Who introduced this bug? It may not have been caused by the previous commit!

Gema Rodríguez
GSyC/LibreSoft
Universidad Rey Juan Carlos
Madrid, Spain
gerope@libresoft.es

Jesús M. González-Barahona
GSyC/LibreSoft
Universidad Rey Juan Carlos
Madrid, Spain
jgb@gsyc.es

Gregorio Robles
GSyC/LibreSoft
Universidad Rey Juan Carlos
Madrid, Spain
grex@gsyc.urjc.es

ABSTRACT

It is common practice that developers mark in the versioning system when they are fixing a software bug. At first glance, it could seem reasonable to assume that the fixed bug had been introduced in the previous modification of those same parts of the source code (i.e., in the previous commit). In fact, many studies on bug seeding start with this assumption. However, there is little empirical evidence supporting this assumption, and there are reasons to suppose that in some cases the bug may have been introduced by other actions, such as an older modification, or a change in the API that is being called.

This paper tries to shed some light on this topic, by analyzing the relationship of bug fixes with their previous commits. To this end, we conducted an observational study on bug reports, their fixes, and their corresponding previous commits for the OpenStack project. Our results show that the assumption that bugs have been introduced in the previous commit does not hold for a large fraction (at least 37%) of the bugs analyzed.

Keywords

Bug introduction, bug seeding, SZZ algorithm, previous commit

1. INTRODUCTION

When a failure is found in the behavior of a software, the developers try to fix it locating and modifying the source code line(s) that are reponsible for the wrong behaviour in the source code. It may semm at first reasonable to assume that previous modifications of this line or these lines are the cause of the bug; the previous modification is what is called *the previous commit*. But in fact, to find when and where a bug was introduced in the source code is not a trivial task, and by far more complex than this assumption. This has largely been ignored in many of the bug-fix literature, mainly because the data related to the origin of a bug is embedded in the evolution of the software [15], meaning that when someone fixes a bug they don't take into consideration the context and the history behind of the project and you cannot have access into this information easily. *Gregorio says: Gema, puedes clarificar la frase anterior? No me ha quedado muy claro que querias decir con lo de embedded in the evolution of the software; quizas habria que poner un ejemplo.Gema says: No se si asi queda mas claro, la idea de embebed la saque del articulo y por lo que entendi con embebed se referia a que no hay ningun log o nada donde los desarrolladores expliquen porque se ha producido un error desde un punto de vista mas historico, que la informacion sobre el error se encuentra embebida en la historia y la evolucion del propio proyecto, no creo que pueda sacar ningun ejemplo*

This is the reason why many studies in the area of mining software repositories start with this implicit assumption. So, for instance, we have found following reasonings in several areas of research, such as:

- in bug seeding studies, e.g., *"This earlier change is the one that caused the later fixed"* [18] or *"The lines affected in the process of fixing a bug are the same one that originated or seeded that bug"* [6],
- in bug fix patterns, e.g., *"The version before the bug fix revision is the bug version"* [13],
- and in tools that prevent for future bugs, e.g., *"We assume that a change/commit is buggy if its modifications has been later altered by a bug-fix commit"* [2].

But although the assumption can be found frequently in the research literature, in our opinion there is not empirical evidence supporting it. That is the reason why we have conducted an observational study on fix-bugs, devoting a significant effort to locate the origin of a bug in the source code and understanding the possible causes. For this we had to take into account the moment in which the line was inserted and the general context of the project.

Figure 1 shows a clear example of what we understand as the cause of the bug. Let's assume that we have three different versions of the same file in the history of the control version of the project.

1. The code on the left (subfigure *(1)*) is the one written to fix the bug.

MSR '16 Austin, Texas USA

DOI: 10.475/123_4

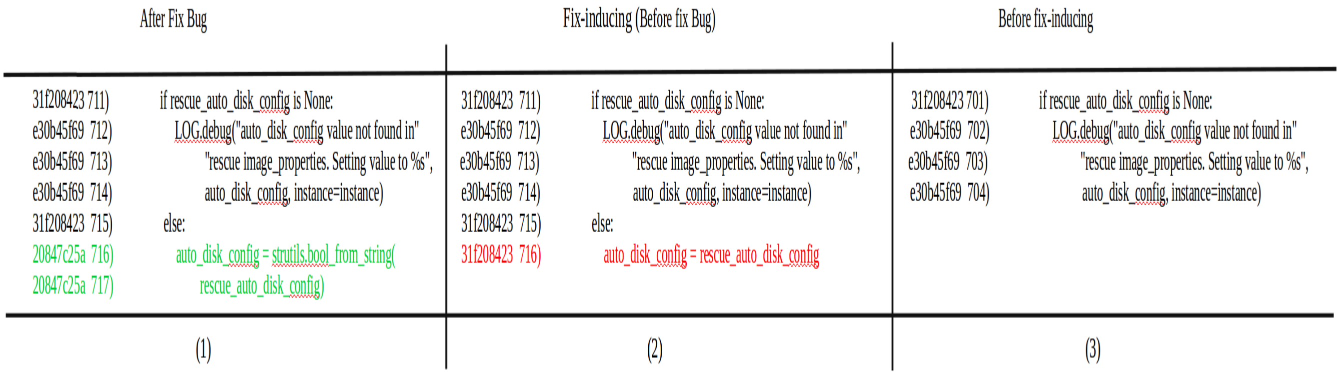
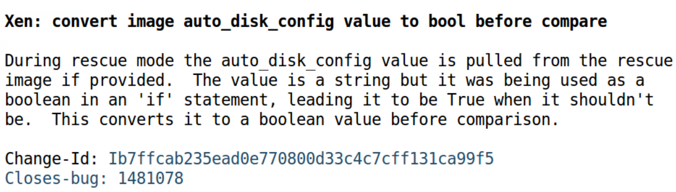


Figure 1: Example of a change in which the bug was introduced in the previous commit. More recent versions of the code are on the left.

