del experimento. Y la segunda sección se centra en detallar los resultados generales del trabajo fin de máster.

6.1. Resultados del estudio

A continuación se exponen los resultados obtenidos tanto en la primera fase de nuestro experimento, como en la segunda fase. En ambos casos, se muestran las gráficas y tablas resultantes tras analizar los datos obtenidos.

6.1.1. Resultados primera fase

Los resultados obtenidos en la primera fase del experimento, hacen referencia a la primera clasificación, agrupando los tickets en 'Es un error', 'No es un error' y ' No podemos saberlo'.

Classification First Stage	
Groups	Tickets belonging to (%)
It is an error	63
It is not an error	21
Unknown	16

Cuadro 6.1: Clasificación de los tickets dentro de cada grupo

La primera tabla obtenida, cuadro 6.1, representa el porcentaje de tickets para cada una de las tres clasificaciones. Podemos ver que de los 100 tickets analizados más de 50 % son clasificados como tickets que notifican un verdadero error, mientras que un 21 % de los tickets describían situaciones que no han sido consideradas como error debido a que eran actualizaciones requeridas, nuevas funcionalidades u otros casos. En esta tabla se observa también como un 16 % de los tickets no han podido ser clasificados con la etiqueta error o no error, es un porcentaje relativamente pequeño como para que pueda influir en gran medida en los resultados de la segunda fase. A pesar de ello, en este momento se asumirá que estamos cometiendo un pequeño error en los datos de la siguiente fase debido a este 16 %.

La segunda tabla obtenida, cuadro 6.2, muestra el porcentaje de los tickets agrupados en una categoría, dependiendo de la importancia que se les dió en Launchpad a lo largo de su vida. En la figura 6.1 podemos ver dichos datos en un gráfico de barras.

Classification depend of the importance				
Importance	It is an error	it is not an error	Unknown	Total
Critical	4	1	1	6
High	25	3	7	35
Medium	16	3	4	23
Low	8	6	3	17
Undecided	10	8	1	19
Total	63	21	16	100

Cuadro 6.2: Clasificación de los tickets dependiendo la importancia de éstos en Launchpad

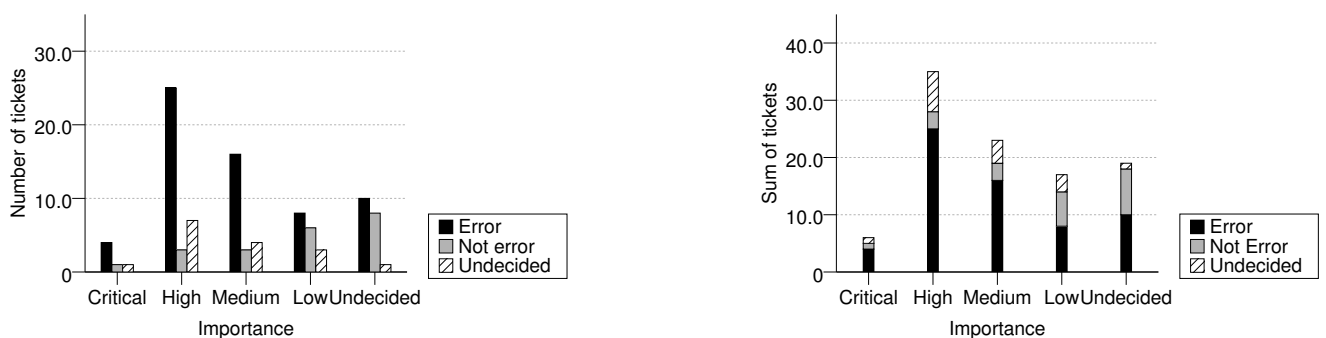


Figura 6.1: Resultados de la primera fase

La conclusión que podemos extraer de esta tabla, asumiendo que la comunidad de Launchpad usa con criterio las etiquetas de importancia en sus tickets, son las siguientes:

1. Un ticket cuya etiqueta de importancia sea *Critical* tiene un 66,66 % de posibilidades de ser un error.
2. Un ticket cuya etiqueta de importancia sea *High* tiene un 71,42 % de posibilidades de ser un error.
3. Un ticket cuya etiqueta de importancia sea 'Medium' tiene un 69,56 % de posibilidades de ser un error.
4. Un ticket cuya etiqueta de importancia sea *Low* tiene un 47,05 % de posibilidades de ser un error.
5. Un ticket cuya etiqueta de importancia sea *Undecided* tiene un 52,63 % de posibilidades de ser un error.

Por lo tanto, el factor de importancia de un ticket, podría ser considerado como ayuda para los desarrolladores a la hora de decidir si un ticket es error o no lo es. Además, este factor puede utilizarse conjuntamente con la presencia de ciertas palabras o keywords para proporcionar unos mejores resultados.

6.1.2. Resultados segunda fase

En esta sección hablaremos sobre los datos obtenidos en la segunda fase de clasificación, donde los resultados indican si fue el/los commits inmediatamente anteriores el/los causantes de insertar el error introducido en el código fuente.

La primera tabla obtenida, cuadro 6.3, muestra el porcentaje de commits previos en cada una de las tres categorías. En ella hemos identificado 124 commit previos, y tras analizarlos observamos como en el 36 % de ellos no insertaron el error en el momento en que se introdujo la línea en el código.

Acabamos de demostrar con los datos obtenidos en nuestro experimento empírico, que la actual premisa establecida en la literatura acerca de que el responsable de añadir un error en el código es el commit previo, sólo se cumple en un 53 % de los casos. Este dato no puede

Classification Second Stage		
Classification	Number of commits	Percentage (%)
It is responsible	66	53
It is not responsible	44	36
Unknown	14	11

Cuadro 6.3: Clasificación de la responsabilidad del commit anterior

ser generalizado puesto que hemos analizado una población relativamente pequeña de tickets, concretamente 100, de los que solamente 63 han pasado la fase de filtrado. A pesar de no poder asegurar que estos porcentajes se cumplan al analizar una población mayor de tickets, si podemos asegurar que hay una leve tendencia que nos indica a pensar que no todos los commits previos son responsables.

Es más, si dividimos aquellos tickets en los que sólo ha habido un commit previo implicado, de aquellos en los que ha habido más de un commit implicado, los resultados son, cuanto menos, curiosos. A continuación se muestran los porcentajes de los commits anteriores en cada una de las tres categorías, tanto para el caso de que sólo tengamos un commit antecesor, cuadro 6.4, como para el caso de los que tienen más de un commit, cuadro 6.5.

