Empirical Software Engineering (2022) 27: 134
https://doi.org/10.1007/s10664-022-10173-y

Upstream bug management in Linux distributions

An empirical study of Debian and Fedora practices

Jiahuei Lin[1] · Haoxiang Zhang[2] · Bram Adams[1] · Ahmed E. Hassan[1]

Accepted: 27 April 2022 / Published online: 20 July 2022

Abstract

A Linux distribution consists of thousands of packages that are either developed by in-house developers (in-house packages) or by external projects (upstream packages). Leveraging upstream packages speeds up development and improves productivity, yet bugs might slip through into the packaged code and end up propagating into downstream Linux distributions. Maintainers, who integrate upstream projects into their distribution, typically lack the expertise of the upstream projects. Hence, they could try either to propagate the bug report upstream and wait for a fix, or fix the bug locally and maintain the fix until it is incorporated upstream. Both of these outcomes come at a cost, yet, to the best of our knowledge, no prior work has conducted an in-depth analysis of upstream bug management in the Linux ecosystem. Hence, this paper empirically studies how high-severity bugs are fixed in upstream packages for two Linux distributions, i.e., Debian and Fedora. Our results show that 13.9% of the upstream package bugs are explicitly reported being fixed by upstream, and 13.3% being fixed by the distribution, while the vast majority of bugs do not have explicit information about this in Debian. When focusing on the 27.2% with explicit information, our results also indicate that upstream fixed bugs make users wait for a longer time to get fixes and require more additional information compared to fixing upstream bugs locally by the distribution. Finally, we observe that the number of bug comment links to reference information (e.g., design docs, bug reports) of the distribution itself and the similarity score between upstream and distribution bug reports are important factors for the likelihood of a bug being fixed upstream. Our findings strengthen the need for traceability tools on bug fixes of upstream packages between upstream and distributions in order to find upstream fixes easier and lower the cost of upstream bug management locally.

Keywords Software ecosystems · Open source collaboration · Linux upstream package management · Upstream bug fixing

Communicated by: Walid Maalej

✉ Haoxiang Zhang
haoxiang.zhang@acm.org

Extended author information available on the last page of the article.

1 Introduction

A software ecosystem consists of many software projects with relationships among them and a shared platform (Van Den Berk et al. 2010). Linux distributions (e.g., Debian) are one of the most well-known software ecosystems. In such distributions, software projects developed internally (in the distribution) or externally (upstream, potentially customized in the distribution) are integrated with each other into a coherent product. To achieve this, maintainers of the distribution package up all projects in a standard format using the distribution's package manager, recording any metadata such as dependencies, licenses, etc. The package manager is able to enforce the metadata for the end users, guaranteeing that each installed package has all the dependencies it needs.

Being the middlemen between the upstream projects and the end users, the distribution maintainers spend a large amount of effort and resources maintaining upstream packages, for example, updating to a new version, applying and reporting bug fixes, etc (Adams et al. 2016; Anbalagan and Vouk 2009; Storey et al. 2016). Prior studies investigated the integration challenges of upstream packages, such as estimating the effort to select the proper packages or ensuring the security of the selected packages (Stol et al. 2011; Li et al. 2005; Merilinna and Matinlassi 2006). Prior studies also surveyed the practices of fixing bugs that are introduced by such integration (Ma et al. 2017; Ding et al. 2017) and the associated effect to developers during integration (Ma et al. 2020). However, prior studies did not investigate (1) to what extent bug fixing effort is spread across upstream developers and distribution maintainers, and (2) the cost going into upstream package management for distribution maintainers.

One effort to reduce the distribution maintainers' required maintenance effort of upstream packages (https://www.freedesktop.org/wiki/Distributions/Packaging/WhyUpstream/) are freedesktop.org's *distribution-neutral packaging guidelines*. These guidelines suggest that distribution maintainers should follow the development practices of upstream projects, e.g., sending changes to upstream. Nevertheless, the guidelines list some exceptions for distribution maintainers to make a local fix, e.g., security issues, unresponsive upstream developers, or discontinued projects.