2. The code in the middle (subfigure *(2)*) shows the moment in which the bug was introduced (being *31f08423* the id of change), the previous commit.
3. The code on the right (subfigure *(3)*) ensures that in previous versions of the file, the bug did not exist.



```
Xen: convert image auto_disk_config value to bool before compare

During rescue mode the auto_disk_config value is pulled from the rescue
image if provided.  The value is a string but it was being used as a
boolean in an 'if' statement, leading it to be True when it shouldn't
be.  This converts it to a boolean value before comparison.

Change-Id: Ib7ffcab235ead0e770800d33c4c7cff131ca99f5
Closes-bug: 1481078
```

Figure 2: Description of the bug-fix commit for a case in which the previous commit caused the bug.

According to the description in the log of the commit that fixed the bug (see Figure 2), commit *31f08423* was the one where the bug was introduced, as it used a variable as string when a Boolean had to be used, keeping the concordance with the rest of the code. So, in this case, the bug was introduce in the previous commit.

On the other hand, Figure 3 shows a clear example of a case where the cause of the bug is not to be attributed to the previous commit. In this example, the bug fix commit log (see Figure 4) describes that the name of an argument changed when updating the version causing the failure in the software. This change was done because of the new requirements in the software version, and is unrelated to the changes performed in the previous commit. When the modified lines where introduced in the first time, they did not contain the bug.

Based on anecdotal evidence like the one presented in Figures 3 and Figure 4, we argue that in projects that are continuously evolving, with an ample developer community, code that before was correct could be buggy at some time. So, changes in other parts of the code may induce wrong behavior (bugs) in places that were correct in the past. This happens often in situations like changes to the API. In the moment the code ws written, it was correct and the woftware worked fine. Additions of new features or enhancements to the API had as a side effect that the formerly correct code presents a wrong behavior, making the software fail. But in such cases, the source of the error cannot be attributed to a change performed in the previous commit, as in that moment it referred to a different API.

Gregorio says: poner el siguiente parrafo bien Gema says: podemos eliminar el siguiente parrafo porque en realidad lo que querÃ■a decir aquÃ■ es la contribucion del paper y si la pongo luego puedo eliminar este parrafo The goal of this paper is to find if the cause of the bug can be assigned to the previous commit, understanding that at the time when the previous commit is introduced not all times the code analyzed was so easy as the code showed in Figures 1 and 3. So the principal aim of this paper to find the responsible of the bug is to know if the previous commits contained buggy code at the moment in which were added/modified or, another change in the software, related to the update of the software and its evolution, caused the bug.

In detail, in this paper we attempt to address the following research question regarding who introduced the bug in the source code:

Gregorio says: rephrase RQs

- *Gregorio says: RQ1: How many tickets in the issue tracking system are (real) bug reports?*
- *Gema says: Me parece bien la pregunta anterior pero queria focalizarla mas desde el punto de vista del commit previo, para que se adapte mas a la idea del articulo ya que en realidad es algo que se debe hacer pero no tiene mucho que ver con encontrar el causante del bug reports, yo habÃ■a pensado algo como: How we can identify a change done to fix a bug inserted in the code source?*
- RQ2: How often is the previous commit the cause of the bug?

Answering RQ1 is a prerequisite to be able to answer RQ2. This is because (real) bugs are managed together in an issue-tracking tool with feature requests, optimizations, test files, among others. As we are only interested in our research in the actual bugs, we need to identify those as a previous step to analyze if they have been caused by the previous commit.

Gregorio says: Aqui deberiamos poner la contribucion de nuestro articulo al estado del arte. The goal of this paper is to shed some ligths on how and when the bug was inserted into the source code, showing different empirical results from

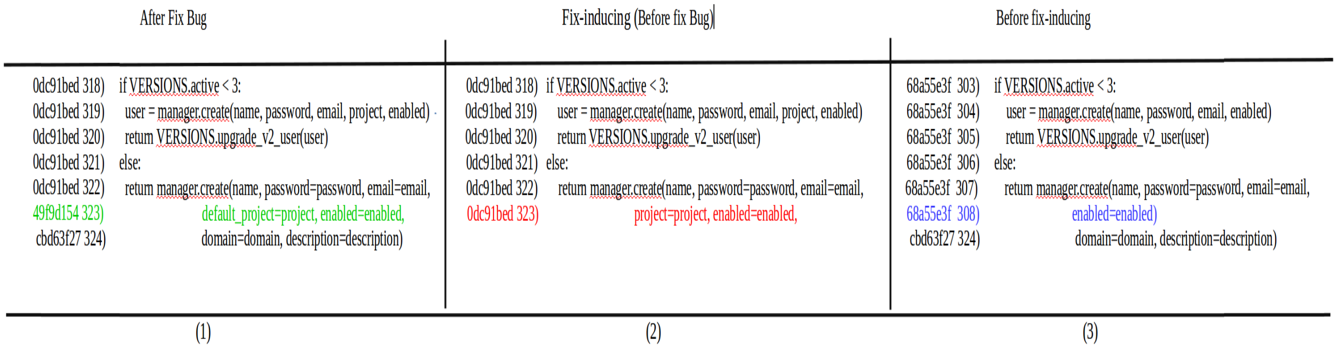
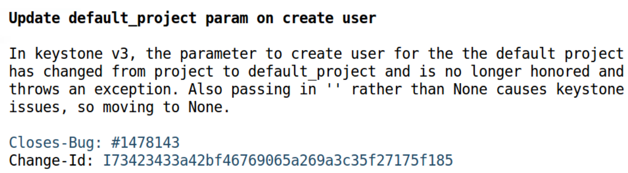