Responsibility when only it is a commit	
Responsibility	Number of tickets (%)
It is responsible	75,86,
It is not responsible	6,90
Unknown	12,24

Cuadro 6.4: Clasificación de la responsabilidad del commit anterior

De los 63 tickets que pasaron a la segunda fase, 29 de ellos solamente tenían un commit previo involucrado, mientras que 25 de los tickets tenían más de un commit. A los 9 tickets restantes no se encontró ningún commit previo puesto que los errores se habían solucionado en el repositorio de otro de los componentes de OpenStack distinto a Cinder y no pudimos extraer los datos.

Ahora bien, en la tabla 6.4 situada encima de este párrafo y perteneciente a los tickets con sólo un commit, podemos observar como en el 75,86 % de ellos, el commit previo fue el

responsable de introducir el error. Mientras que solamente el 6,90 % de los commits no fueron responsables del error. En este caso, podemos pensar que nuestra hipótesis no es significativa y se podría seguir con la premisa actual de que el commit previo es el responsable del error, en cierto caso algo lógico puesto que solamente un commit previo se ve implicado.

Lo interesante aparece cuando observamos la tabla, cuadro 6.5 con los resultados de los tickets donde se ven implicados más de un commit previo. Ahí podemos comprobar como nuestra hipótesis vuelve a recobrar fuerza, ya que el porcentaje de commits previos responsables y el porcentaje de commits previos no responsables se puede considerar idénticos, puesto que en un 46 % de los casos el commit previo es responsable mientras que en un 44 % de los casos el commit previo no es responsable.

Responsibility when there are more than one commit	
Responsibility	Number of tickets (%)
It is responsible	46,32
It is not responsible	44,21
Unknown	9,47

Cuadro 6.5: Clasificación de la responsabilidad del commit anterior

En las gráficas inferiores se muestra unos gráficos de barras obtenidos con los datos de las tablas 6.4 y 6.5 .

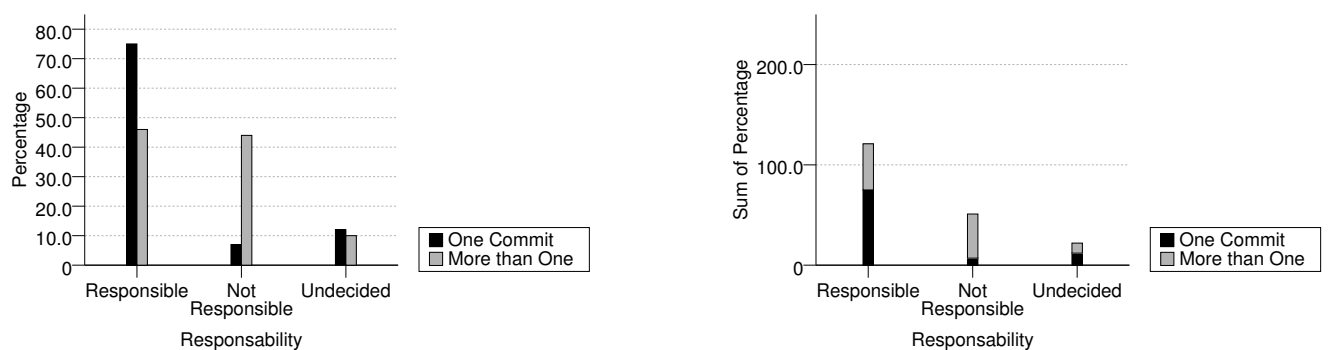


Figura 6.2: Resultados de la segunda fase

Con todos los datos analizados, podemos realizar multitud de tablas que nos ayudan de una forma u otra a obtener diversas tablas como la mostrada en el cuadro 6.6 , en ella podemos visualizar el grado de responsabilidad que ejercen todos los commits previos en un ticket. Dicha

tabla separa los tickets en los cuales hay dos, tres, cuatro y más de cinco commits implicados, obteniendo los siguientes resultados:

1. Un ticket en el cual se vean implicados dos commits, en un 57 % de los casos la responsabilidad se reparte, mientras que en un 43 % de los casos los dos commits son responsables del error.
2. Un ticket en el cual se vean implicados tres commits, en el 100 % de los casos la responsabilidad del error supera el 50 % , por tanto dos de los tres tickets son responsables.
3. Un ticket en el cual se vean implicados cuatro commits, en un 33 % de los casos la responsabilidad es menor al 50 %, mientras que en un 66 % de los casos la responsabilidad supera el 50 %.
4. Un ticket en el cual se vean implicados más de cinco commits, en un 83 % de los casos la responsabilidad del error es menor del 50 %.

Con esto, podemos decir que podría haber una tendencia marcada, en la cual, aquellos tickets que tengan más de cinco commits previos, la mayor parte de ellos no son responsables de haber introducido el error. Mientras que para tickets con 3 y 4 commits previos, la responsabilidad que ejercen la mayor parte de los commits previos es elevada.

Grade of total responsibility of each commit inside a ticket			
Number of commits	less than 50 %	responsibility = 50 %	more than 50 %
2	0	57,14	43,86
3	0	0	100
4	33,33	0	66,66
+5	83,33	3	16,66

Cuadro 6.6: Grado de responsabilidad de cada commit dentro de un ticket

En las gráficas inferiores se muestra unos gráficos de barras obtenidos con los datos de las tablas 6.6

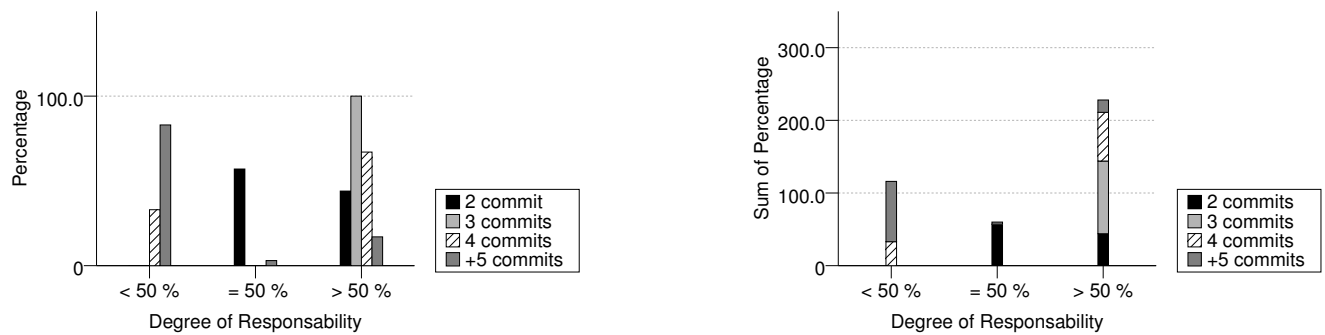


Figura 6.3: Grado de responsabilidad de cada uno de los commits en un ticket

6.2. Resultados generales

Tras la realización de este proyecto se han obtenido dos items, el primero de ellos se trata del estudio empírico realizado, del que tenemos resultados favorables para continuar investigando en este área. El segundo item, se trata de la herramienta desarrollada que servirá de ayuda en los consiguientes análisis.