In such exceptional cases, distribution maintainers need to step up and develop their own patches, i.e., bypassing the upstream bugs, to protect their users from potential risks of upstream code (Ma et al. 2017). However, this practice is not straightforward since distribution maintainers are not familiar with the upstream projects to propose the most appropriate fix, or they could even introduce new bugs in such patches (Hauge et al. 2010; Li et al. 2005; Merilinna and Matinlassi 2006). Furthermore, while distribution maintainers can create patches, they also need to maintain those patches and integrate them with the subsequent releases of the upstream package, until it incorporates the fixes (Ma et al. 2017). Even then, the typical lack of documentation for changes in upstream releases hinders the ability of distribution maintainers to understand which upstream release incorporates their needed fixes (Stol et al. 2011; Ma et al. 2017).

In this paper, we measure how upstream packages in a distribution are managed in terms of fixing bugs and compare the costs of locally fixing bugs of upstream packages by distribution maintainers to the costs of upstream fixes to such bugs, as well as locally fixing bugs of in-house packages (i.e., the packages developed by the distribution itself). We investigate 143,362 high-severity (i.e., high, urgent, critical) bugs[1] across two popular Linux

[1]Unless noted otherwise, the remainder of this paper refers to high-severity bugs as "bugs".

distributions, i.e., Debian and Fedora, that use two different types of package managers and different bug management strategies. Debian is a purely open-source distribution, while Fedora forms the core of the commercial Red Hat distribution. We focus on high-severity bugs since they typically represent fatal errors or crashes that are important for developers to fix (Menzies and Marcus 2008; Lamkanfi et al. 2010). We structure our study by answering the following research questions, yielding the following key findings:

RQ1: ***What percentage of upstream bugs are fixed upstream versus locally?***

This first RQ analyzes the prevalence of high-severity bugs in upstream vs. in-house packages. High-severity bugs are dominant in the upstream packages of the studied distributions, yet for 27% of them we found explicit traces of where they were fixed (14% fixed upstream, 13% fixed in the distribution), while for the remaining 73% it was unclear where they were fixed in Debian. The proportion of high-severity bugs per upstream package that are fixed by upstream has dropped over time in Fedora and has been fluctuating in Debian. Similarly, the proportion of opened bugs per upstream package has also dropped over time in the studied distributions. Moreover, Debian and Fedora share 9 out of the top 10 categories of buggy upstream packages.

RQ2: ***What is the time and cost for fixing high-severity upstream bugs locally in distributions and maintaining such distribution fixes?***

We observe that bugs in upstream packages take 52% and 49% more time to be fixed locally in Debian and Fedora, respectively, compared to bugs in in-house packages, but 39% less time than upstream fixed bugs in Debian. Moreover, in the bug comment discussions, maintainers mention more links to reference documents of a distribution (e.g., design docs, bug reports) when fixing upstream bugs locally than when fixing bugs in in-house packages in Debian, but fewer links than upstream bugs that are fixed upstream. In terms of the number of participants per day and the number of comments per day, we observe that the cost of fixing upstream bugs, both upstream and locally, is similar to fixing bugs in in-house packages. In addition, distribution maintainers maintain fewer local patches (fixes) per upstream package within one distribution release than developers do for in-house packages.

RQ3: ***Which high-severity bugs are most likely to be fixed upstream?***

Since fixing upstream bugs locally requires a longer time than fixing in-house bugs, this RQ investigates the factors that are associated with a bug being fixed by upstream developers, by leveraging logistic regression models. In particular, we extract 27 features in 6 dimensions to estimate the association between these features and the likelihood of a bug being fixed upstream. Our models achieve a median AUC of 0.87 and 0.73 for Fedora and Debian, respectively. For the performance across the studied distributions, the model trained on Fedora achieves an AUC of 0.63 for Debian and the model trained on Debian achieves an AUC of 0.81 for Fedora. Our results show that the features related to bug history (e.g., reported bugs in upstream) are correlated the most with a bug being fixed upstream. Moreover, the more similar to a previously reported upstream bug, the more likely a bug is fixed upstream.

Since only the minority of upstream bugs are explicitly recorded to be fixed upstream or locally, distributions and upstream projects should establish a process to explicitly track the fixing process of bugs in upstream packages. In addition, even though the cost of fixing a bug in upstream packages locally at the fine-grained level (i.e., in terms of the number of

participants per day and comments per day) is similar to in-house bugs, users wait a longer time to get fixes and maintainers need more information for fixing upstream bugs. Based on our models, the features in the bug history dimension have a relatively large explanatory power regarding the likelihood of a bug being fixed upstream. We suggest that maintainers should check upstream projects or peer-distributions for similar bug reports, especially those with potential fixes, beyond the complex relation between upstream projects and distributions that carry them.