Figure 3: Example of a change where the previous commit, 0dc91bed, did not insert the bug. More recent versions of the code are on the left.



```
Update default_project param on create user

In keystone v3, the parameter to create user for the the default project
has changed from project to default_project and is no longer honored and
throws an exception. Also passing in '' rather than None causes keystone
issues, so moving to None.

Closes-Bug: #1478143
Change-Id: I73423433a42bf46769065a269a3c35f27175f185
```

Figure 4: Description of the bug-fix commit for a case in which the previous commit did not cause the bug.

the the current assumption about the bug was originated in the previous modifications of the lines involved in the fix bug. Thus, our idea is based on the thought that some changes into the code source don't introduce any bug in the software, due to at the moment in which this lines were written, the code wasn't buggy. Finally, our results could lead to think that all the articles based on this assumption are not using the best data in their approach and as a result they can improve their results with our idea. On the other hand, this could be very useful in future researchs, could help developers understanding the bug seeding process.

The remainder of this paper is structured as follows. Next, we present the current body of knowledge in section 2. Section 3 describes the methodology used to identify the moment in which the bug was introduced in the source code, followed by the results obtained after applying our approach to a selection of OpenStack bug fixes in Section 5. Section 6 answers the research questions and discusses potential applications and improvements of our approach. After reporting the limitations and threats to validity in Section 7, we draw some conclusions and point out some potential future work in Section 8.

2. RELATED WORK

The first algorithm to identifying bug-introducing code changes automatically was proposed by Sliwersky et al. [16]. Currently, it is a well-known algorithm called SZZ, which is based on text differences to discover modified, added and deleted lines between the bug-fix and its previous version. The SZZ algorightm uses the CVS `annotate` command[1] to identify the last commit that touched these lines.

An improvement to the SZZ algorithm is described by Kim et al. [8]. There the authors used annotation graphs instead of CVS annotation to locate, in the previous versions, the lines affected by modification and deletion. Also, they avoid some false positives by not considering blank spaces, changes in the format or changes in the comments.

Sinha et al. present another technique to identify the origins of a bug in [15]. Their technique is not text-based technique, as the SZZ algorithm, the authors analyze the effects of bug-fix changes on program dependencies. So, taking into account the semantics of the source code they achieved higher accuracy in identifying the origins of a bug.

The two approaches have some metodological patterns in common:

1. They find the differences between the bug-fix version and the previous version of the file to recognize those changes done by the bug-fix commit.

2. They look back in the code revision history until they identify which version touched the lines affected in the bug-fix for the last time.

Williams et al. revisited the SZZ algorithm to track bug-inducing changes and identify types of changes [18]. Yang et al. applied SZZ to find what kind of bug-inducing changes are likely to become a great threat after being mared as bug-fix changes [20]. Finally, some bug prediction algorithms are based on SZZ; Kim et al. showed how to classify file changes as buggy or clean using change information features and source code terms [7]. *Gregorio says: Creo que deberiamos hablar de mas papers que hacen uso de SZZ y derivados*.

The SZZ algorithm (and its *successors*) have had a considerable impact in the research community. Noteworthy is the fact that the paper with original the SZZ algorithm [16] has been cited, according to Google Scholar, 463 times as of January 2016. An enhanced version of the SZZ algorithm [8] counts with 123 citations.

The SZZ algorithm (and its *successors*) have been widely used in the research community. Williams et al. revisited the SZZ algorithm to track bug-inducing changes and identify types of changes [18]. Yang et al. applied SZZ to find what kind of bug-inducing changes are likely to become a

[1] Other versioning systems provide similar functionality to CVS `anotate`; for instance, git offers `blame`.

great threat after being mared as bug-fix changes Finally, some bug prediction algorithms are based on SZZ; Kim et al. showed how to classify file changes as buggy or clean using change information features and sour code terms [7].

Gregorio says: maybe talk about my paper with dmg and ahmed, where bugs could be found elsewhere [3]. There the talk is precisely about those bugs whose origina are elsewhere. Title of the paper: Change impact graphs: Determining the impact of prior codechanges*Gema says: me parece bien, no lo he leido pero nadie mejor que tu para resumirlo en esta seccion*

In the research literature, we can already find methods that consider other sources of information than the previous commit. In fact, German et al. [3] point out that software is in constant change, and that changes performed may have impact across the whole system and may lead to the manifestation of bugs in unchanged parts. In this case, a bug emerges in a different location from the source of the bug, which is a change to a function somewhere else in the source code base.

3. METHODOLOGY

All data needed to analyze when the bug was introduced can be obtained from the issue tracking systems and the code review systems used generally by free/open source software (FOSS) projects. In our analysis, we have focused on Launchpad[2] as issue tracking system, and Gerrit[3] as code review supporting tool, as they are widely used by FOSS projects nowadays, but our methodology should be generalizable to any such tool.

The Launchpad of each project works with issue reports called tickets, which describe bug reports, feature requests, maintenance tickets, and even design discussions. In our study, however, we are only interested in those tickets that have following properties:

1. They describe a bug report, and
2. They have been closed and merged in the code source to fix the described bug.

In these bug reports we can find a comment with the link to Gerrit where the bug was fixed. It is in Gerrit where we can see all the patchsets proposed and the comments done by the reviewers.