La duración del proyecto ha sido aproximadamente un año, los primeros seis meses estuvieron dedicados a entender el problema, realizando pruebas y análisis iniciales con el fin de acabar encontrando una metodología adecuada. Los siguientes dos meses, estuvieron dedicados a analizar los 100 tickets extraídos del proyecto Cinder, en la fase de análisis conté con la ayuda de mi compañera Andrea Villaverde, debatiendo opiniones y comprobando que la metodología descrita era la adecuada. La herramienta fue desarrollada y validada durante los ultimo cuatro meses, añadiendo cada vez mas funcionalidades, hasta cumplir con los objetivos. Obteniendo una herramienta que posee aproximadamente de 1000 líneas de código desarrollado accesibles en ¹. La herramienta se encuentra disponible en <http://bugtracking.libresoft.es>, por tanto cualquier persona que quiera probarla puede hacerlo.

En <http://gemarodri.GitHub.io/Presentacion/> podemos encontrar una web que reúne todos los recursos disponibles en el TFM, como son la memoria, la herramienta desarrollada y su código fuente. Además hemos añadido una demo en la que se presenta el funcionamiento de la herramienta.

¹<https://GitHub.com/Gemarodri/BugTracking>

Capítulo 7

Conclusiones

Con este capítulo llegamos al final de nuestra memoria, describiremos en él si los objetivos marcados en los primeros capítulos se han cumplido o no. Hablaremos además, sobre los conocimientos aplicados y las lecciones aprendidas en el transcurso de este proyecto, así como de las líneas de trabajo futuro. Finalizando dicho capítulo con una valoración personal.

7.1. Consecución de objetivos

Hemos sido capaces de situar al lector en un plano en el que pueda entender la literatura actual sobre el área de conocimiento que engloba el bug- seeding, comprendiendo la necesidad de investigar la importancia que el commit previo ejerce en la responsabilidad de un error. Entendiendo el proceso necesario para rastrear un error, por tanto hemos cumplido con el *Objetivo 1*.

En el capítulo 4 se ha detallado todos los pasos de los que consta nuestra metodología, de tal manera que cualquiera pueda realizar el mismo experimento, cumpliendo así el *Objetivo 2*.

Hemos aportado datos y gráficas fiables en las secciones 4.2 y 4.3. Por tanto los *Objetivo 3*, *4*, y *5* quedan realizados. Dichos datos incitan a seguir investigando, dándole un mayor alcance a la investigación, puesto que creemos que los resultados son favorables a nuestro planteamiento.

El *Objetivo 6* no hemos sido capaces de realizarlo completamente, ya que solo se ha automatizado la primera fase del estudio en la herramienta. Esto se debe a que la segunda fase todavía recomendamos hacerla manualmente, debemos tener muchos factores en cuenta al analizar código y probablemente, cuanto más código analicemos más situaciones diferentes pode-

mos encontrarnos, para tenerlas todas ellas en consideración.

Creemos que la herramienta cumple con el *Objetivo 7*, y que su uso pueda servir de ayuda en estudios posteriores. Además la herramienta posee un carácter genérico, por tanto no solo sirve para el estudio de Cinder, si no que se puede usar para analizar cualquier otro componente de OpenStack, como indica el *Objetivo 8*.

El estudio sirve como método de validación para la herramienta, y después de usar la herramienta podemos decir que el *Objetivo 9* se ha cumplido, ya que ahorramos tiempo en el análisis de la primera fase.

7.2. Aplicación de lo aprendido

Muchos han sido los conocimientos adquiridos en estos dos años de Máster, gracias a ellos, el trabajo descrito en la presente memoria ha sido más fácil. Asignaturas como DAT *Desarrollo de Aplicaciones Telemáticas*, ISRH *Integración de Servicios y Redes Heterogéneas*, ISI *Ingeniería de Sistemas de Información* o GORS *Gestión y Operación de Redes y Servicios*, han aportado las tablas para poder manejarlas, con relativa soltura, a la hora de programar o para resolver problemas.

Los conocimientos y habilidades previamente adquiridos durante los cursos, de los cuales he hecho uso durante este trabajo fin de máster, son los siguientes:

1. Se han aplicado los conocimientos sobre las tecnologías que engloban HTML5, así como el lenguaje JavaScript y el entorno NodeJS, adquiridos en las asignaturas anteriormente nombradas.
2. Una de las grandes habilidades que me ha aportado el TFM, es el aumento de la capacidad autodidacta que todo trabajo de investigación requiere. Durante el máster, muchas de las asignaturas requerían cierto trabajo de investigación por parte del alumno. Los profesores aportaban ideas y conocimientos para que fuese el propio alumno quien llegase a una solución válida, potenciando de esta manera el aumento de la capacidad autodidacta.
3. Por último, se han aplicado los conocimientos en la integración de diferentes tecnologías para lograr crear un Mushup, que integra varias herramientas en una sola. Asignaturas como DAT e ISRH ayudaron en que la herramienta se desarrollase con mayor velocidad.

7.3. Lecciones aprendidas

Durante el desarrollo del TFM, muchos han sido los conocimientos adquiridos, ya que dicho TFM empieza con el estudio y entendimiento de un nuevo campo de conocimiento llamado bug-seeding, donde encontramos unas motivaciones que nos llevaron a realizar un estudio empírico. Y fue durante el desarrollo del estudio cuando encontramos la necesidad de crear una herramienta que nos proporcionase ayuda en nuestro análisis.