3.1 Fist Stage: Filtering

First, we have to identify what issues found in Launchpad are bug reports. This is not a trivial task and is labour intensive as it has to be done manually. As the process is repetitive, we developed a web-based tool[4] that helps in the classification process. This tool offers all relevant information required to decide if an issue corresponds to a bug report or not. The tool uses information extracted automatically from the project repositories, and offers a web-based interface which allows for collaboration, traceability and transparency in the identification of bug reports.

During the identification of the issues, we have to take into account the next parameters for each ticket:

- The title of the issue report
- The description of the issue report
- The description of the fix commit
- The changes to the source code, as sometimes neither the descriptions nor the comments by developers and reviewers in the Launchpad and Gerrit of each ticket, clarified the underlying ticket.

We can see a screenshot of the web interface of the tool in Figure 5. The left side is used to display the information extracted from Launchpad and Gerrit, and the right part is the one in which the researchers can write and classify the ticket into one of the three groups. Additional meta-data, such as keywords, comments and the reviewer are included in the database.

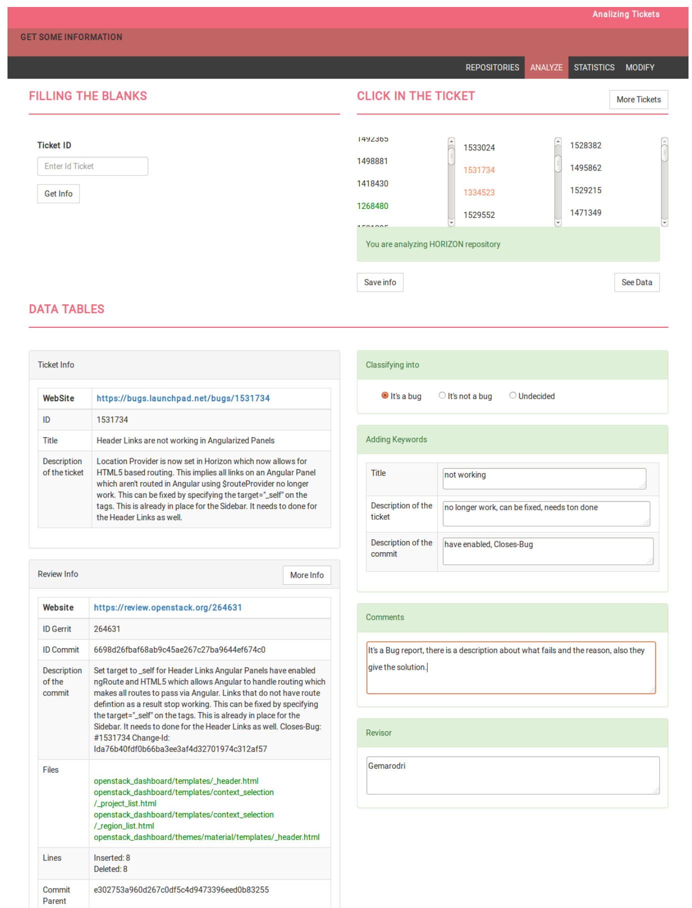


Figure 5: Screenshot of the tool used to classify the tickets.

Each ticket was then categorized into one of three following groups:

1. Group 1 (*Bug Report*): The ticket describes a bug report.
2. Group 2 (*Not Bug Report*): The ticket describes a feature, an optimization code, changes in test files or other not bug reports.

[2] https://launchpad.net/
[3] https://www.gerritcodereview.com/
[4] bugtracking.libresoft.es

3. Group 3 (*Undecided*): The ticket presents a vague description and cannot be classified without doubts.

From the experience of analysing a small number of tickets, we agreed on following four criteria:

1. Each time that the title or the description of a ticket describes an unexpected behaviour in the program, our criteria indicated that it was considered as a bug report.

2. The description of the ticket presents an optimization, deletion of a dead code or the implementation of new characteristics, we agreed not to classify it as a bug report because there is no failure.

3. When the ticket described that some updates were required, the ticket is a bug report. We consider all tickets that require updating as bug reports, because updating a software hints to the software not operating as expected.

4. When only test files are affected in a ticket, we classified it as not being a bug report. We consider bug errors in test files are a different type of bugs, as the software may still work as expected. *Gregorio says: no se si esto es lo que querias decir* *Gema says: si, es eso lo que queria decir*

Sometimes we were unable to answer all the questions due to having insufficient data or because of the complexity of the issue. In this case, the ticket was classified into the *Undecided* group.

3.2 Second Stage: Who caused the Bug?

In this second part, our work was focused on analyzing the previous commit exclusively for those tickets classified in the *Bug Report* group. Therefore we had to locate the line that contained the bug, inquire the reason of the software failure, and gathering additional information on the context of the project.

For that, we had to analyze the lines involved in the bug fix and in the *parent* commit of the bug fix commit, being sure that the lines were added,inserted or modified in the previous commit, we refered to *parent* commit as the last commit that was edited to produce the subsequent state of the file. We do this process to be sure that we are looking the correct change, because some times although the commit added many lines, if you look the code before the commit you can check that some of the lines added was there, and in that case, is a false positive where the previous commit didn't cause the bug. *Gregorio says: Gema, puedes aclarar este parrafo? Que es el "parent" commit? Es lo mismo que el previous commit? Es la primera vez que hablamos de el. Quizas sea buena idea que lo reformules en castellano, porque ahora mismo es criptico* *Gema says: el parent commit es el ultimo commit que se hizo en el fichero antes de que se realizase el que arreglo el error, por tanto lo que hago usar diff y blame en el commit parent y en el commit que arregla el error para saber que lineas se shan cambiado y que commit las cambio), y despues hago lo mismo con el commit previo y el commit parent del pervio para saber que en realidad en ese momento se produjo el cambio. En el gerrit nos dan esa info como se puede ver en* **??**

The analysis was done manually. We used *git blame* to see the previous commit for each line of the involved file. Also,

Author	Andrew Laski <andrew.laski@rackspace.com>	Aug 3, 2015 10:29 PM
Committer	Andrew Laski <andrew.laski@rackspace.com>	Aug 3, 2015 10:29 PM
Commit	20847c25a8157a10b765387ff8dbda31f8f4e91a (gitweb)	
Parent(s)	db7fc595ebc86b19ead193a3571e4db2ba8de8f5 (gitweb)	
Change-Id	Ib7ffcab235ead0e770800d33c4c7cff131ca99f5	

we used *diff* to see the differences between the two files, in our case as the file is going to be the same, between the file in two different moments in the control version system.

The procedure for each file involved in a bug fix is as follows:

1. git checkout *commit that fixed the bug*, git blame *file involved*. In this step we can see the lines added, modified or deleted by the commit that fixed the bug.

2. git checkout *parent of commit that fix the bug*, git blame *file involved*. In this step we can see the previous commits for the different lines touched in the fixed bug.

3. git checkout *parent of previous commit*, git blame *file involved*. With this step we can ensure that the previous commit inserted these lines.

Finally we need to discard some noise presents in our final results according to the responsibility of the previous commit inserting the bug in the code source. Due to they were not responsible for cause the bug, we delete those previous commit which presents the following criteria:

- Blank lines
- Format changes
- Copied lines
- Changes in the comment.
- Updates in the version of a file.

4. EVALUATION

We have validated our methodology analyzing tickets from OpenStack. OpenStack is a cloud computing platform with a huge developing community (more than 5,000 developers) and significant industrial support from several major companies such as Red Hat, Intel, IBM, HP, etc. OpenStack was particularly of interest because of its continuously evolving due to its very active community. Currently it has more than 233,000 commits with more than 2 million lines of code [5]. All its history is saved and available in a version control system, as well as its issue tracking system (Launchpad[6]) and the source code review system (Gerrit[7]).

OpenStack is composed by 9 projects, but we only focused on the main four of them: Nova, Cinder, Neutron and Horizon. As can be seen from Table 1, these projects have been very active during their entier history, and in the last year.

For these four projects we analyzed if bug fixes where introduce in their previous commits. For the first stage, we used the tool described in 3.1. Each ticket was analyzed by two researchers independently. The second stage was done manually by the first author.

[5]http://activity.openstack.org/dash/browser/
[6]https://launchpad.net/openstack
[7]https://review.openstack.org/

	All History	Last Year (2015)
Nova	14,558	3,283
Fuel	9,139	5,123
Neutron	8,452	3,855
Horizon	4,871	1,994
Cinder	4,556	1,832
Keystone	4,874	1,795
Heat	6,395	2,372
Glance	2,651	723
Tempest	4,141	1,312

Table 1: Commits per Project in OpenStack

5. RESULTS

A total of 459 different tickets from the Launchpad of the four main projects in OpenStack: 125 tickets from Nova, 125 tickets from cinder, 125 tickets from Horizon and 84 tickets from Neutron.

5.1 Fist Stage

We classify a total of 459 tickets using the tool, resulting in 917 reviews*Gregorio says: Mirando la Tabla 2, veo que las revisiones suman 917, no 918!*.*Gema says: Es cierto, son 917* Only those tickets classified as bug reports by both researchers were considered in the next stage, which analyzes if the cause of the bug was introduced in their previous commits.

Table 2 shows the classification percentages for each researcher after analyzing the tickets, and the number of tickets classified by two different researchers in the same group. As a result, researchers identified 292 tickets in the same group, that is, their results matched in over 70% of the cases. Of those, 209 tickets had been classified in the *Bug report* group, 74 in the *Not Bug Report* group and 9 tickets classified in the *Undecided* group.

We also measured the concordance in the classification of each developer according to the project analyzed (see table 3). Values obtained by the three researchers are very similar, in general around a 70%. The concordance values were always above 60%.

After this, we can answer the first research question because at this moment we have all the data necessary and all the knowleadge to can distinguish bug reports from others reports.

Gregorio says: Tiene sentido anadir numeros absolutos, como he hecho a la columna de Total en la Tabla 3? Gema says: Si, tiene sentido

> **RQ1: Using all the information available in the bug tracking system and code review systems related to a fix-bug, we have obtained that in at least 72% of the tickets analyzed the fix-bugs were real bug reports.**

Gregorio says: Molaria hablar de cuanto tiempo ha llevado realizar esta etapa de clasificacion por cada uno de los investigadores (o sea, sin contar el desarrollo de herramienta y conseguir y cargar los datos de bugs y demas en la base de datos) Gema says: El problema es que no hemos medido el tiempo, he hablado con ellos y me dicen que de media de 5 a 10 minutos por ticket, a mi me llevo unos 5 minutos de media, por que ya estaba mas acostumbrada, pero de todas formas se podrÃ■a medir en github en tiempo que les ha llevado a cada uno entre ticket y ticket, en aquellos casos que se puedan medir y hacer una media, Que te parece?