Por tanto, puedo enumerar cuales han sido los conocimientos y/o habilidades adquiridos:

1. Profundizar los conocimientos sobre el *FLOSS*.
2. Comprensión del estado actual en el que se encuentra el campo de conocimiento tratado en este TFM, el Bug Seeding.
3. Aprendizaje de la metodología de trabajo que posee un gran proyecto como es OpenStack, entendiendo el uso de herramientas como Launchpad y Gerrit en dicho proyecto.
4. Creación de una APIRestFul necesaria en nuestra herramienta.
5. Montar un servidor node en una maquina virtual que albergue nuestra herramienta.
6. Describir una metodología detallada y precisa, de tal manera que pueda ser automatizada, facilitando el estudio a los desarrolladores.
7. Adentrarme en el entendimiento del lenguaje, desconocido para mi, Python.

Las lecciones que puedo sacar tras todo el trabajo realizado en el TFM son:

1. En los primeros meses de la investigación, el deseo de obtener los primeros resultados con la mayor rapidez posible, provocó una pérdida tiempo analizando tickets sin llegar a entender en profundidad cual era el objetivo final. Teniendo que repetir, en varias ocasiones, el análisis. Por tanto, entendí que es más importante dedicar el tiempo necesario a entender los objetivos y el problema que se plantea antes de conseguir resultados rápidos.
2. Analizar cual son las necesidades que tenemos y como la tecnología nos facilita el trabajo.
3. Comprender el estado del arte actual para entender como ayuda nuestra investigación en él.

7.4. Trabajos futuros

Hay dos líneas de trabajo futuro, una relacionada con el trabajo de investigación y la otra relacionada con la herramienta que hemos desarrollado. A pesar de que ambas líneas deberán ser complementarias, podemos permitir que la línea de investigación siga su curso, llevando a cabo experimentos con nuevas ideas que han surgido gracias a la realización de este primer estudio. Mientras que la herramienta puede seguir evolucionando y mejorando, aportando más funcionalidades, siempre y cuando siga cumpliendo las necesidades que el estudio de investigación requiera.

Por todo ello, los trabajos futuros en la investigación son:

- Extender la investigación a otros componentes de OpenStack.
- Experimentar con los datos obtenidos y el machine learning en busca de características claves que nos ayuden a tomar la clasificar los tickets con la mayor precisión posible.
- Obtener información más detallada sobre los commits y los ficheros involucrados, ya que podríamos obtener nuevas relaciones que nos ayuden en un futuro para predecir si el commit es el causante o no. Actualmente, no hemos realizado la extracción de dichas características en nuestro estudio.
- Añadir nuevos campos para analizar, como son la hora, el día de la semana o incluso en nombre del responsable de introducir el commit para encontrar alguna relación en el momento de introducir el error en el código.

Si nos centramos en la herramienta, los posibles trabajos futuros se centran en:

- Automatizar la segunda fase, incluyéndola en la herramienta.
- Proporcionar una mayor interactividad con el usuario cubriendo sus necesidades, para ello sería imprescindible que un grupo de desarrolladores probasen nuestra herramienta y nos aportasen sugerencias o mejoras.
- Añadir la representación de gráficas y tablas de clasificación que actualmente no se encuentran disponibles como son aquellas obtenidas en la segunda fase del experimento.

- Brindar al desarrollador sugerencias de clasificación basadas en las keywords y otras posibles características que puedan ayudar. Para ello previamente hemos tenido que usar machine learning.
- Estandarizar todo lo posible la herramienta para que pueda usarse con cualquier proyecto que use Launchpad y Gerrit.
- Optimizar al máximo el código, ya que no sabemos como puede afectarle el trabajar con una gran cantidad de datos.

7.5. Valoración personal

Hallar un nuevo area de conocimiento, antes desconocida para mi, en la cual hay muchas posibilidades de investigación, me ha aportado esa curiosidad de querer seguir investigando, aportando nuevas ideas y aprendiendo cada día un poco más sobre este tema.

El esfuerzo ejercido durante la investigación y las fases de análisis se ve recompensado al llegar a obtener datos que avalan una teoría inicial. Además poder crear una herramienta que permita ahorrar tiempo de análisis, en la cual puedas añadir funcionalidades dependiendo de tus necesidades, es realmente gratificante.

Poder hacer uso de las tecnologías más activas actualmente como son HTML5 y Bootstrap y afianzar de la misma manera los conocimientos sobre JavaScript y jQuery, me ha aportado una mayor seguridad para afrontar futuros retos en los cuales se requiera el uso de estas tecnologías.

Por todo ello, me gustaría concluir diciendo que la valoración personal ha sido realmente positiva, he aprendido muchas cosas las cuales podré aplicar en el futuro, tanto a nivel personal como a nivel profesional, y me siento en el deber de dar las gracias al grupo de LibreSoft, ya que son ellos los que me han ayudado en todo momento para en el TFM.

Apéndice A

Artículo realizado

En la memoria se adjunta el artículo que hemos escrito, donde se describe detalladamente la investigación llevada a cabo. El artículo plantea y da respuesta a ciertas preguntas de investigación, y detalla el proceso seguido para extraer los datos de los que hemos obtenidos distintas gráficas y conclusiones

BUG SEEDING AND THE IMPORTANCE OF PREVIOUS COMMIT

Gema Rodríguez Pérez

Abstract

This is a study about code review, the main objective in this paper is the study of when an error was caused by the previous commit, where a commit is a change to the source code submitted. For us, the previous commit is the commit immediately prior. However, there is not enough empirical evidence. We have conducted an observational study that englobes one hundred tickets extract from a free and open-source cloud computing software platform. The experiment was focused as a first approach to prove the premise established does not hold for all cases. The results of this study shows a trend in the data involving that non fixed bugs were introduced by the prior commit.

1 Introduction

The software engineering has a key problem, the continuous code changes. So, it is important to help developers understand and manage this process. The code changes involve the emergence of bugs, and with them, the possible failure of the software. Concepts as Bug Seeding help us to find where the bug was inserted in the source code [9], and who was the responsible. Currently, you can find from several tools [3] [1] which analyze the code and can prevent you when a potential bug is found, to a wide collection of articles which describes how find an error in the source code [7] [8] or who is the responsible.

But all of them have something in common, from the initial studies an implicit assumption was mentioned in which errors were caused by the previous commit as in [2], where the authors introduce a new technique to classify changes into two main categories, buggy or clean. But they take on that the last commit in the line was where the responsible bug was.

Others authors [5] do not even mention who is the responsible, because they have assumed the fixed bugs are introduced by the previous commits. Nevertheless, there could be a doubt about that this empirical evidence exists and we decided to try in a particular case of OpenStack.