5.2 Second Stage

In this stage we analyzed the 189 tickets classified *Gregorio says: Pero no eran 209? Gema says: deberian ser 209 pero me han quedado 20 bugs report por analizar, si tengo tiempo cuando llegue a casa lo hago y anado los numeros* as bug reports and their previous commits. The possible outcome of the analysis was one of the following three options:

- Cause
- No Cause
- Undecided

This analysis takes into account that the bug could have been inserted in many lines that belong to many different previous commits, but in fact, not all of them could cause the bug. It may happen that in the previous commit, lines have been copied from further previous commits, comments may have been modified or blank spaces/lines have been introduced. Thus, the cause could be found in a single previous commit, in many or even in none. *Gregorio says: Todavia no esta del todo claro... yo creo que esto merece un ejemplo - incluso con codigo real, tal y como hemos hecho al principio del articulo. Ademas, nos lo podemos permitir; tenemos espacio.Gema says: perfecto, ahora busco un ejemplo y lo anado abajo*

We identified a total of 348 previous commits which could be the cause of the 189 bug reports under analysis. Then, we analyzed the bug reports together with their previous commits discarting the cases where the previous commit was a false positive (blank lines, changes in comments or even a change in the version of the file). So, in total we have analyzed 308 previous commit.

As can be seen in Table 4, from the 308 previous commits, 152 have been considered to be the cause of the bug, whereas 114 were identified as not beign the cause. We were unable to decide in 42 cases, as only lines had been added or due to our limited knowleadge about the code. *Gregorio says: Gema, puedes poner un ejemplo de estas dos cosas? Uno de solo added lines y otro de limited knowledge. Si el ejemplo es verdadero, mejor.Gema says: perfecto, ahora busco un ejemplo y lo anado abajo* We discarded 40 more because they were false positives (*noise*) such as blank lines, changes in comments or even a change in the version of the file.

If we attend to how many previous commits each of the 189 bug reports analyzed had, we see that 131 only had a previous commit, whereas 58 had more than one previous commit. As can be seen from Table 5, from the 131 unique previous commits, 65 were the cause of the bug, and 30 not caused the failure. For the 58 bugs that had more than one previous commit, a total number of 179 previous commits were identified; of them 86 were the cause of the bug, while 82 were not. *Gregorio says: Pon ejemplos, a ser posible reales, de esto, porfa.Gema says: perfecto, ahora busco un ejemplo y lo anado abajo*

Gregorio says: No me queda claro que quieres decir con mas de un identificador previoGema says: supongo que despues de ver el ejemplo se entienda mejor, identificador previo se refiere a que al arreglar el error se shan tocado varias

	Bug Report	Not Bug Report	Undecided	Total
R1	(184) 55%	(115) 34%	(35) 11%	334 (100%)
R2	(188) 76%	(54) 22%	(7) 3%	249 (100%)
R3	(188) 56%	(116) 35%	(30) 9%	334 (100%)
Agree	(209) 72%	(74) 25%	(9) 3%	292 (100%)

Table 2: Statistics for each researcher as a result of the classification process. For each researcher R, the number of tickets (and percentages) classified into the three groups is given. The *Agree* row gives the number of tickets (and percentages) where two researchers agreed.

	Nova	Cinder	Horizon	Neutron	Total
R1 – R2	(44) 70%	(40) 77%	(37) 60%	-	(121) 68%
R1 – R3	-	(46) 73%	(48) 76%	(26) 62%	(120) 71%
R2 – R3	(41) 66%	(10) 100%	-	-	(51) 71%

Table 3: Concordance among researchers for each repository.

	Before Deleting Noise	After Deleting Noise
Cause	(152) 44%	(152) 49%
Not Cause	(154) 44%	(114) 37%
Undecided	(42) 12%	(42) 14%

Table 4: Number of times (and percentage) where the previous commit is the cause, not the cause or could not be classified, before and after deleting noise.

lineas que han sido anadidas o modificadas en commit diferentes, por tanto tenemos mas de un commit diferente, aqui use identificador en vez de commit y puede ser un error

Gregorio says: Para los 58 bugs que tenian mas de un identificador previo hay 179 commits. Bien. Dices que 86 causaron el bug, 82 no y en 11 no se ha podido determinar. Lo que no veo es por que no agruampos estos datos segun los 58 bugs Gema says: Esos datos estan agrupados en la tabla 7, el fin de esta tabla era mostrar que cuando hay un solo comit previo el numero de no causantes del error es menor que cuando hay mas de un commit previo

	One previous commit	More than one previous commit
Cause	(65) 50%	(86) 48%
Not cause	(30) 23%	(82) 46%
Undecided	(36) 27%	(11) 6%

Table 5: Probability of the cause of the bug when the bug report present one previous commit or more than one previous commit.

We also studied the distribution of the number of previous commits for each bug. As shown in Table 6, the number of commits that can be considered as previous is 1, followed by 2 commits. *Gregorio says: the number of commits that can be considered as previous is 1, followed by 2 commits. No se que queremos decir Gregorio says: Tenemos que responder a estas dos preguntas todavia: Por que es interesante hacer lo que se comenta en este parrafo? Que utilidad tiene? Gema says: Creo que era interesante saber como se distribuyen los causantes de los bugs en cada projecto y cuantos commits previos presentan los bugs report en cada projecto analizado, ademas cuantos mas commits previos, en ocasiones, es muy dificil saber quien es el responsable por tanto pense que podria haber algun patron en relacion con el numero de commits previos y cuantos de ellos son responsables, pero parece que esto no ocurre con la muestra que hemos analizado.*

Finally, we were interested in analyzing, for those cases where more than one previous commit exist, how many of them introduced the bug in the code source. Even if several previous commits are involved, it may be the case that none, only one or all of them is the cause of the bug. *Gregorio says: Cuando hay 3 previous commits, no podria darse el caso de 2 de ellos responsables? Gema says: si, eso sucede y eso casos se encuentran dentro de al menos un responsable, (at least one responsible)* Results are given in Table 7; in 8 bug reports all the previous commits were identified as the cause, in 30 bug reports at least one of the previous commits caused the bug, and in 11 bug reports none of the previous commits introduced the bug. If we look at bugs that had two previous commits, in 4 cases both commits were the cause, in 9 cases only one of them was the cause and in another 4 cases non of them could be determined as the cause.

Gregorio says: Entiendo el resultado del parrafo anterior, pero me falta una explicacion de por que esto es interesante y para que puede servir. Gema says: Para tener una idea de como se distribuyen los causantes del error, podrÃ■a eliminar las tablas si en realidad no las encuentras funcionales y no aportan informacion.

RQ2: Only 50% of the previous commits analyzed caused the failure in the system, whereas the 37% of them did not introduce the bug in the code source.

6. DISCUSSION

The experience gained in this study with exposure to several hundreds of bugs allows us to state that determining who (or what) introduced a bug is a non-trivial task.

Although at first, as shown in the Figure 1 and Figure 2, one may think that this is an easy task, we have found many examples where we have been unable to determine the cause

	One previous commit	two previous commit	three previous commit	four previous commit	+five previous commit
Neutron	11	3	2	2	0
Horizon	39	8	3	2	4
Nova	44	5	2	4	4
Cinder	37	9	6	2	2
Total	131	25	13	10	10

Table 6: Distribution of number of previous commit per Bug Report in each project

	two previous commit	three previous commit	four previous commit	+five previous commit	Total
All Responsible	4	3	0	1	8
At least one responsible	9	7	5	9	30
None Responsible	4	2	4	1	11
Undecided	1	0	1	0	2

Table 7: Number of previous commit responsibles per bug report

as no previous commit can be identified. This is, for instance, the case when only code has been added and there is no way to identify the previous commit. In this case, further research could find out if this is not really the addition of a new feature rather than a bug.

And we are talking about a bug report not a new feature, these kinds of cases use to be when a researcher forgot check some case inside a function.

1. Is responsible the function where these lines are content?

2. Is responsible the last commit that modify something in the function?

Gema says: Puedes elaborar un poco mas el parrafo anterior? Puede ser en castellano

There are other cases where it is difficult to determine if the change is the cause. For instance, when additional conditions are added to *if*s or *else*s, one might think that in the previous commit those were not included due to an error, so that the previous commit becomes the cause of the bug. However, situations exist where the additional conditions in an if appear because of the introduction of a new functionality, and thus the line in the previous commit was correct at the time it was introduced. In our analysis, if the latter situation is not explicity mentioned we have considered that the previous commit caused the error.

But already the identification of an issue as a bug report is a process that is not as straightforward as one might think. Out of 459 tickets we were only capable to achieve a consensus for 292 cases (63.6%), which hints to the complexity of the task. The amount of information, the number of fields and the requirement of human interaction made us invest time in the creation of a tool that assisted us in the process.

In any case, our research shows evidence that assuming that the previous commit is where the cause of a bug can be found does not hold for a not insignificant percentage of bugs. *Gregorio says: Tengo que elaborar mas esto, que es el punto fuerte del articulo.*

7. THREATS TO VALIDITY

As any other empirical study, this one presents several threats to its validity, external and internal, that have to be considered and taken into account. In order to allow others to study it in detail, replicate it or even build on top of it, we have set up a replication package including data sources, intermediate data and scripts that can be obtained at the following URL: *Gema says: Incluir la URL en la que pondre todos los datos y scripts utilizados para los datos.*

The number of the tickets extracted form Launchpad is high, but probably not as high as to state that it can be representative of all free/open source systems, or the software industry. It has not to be forgotten that the analysis requires a lot of human effort, so that achieving large numbers of cases is difficult.

The internal threats related to the researchers that have conducted the study are following:

- We have not considered those tickets where the two researchers showed discordance.
    1. Should they discuss after their analysis to reach a better classification?, Should the tool provide this?
    2. Does the Bug report only the same ticket classified as Bug report for all the researchers?
- We have not taken into account errors that have been classified into *Undecided*, and probably we have lost some *actual* bug reports.
- There could be some lax criteria involving the subjective opinion of the researchers.
- Although the researchers are experienced programmers, they are not experts in the OpenStack project, and their inexperience may have influenced the results of the analysis.
- We are only using part of the information that the tickets provide, like comments and text. There could

be a recognized pattern in the data, unknown at first sight, that involves other parts of the information.

- We have used a random script to extract the tickets from Launchpad that have been reported during 2015. There could be unintended bias of the data, because many reasons, as for instnace the phase of the project.
- In some cases, researchers may have classified the previous commit as the cause of the bug, even if this may not be the case (see discussion on additional conditions in if statements).

The external threats, related to the case of the project, are following:

- The word *bug* is continuously mentioned in the description and commit of a ticket even when we found it is not an error. This could lead to the incorrect classification during the reviewing process.
- Some tickets are not explicitly described, which could increase the percentage of *Undecided*. This is especially true if the reviewers are not from OpenStack.
- OpenStack is a special project with a constant evolution due to their active community of developers. Maybe, in other projects with less commits per year, results may be totally different.