We chose OpenStack as research project because is a big project with a really active community of developers. Furthermore, it is backend by important companies as Cisco, Paypal, Ebay, NASA, Dell, HP or Intel, and it has become the most Open Source IaaS project in the three last years.

The study is based on an empirical experiment where one hundred tickets were extracted from an specific repository of OpenStack. Understanding the ticket as a written evolution of a bug from it is opening until it is fixed, aka bug report. The analysis of these tickets helps us to scope the principal aim, which it is none other than find the cases where we can corroborate that the error was caused by the previous commit.

The present study answers the following research questions:

- Research question 1 : Is the ticket analyzed a real bug?

- Research question 2: What percentage of real bug fixes can we corroborate, in a safe way, that the antecesor change caused the error?
- Research question 3: What percentage of real bug fixes can we corroborate, in a safe way, that the antecesor change did not cause the error?
- Research question 4: How can we classify the antecesor changes?

This paper describes our empirical experience to classify the percentage times that the actual teorema is correct, is not correct and is difficult to give a proper classification. We discuss how to extract data from a specific repository and classify them as well as how the experiment was done in Section 2. The research questions are answered in section 2.1 y 2.2. Section 3 explains some of the threats that our experiment is exposed to, the intrinsic characteristics and the extrinsic characteristics. Section 4 closes with the conclusions of this study.

2 The experiment

OpenStack is an open source cloud computing platform for all types of clouds, this project was created by developers and cloud computing technologists from around the world. OpenStack is a good project to be analyzed because of their high scope and their heterogeneous nature, hundreds developers are contributing to provide an infrastructure for the world's largest brands. OpenStack has about 3,283 tickets, from which it has closed about 462 tickets through 135 different developers in all its history. Statistics have been provided by Bicho at <http://activity.openstack.org/dash/browser/>, a tool which retrieves and organizes information from issue tracking systems.

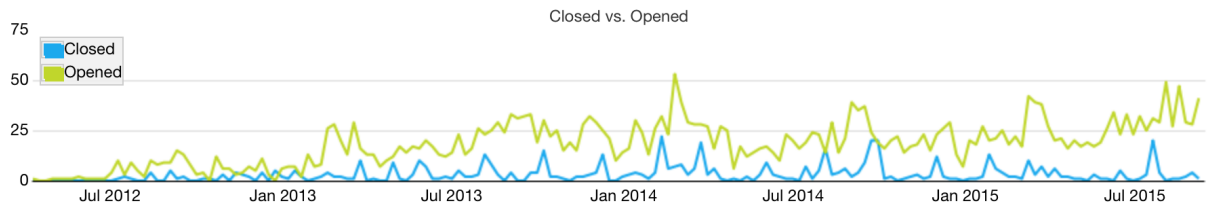


Figure 1: Evolution over the years of tickets closed and closers

We have carried out an empirical experiment where one hundred tickets were analyzed. These tickets were taken randomly from Cinder repository. Cinder is a component of OpenStack, it works as a Block Storage manipulating volumes and snapshots. Cinder has a Data Base in which the state of each volume can be found.

OpenStack is a combination of software tools for building and managing cloud computing platforms for public and private clouds. Principally, users deploy it as an infrastructure as a service (IaaS) solution. The technology consists of many different moving parts. In particular, OpenStack has nine key components that can be identified as part of the 'core'. Officially, OpenStack community maintained these system.

- Compute (Nova) is the primary engine of an IaaS system. It is used for deploying and managing virtual machines.
- Object Storage (Swift) is a scalable redundant storage system for objects and files.

- Block Storage (Cinder) manages the creation, attaching and detaching of the block devices to servers, being able to access specific locations on a disk drive.
- Networking (Neutron) provides the networking capability for OpenStack managing IP addresses.
- Dashboard (Horizon) is the only graphical interface to OpenStack whereby administrators can gain graphical access.
- Identity (Keystone) provides a central directory of users mapped to the OpenStack services that administrators and users can access.
- Image Service (Glance) provides virtual copies of services to OpenStack. It allows use of these images as templates when deploying new virtual machine instances.
- Telemetry Service (Ceilometer)
- Orchestration (Heat) helps to manage the infrastructure needed for a cloud service to run through both a REST API and a Query API.

Each one of the above has its own API to achieve integration. Furthermore, all of them have a repository where the community implements improvements; fixing bugs ... etc.

We focused in Cinder repository, which is composed of:

- Cinder API: Accepts orders and routes them to cinder volume.
- Cinder Volume: Reacts to these requests, writing or reading the database. Furthermore, it interacts with other processes such as cinder Scheduler through the message queue. Also acts directly upon storage, providing hardware or software.
- Cinder Scheduler: Selects the node to storage and volume where created.

The experiment consists of two stages in which two different persons with knowledge of programming working in parallel analyzed and classified the tickets. Only at the end of the two stages did these people compare the results with each other, thus reaching an agreed classification when a problematic difference was found.

2.1 First Stage

The first stage considers a classification, based on our knowledge about the ticket we have analyzed, in three groups: 1. When we are sure that the ticket it is an error. 2. When we are sure that the ticket it is not an error. 3. When we have doubts about how classify the ticket, it is hardly classifiable. Henceforth, we will refer to group 1 as 'Error', group 2 as 'Not Error' and group 3 as 'Undecided'. In OpenStack, bugs reports for a project are generally tracked on Launchpad of this project, so in our case, we extracted the bugs reports from Launchpad Cinder at <https://bugs.launchpad.net/cinder>. Bugs reports with different status' are being found at this Launchpad, but we only are interested in tickets whose states are 'Fix Committed' or 'Fix Released' because only in these cases the changes in the code are visible. Each ticket has an id unique at Launchpad, and with this id we can see all the information about the bug.

First of all, we must be clear in what we will analyze from the ticket that will help us to classify them. We must carefully read the title and the description of the ticket as well as the title of the commit which fixed the patch, because this is where we will obtain important information to decide to which group it belongs. Each ticket has an id unique at Launchpad, we

will refer to it as 'n_ticket', and with this id we can see all information about the bug at https://bugs.launchpad.net/cinder/+bug/n_ticket. The title of the commit can, sometimes be found in the same link under the comment 'Fix merged to cinder (master)'. OpenStack provide us a Web interface for the Gerrit Code Review system to see proposed changes, and review them at <https://review.openstack.org/>, specifically we will find this code review to each 'n_ticket' at https://review.openstack.org/#/c/n_review, where 'n_review' is the review unique id of each 'n_ticket'. In the review code all the information about the patch that fixed the bug is shown.

In spite of low number of tickets we have analyzed, both developers found main differences to extract data or to classify. In addition, some tickets did not present the same characteristics as others, so we need to define certain criteria that must be followed in case of doubt about it being a bug or not. We have based the answer on the following criteria:

- Are there only test files? It is not an error. Our criteria indicate that test files will not be analyzed because we consider these files indispensable when a feature is implemented. With a test file the working order of the program is checked, in a way to give consistency to the program. A bug in the code, implicates there being a bug in the test file, provided that the test has been implemented. So, we are not interested in these files.
- Does the title of the ticket describe a new feature? It is not an error. New features are not considered errors, because there is no failure. The optimization, deletion of a dead code or new characteristics to users are involved here.
- Does the title of the ticket describe the program as not working as expected? It is an error. Sometimes the error is not in a main function which prevents Cinder from working, because a developer has considered it an important error, although not relatively important. But it is not working as he/she expected.
- Does the title of the ticket describe that updates are required? It is an error. We consider all tickets that require updating as errors, because if it is not updating Cinder is not operating as expected and will cause errors.

Sometimes we are unable to answer all the questions due to having insufficient data or because of the complexity of the issue, in this case, the ticket will be classified into the 'Undecided' group.

At this moment, we are ready to answer the request 1 showing a graph where a percentage of the tickets are in each of three groups mentioned before showing.

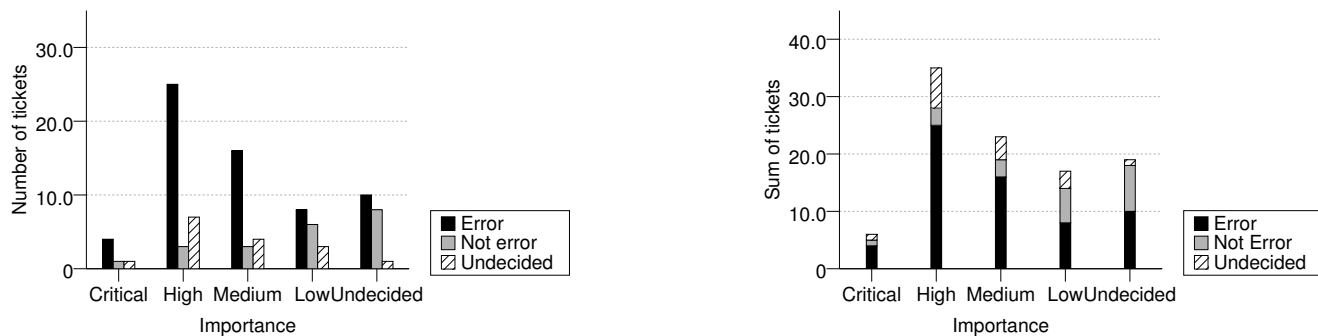


Figure 2: Percentage of each group depending on the importance

In figure 2 is shown a classification of three groups which depend on the importance of label that the developers into Cinder had given to ticket.

2.2 Second Stage

In the second stage, we only focused in the analysis of 'Error' group. Therefore, we are committing an error in the final results due to some errors being lost when we discard the 'Undecided' group because probably in this group there are some bugs that we haven't classified as such. In fact, this is not very important because the percentage of that group is only 16% and the final results won't have relevant changes if we considered them. So, we decided to assume that error.

Now the more difficult part begins. Commits will be focused on, having to identify which lines have been removed, modified or added in each of them. Also we should know in which files are the answers to the question about who is responsible.

Sometimes, after having read the description of the ticket, we know if the ticket is an upgrade error. In this case, according to our criteria nobody is responsible for the evolution of the code. Therefore, if an update is necessary, despite being an error there is no commit responsible for the bug. In other cases, after the description we can conclude that we are unable to identify a responsible commit, because the ticket includes more than one component within OpenStack and although it has been opened in Cinder could have been closed in another repository that belongs to another component of OpenStack. Therefore we do not have a review where we analyze the code and identify the commit responsible.

If the ticket is not in any of the above situations, we have to analyze some characteristics of the commit as the kind of code it is, the number of commits involved and the number of folders modified by the fix commit. There is a free and open source distributed version control system called Git that provides us with several helpful commands for the analysis in the code review.

Firstly we have cloned locally the Cinder repository using the 'git clone'. Then once we have chosen a ticket, extract the id of the fix commit and the id of parent commit from a web of Gerrit as well as the name of the files involved. We use 'git checkout id commit' to change between different status of these files, beginning with the id of the fix commit to obtain all characteristics mentioned before. So, using 'git blame file' command in our local repository, we find what revision and author last modified each line of a file, we save the output in a file. Then, we changed branch into the parent commit and did the same to later compare both files.

Continuing with the methodology of the experiment, we have assumed there are three kinds of codes depending on the character of the lines, which are on the involved files.

1. New Code: When every change was executed, lines were added to each of the files.
 2. Deleted Code: When every change was executed, there were deleted lines in each of the files.
 3. Modified Code: When the previous two codes are combined. This is the most common type of code as we can see in figure 3
-
1. Number of added lines: Are the number of lines in a file ,which have been added in the actual commit (commit2) with respect to the previous commit (commit1).
 2. Number of deleted lines: Are the number of lines in a file, which have been deleted in the actual commit with respect to the previous commit.
 3. Number of modified lines: Are the sum of the number of added lines and the number of deleted lines.