8. CONCLUSIONS AND FUTURE WORK

The empirical experiment carried out in OpenStack supported that the current premise assumed does not hold for a large fraction of the analyzed bugs, because around the 40% of the previous commits were not the cause of the bug.

With our results we can identify which ones are real changes that introduced the bug, and this could be useful to improve the accuracy of those tools developed to prevent bugs. Also, the software developers stand to benefit from identifying where the bug was inserted, improving their methodology.

A final field in our future work concerns the full automation of the methodology could developer an automatic classifier base on the idea that not all the previous commit injected the bug. Another interesting investigation could perform the same empirical study in a project with a community less active, to can prove if our idea is fulfill in other projects. *Gregorio says: Puedes aclarar esto, porfa? En castellano si quieres Gema says: Si, la idea es realizar el mismo estudio pero con otro projecto que no sea OpenStack, porque en realidad OpenStack esta evolucionando continuamente. Y para poder demostrar que en realidad la idea se cumple en todos los projectos, deberiamos analizar un projecto que sea mas 'estable' y en el que la API no se este cambiando continuamente*

9. ACKNOWLEDGMENTS

We thank Dorealda Dalipaj and Nelson Sekitoleko, two PhD students in our research team, that participated in the process of classifying bug reports. We also want to express our gratitude to Bitergia[8] for the OpenStack database and the support they have provided when questions have arised. Finally, we would like to acknowledge the Spanish Government because all authors are funded in part by it, through project TIN2014-59400-R.

[8] http://bitergia.com/

10. REFERENCES

[1] A. Bachmann, C. Bird, F. Rahman, P. Devanbu, and A. Bernstein. The missing links: bugs and bug-fix commits. In *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*, pages 97–106. ACM, 2010.

[2] M. Fejzer, M. Wojtyna, M. Burzańska, P. Wiśniewski, and K. Stencel. Supporting code review by automatic detection of potentially buggy changes. In *Beyond Databases, Architectures and Structures*, pages 473–482. Springer, 2015.

[3] D. M. German, A. E. Hassan, and G. Robles. Change impact graphs: Determining the impact of prior codechanges. *Information and Software Technology*, 51(10):1394–1408, 2009.

[4] K. Herzig, S. Just, and A. Zeller. It's not a bug, it's a feature: how misclassification impacts bug prediction. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 392–401. IEEE Press, 2013.

[5] A. Hindle, D. M. German, and R. 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.

[6] D. Izquierdo-Cortazar, A. Capiluppi, and J. M. Gonzalez-Barahona. Are developers fixing their own bugs?: Tracing bug-fixing and bug-seeding committers. *International Journal of Open Source Software and Processes (IJOSSP)*, 3(2):23–42, 2011.

[7] S. Kim, E. J. Whitehead Jr, and Y. Zhang. Classifying software changes: Clean or buggy? *Software Engineering, IEEE Transactions on*, 34(2):181–196, 2008.

[8] S. Kim, T. Zimmermann, K. Pan, and E. J. 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.

[9] S. Koch. *Free/open source software development*. Igi Global, 2005.

[10] D. MacKenzie, P. Eggert, and R. Stallman. *Comparing and Merging Files with GNU diff and patch*. Network Theory Ltd., 2003.

[11] E. W. Myers. Ano (nd) difference algorithm and its variations. *Algorithmica*, 1(1-4):251–266, 1986.

[12] A. T. Nguyen, T. T. Nguyen, H. A. Nguyen, and T. N. Nguyen. Multi-layered approach for recovering links between bug reports and fixes. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, page 63. ACM, 2012.

[13] K. Pan, S. Kim, and E. J. Whitehead Jr. Toward an understanding of bug fix patterns. *Empirical Software Engineering*, 14(3):286–315, 2009.

[14] L. Prechelt and A. Pepper. Why software repositories are not used for defect-insertion circumstance analysis more often: A case study. *Information and Software Technology*, 56(10):1377–1389, 2014.

[15] V. S. Sinha, S. Sinha, and S. Rao. Buginnings: identifying the origins of a bug. In *Proceedings of the 3rd India software engineering conference*, pages 3–12.

ACM, 2010.

[16] J. Śliwerski, T. Zimmermann, and A. Zeller. When do changes induce fixes? *ACM sigsoft software engineering notes*, 30(4):1–5, 2005.

[17] E. Ukkonen. Algorithms for approximate string matching. *Information and control*, 64(1):100–118, 1985.

[18] C. Williams and J. Spacco. Szz revisited: verifying when changes induce fixes. In *Proceedings of the 2008 workshop on Defects in large software systems*, pages 32–36. ACM, 2008.

[19] R. Wu, H. Zhang, S. Kim, and S.-C. Cheung. Relink: recovering links between bugs and changes. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, pages 15–25. ACM, 2011.

[20] H. Yang, C. Wang, Q. Shi, Y. Feng, and Z. Chen. Bug inducing analysis to prevent fault prone bug fixes. In *Proceedings of the Twenty-Sixth International Conference on Software Engineering and Knowledge Engineering (SEKE 2014)*, pages 620–625, 2014.

[21] Z. Yin, D. Yuan, Y. Zhou, S. Pasupathy, and L. 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.

[22] T. Zimmermann, S. Kim, A. Zeller, and E. J. 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.

[23] T. Zimmermann, A. Zeller, P. Weissgerber, and S. Diehl. Mining version histories to guide software changes. *Software Engineering, IEEE Transactions on*, 31(6):429–445, 2005.