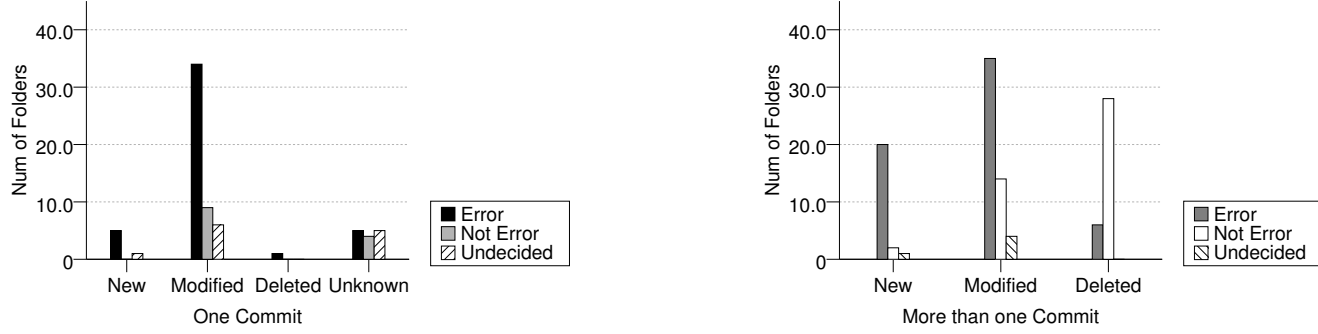


Figure 3: Number of folders in each group depend on type of code

According to our experience analyzing these 100 tickets, the modified code and the deleted code are better than the new code because we have a limited reference as to where the application was failing. But, identifying the modified code is really difficult because the same lines could be in both files but located differently. So, we have been careful with this, and have checked it manually. We use 'diff -B file1 file2' command to display linebyline difference between two files, the option '-B' meaning ignore blank lines. Diff is an algorithm extensively explained at [6], [4], which provides us several source code management systems. This tool examines both files and returns what changes, showing the number of line, need to be made for first file and second file to match, that is, returns the differences found between them.

At this stage, we got a list with all the previous commits, which could be the responsible for the bug. At times, we can discard some commits belonging to the previous list due to the commit only adding a blank line, added or modified commentaries, or maybe, it adds to the actual number of the version file.

Once we have that list, we have to identify those commits responsible. When there are more than one commit implicated in the same file, not everyone has to be responsible for the commit. Therefore, comparing the status of the file, before commit and after commit, we are able to identify those commits that are responsible and those that are not. Obtaining the percentage of the degree of involvement of each of the commits involved classifying them according to their responsibility in:

1. It is the responsible
2. It is not the responsible
3. Undecided

It is now that we are ready to answer the request 2 and request 3 showing a percentage about when the antecesor change caused the error, when the antecesor change did not cause the error, and the cases hardly classifiable.

The figure 4 shows the percentage of each group depending on how many commits have been implicated in the ticket. 'More than one commits' is referring to the number of independent commits without considering tickets.

The figure 5 shows the responsibility in 'More than one commit' depend on the degree of responsibility of each commit inside a ticket.

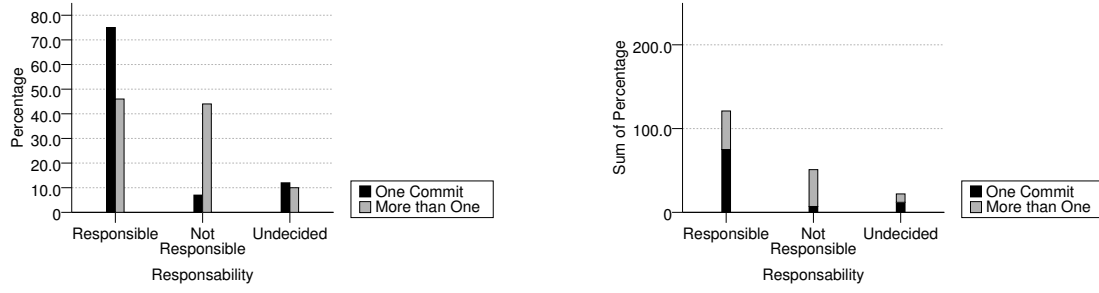


Figure 4: Responsibility depends on number of commits implicated

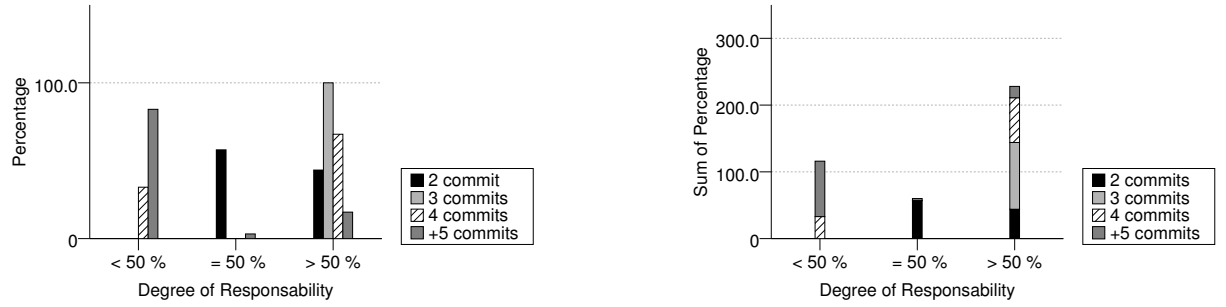


Figure 5: Degree of responsibility in more than one commit

3 Threats to validity

We have analyzed only 100 tickets, so our conclusions in the main are mere estimates, but along the right line. This is the first big threat. It may happen, that only with 100 seemingly random tickets, there may be a tendency, a prior unknown which matches the day of the week or month of the year.

Even so, setting aside the number of ticket analyzed, our model has a number of threats both intrinsic and extrinsic that make our model not 100% valid. Begin with the intrinsic threats: Begin with the intrinsic threats:

- Have not taken into account errors that have been classified into 'Undecided' group.
- There could be some lax criteria involving the subjective opinion of the reviewers.
- We are not experts analyzing and classifying tickets, so it could favor the appearance of a little bias.
- Errors that have not been classified due to lacking of a review closed into Cinder repository.
- We only are using a the part of the information that the ticket provide us, like comments and text. There could be a recognized pattern, but unknown at first sight, that englobes other part of the information, or the whole information.
-

Following with the extrinsic threats: Related to developers:

- Continually mentioned bug word into description and commit of a ticket when it is not an error. It could lead the reviewer to an incorrect classification
- The lack of detail in the description of the ticket implies that reviewers should be experts in OpenStack. Increasing the percentage of group 'Undecided' if it does not.
- The personal opinion of the developers to attach a grade of importance label could modify the figure 2.

4 Conclusions

The conclusions that we can extract after analyzing the data obtained and seeing the graphs, corroborate our first thoughts about the responsibility practiced by the previous commit. Now, we have a little empirical evidence which demonstrate that in a 33 % of cases analyzed, the premise assumed before, which involve the previous commit as a responsible, is not fulfilled.

In spite of the lower population of tickets used in the research project, we can be almost certain that this pattern would be repeated when a biggest population of tickets are used. In that way, we would contribute to the current literature with this idea.

References

- [1] Mikołaj Fejzer, Michał Wojtyna, Marta Burzańska, Piotr Wiśniewski, and Krzysztof Stencel. Supporting code review by automatic detection of potentially buggy changes. In *Beyond Databases, Architectures and Structures*, pages 473–482. Springer, 2015.
- [2] Sunghun Kim, E James Whitehead Jr, and Yi Zhang. Classifying software changes: Clean or buggy? *Software Engineering, IEEE Transactions on*, 34(2):181–196, 2008.
- [3] Sunghun Kim, Thomas Zimmermann, Kai Pan, and E James Whitehead Jr. Automatic identification of bug-introducing changes. In *Automated Software Engineering, 2006. ASE'06. 21st IEEE/ACM International Conference on*, pages 81–90. IEEE, 2006.
- [4] Eugene W Myers. Ano (nd) difference algorithm and its variations. *Algorithmica*, 1(1-4):251–266, 1986.
- [5] Jacek Śliwerski, Thomas Zimmermann, and Andreas Zeller. When do changes induce fixes? *ACM sigsoft software engineering notes*, 30(4):1–5, 2005.
- [6] Esko Ukkonen. Algorithms for approximate string matching. *Information and control*, 64(1):100–118, 1985.
- [7] Zuoning Yin, Ding Yuan, Yuanyuan Zhou, Shankar Pasupathy, and Lakshmi Bairavasundaram. How do fixes become bugs? In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, pages 26–36. ACM, 2011.
- [8] Thomas Zimmermann, Sunghun Kim, Andreas Zeller, and E James Whitehead Jr. Mining version archives for co-changed lines. In *Proceedings of the 2006 international workshop on Mining software repositories*, pages 72–75. ACM, 2006.

- [9] Thomas Zimmermann, Andreas Zeller, Peter Weissgerber, and Stephan Diehl. Mining version histories to guide software changes. *Software Engineering, IEEE Transactions on*, 31(6):429–445, 2005.

Bibliografía

- [bCL11] Launchpad Help by Canonical Ltd. Launchpad help. <https://help.launchpad.net>, 2011.
- [Con15] Software Freedom Conservancy. Git documentatios. <https://git-scm.com/doc>, 2015.
- [Fou15a] JQuery Foundation. jquery, write less, do more. <https://jquery.com>, 2015.
- [Fou15b] Node.js Foundation. Node.js information. <https://nodejs.org/en/>, 2015.
- [FWB⁺15] Mikołaj Fejzer, Michał Wojtyna, Marta Burzańska, Piotr Wiśniewski, and Krzysztof Stencel. Supporting code review by automatic detection of potentially buggy changes. In *Beyond Databases, Architectures and Structures*, pages 473–482. Springer, 2015.
- [Git15] Github. Documentation. <https://help.github.com>, 2015.
- [HGH08] Abram Hindle, Daniel M German, and Ric Holt. What do large commits tell us?: a taxonomical study of large commits. In *Proceedings of the 2008 international working conference on Mining software repositories*, pages 99–108. ACM, 2008.
- [Hos15] Google Project Hosting. Gerrit code review- a quick introduction. <https://review.openstack.org/Documentation/intro-quick.html>, 2015.
- [ICCGb11] Daniel Izquierdo-Cortazar, Andrea Capiluppi, and Jesus M Gonzalez-Barahona. Are developers fixing their own bugs?: Tracing bug-fixing and bug-seeding com-

- mitters. *International Journal of Open Source Software and Processes (IJOSSP)*, 3(2):23–42, 2011.
- [Koc05] Stefan Koch. *Free/open source software development*. Igi Global, 2005.
- [KWJZ08] Sunghun Kim, E James Whitehead Jr, and Yi Zhang. Classifying software changes: Clean or buggy? *Software Engineering, IEEE Transactions on*, 34(2):181–196, 2008.
- [KZPWJ06] Sunghun Kim, Thomas Zimmermann, Kai Pan, and E James Whitehead Jr. Automatic identification of bug-introducing changes. In *Automated Software Engineering, 2006. ASE'06. 21st IEEE/ACM International Conference on*, pages 81–90. IEEE, 2006.
- [Lib15] Bootstrap LibrosWeb. Bootstrap 3, el manual oficial. https://librosweb.es/libro/bootstrap_3/, 2015.
- [MES03] David MacKenzie, Paul Eggert, and Richard Stallman. *Comparing and Merging Files with GNU diff and patch*. Network Theory Ltd., 2003.
- [Mye86] Eugene W Myers. An (nd) difference algorithm and its variations. *Algorithmica*, 1(1-4):251–266, 1986.
- [Nic15a] Mozilla Developers Network and individual contributors. Developers tools html5. <https://developer.mozilla.org/es/docs/HTML/HTML5/>, 2015.
- [Nic15b] Mozilla Developers Network and individual contributors. Javascript. <https://developer.mozilla.org/es/docs/Web/JavaScript>, 2015.
- [Pep11] Ken Pepple. *Deploying openstack*. .“Reilly Media, Inc.”, 2011.
- [PKWJ09] Kai Pan, Sunghun Kim, and E James Whitehead Jr. Toward an understanding of bug fix patterns. *Empirical Software Engineering*, 14(3):286–315, 2009.
- [ŚZZ05] Jacek Śliwerski, Thomas Zimmermann, and Andreas Zeller. When do changes induce fixes? *ACM sigsoft software engineering notes*, 30(4):1–5, 2005.

- [Ukk85] Esko Ukkonen. Algorithms for approximate string matching. *Information and control*, 64(1):100–118, 1985.
- [w3s15] w3schools.com. Css3, introduction. http://www.w3schools.com/css/css3_intro.asp, 2015.
- [YYZ⁺11] Zuoning Yin, Ding Yuan, Yuanyuan Zhou, Shankar Pasupathy, and Lakshmi Bairavasundaram. How do fixes become bugs? In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, pages 26–36. ACM, 2011.
- [ZKZWJ06] Thomas Zimmermann, Sunghun Kim, Andreas Zeller, and E James Whitehead Jr. Mining version archives for co-changed lines. In *Proceedings of the 2006 international workshop on Mining software repositories*, pages 72–75. ACM, 2006.
- [ZZWD05] Thomas Zimmermann, Andreas Zeller, Peter Weissgerber, and Stephan Diehl. Mining version histories to guide software changes. *Software Engineering, IEEE Transactions on*, 31(6):429–445, 2005.