Paper organization. Section 2 provides the background of upstream package management in Linux and prior related work to our study. Section 3 describes how we design our study and obtain our studied dataset. Section 4 answers our research questions. Section 5 discusses the implications of our study. Section 6 discusses the threats to the validity of our study. Finally, Section 7 concludes the paper.

2 Background and Related Work

In this section, we briefly introduce upstream package management in Linux distributions and how to fix bugs in upstream packages. We also discuss prior studies that are related to our study. We discuss related work on empirical studies of (1) upstream and downstream coordination and (2) bug fixing cost.

2.1 Upstream Package Management in Linux Distributions

A Linux distribution contains the kernel, a set of packages, and a package management system. The package manager facilitates the installation of software from scratch by searching the available packages in specific repositories, while ensuring that specified version constraints are followed. As shown in Fig. 1, upstream developers are members of upstream projects in charge of developing those projects. The integration (and customization) of such upstream projects into upstream packages is done by (distribution) maintainers. While the role of "maintainer" is specific to a distribution, the person having that role could be part of the upstream project (e.g., Fedora) or external to it (e.g., Debian). Finally, distributions also feature in-house packages developed by (in-house) developers.

Given that distributions distribute both upstream and in-house packages to their user base, they also play a major role in the management of bugs reported for those packages. While in-house package bugs obviously are the responsibility of distributions themselves, upstream bugs should be reported to and resolved by the upstream projects in order to propagate the fixes to downstream distributions. Freedesktop.org, a website that offers interoperability specifications on the software development management (e.g., packager policy, naming conflicts between files in two upstream packages) for Linux distributions, offers an official guideline on how to manage upstream package bugs in Linux distributions (https://www.freedesktop.org/wiki/Distributions/Packaging/WhyUpstream/), encouraging developers not to deviate from the software development practices of the upstream projects. The guideline highlights the benefits and shortcomings of upstream package management, such as the benefit of incremental feature improvements, and the shortcoming that upstream projects can be unresponsive.

However, in practice, some upstream bugs are resolved by distribution maintainers. This could indicate an important challenge as maintainers often lack the expertise of upstream package developers to investigate the root causes of those bugs (Ma et al. 2017). Distribution

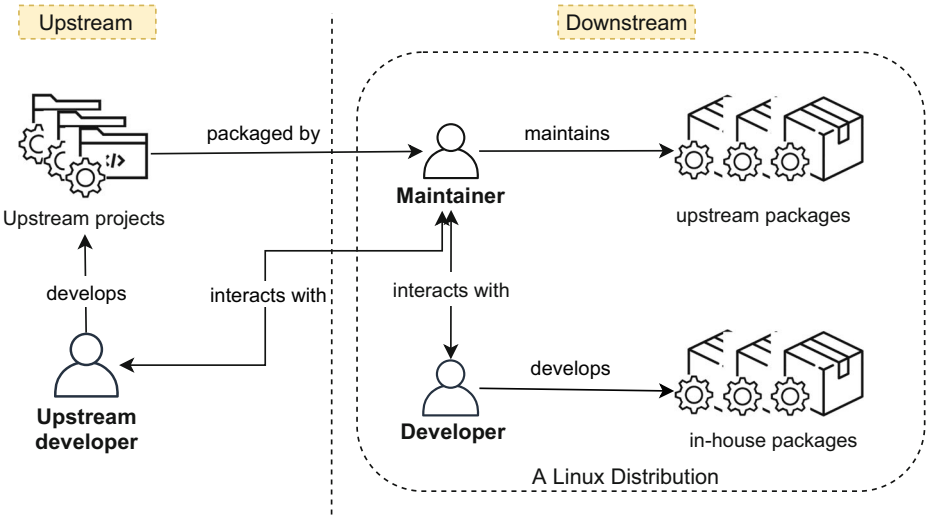


Fig. 1 Simplified illustration of the three major roles involved in the maintenance of upstream packages in Linux distributions. The maintainer role takes the responsibility of fixing bugs in upstream packages, interacting with the upstream developers, and integrating upstream fixes into a distribution. While the "maintainer" role is distribution-specific, the person having that role in a distribution could be part of the upstream project or external to it. The developer role develops in-house packages and addresses bugs in these packages

maintainers can feel compelled to provide quick workarounds for the upstream bugs in the form of local bug fixes (Ding et al. 2017). While providing quick workarounds can avoid delays due to communication with the upstream, it conflicts with the official guideline on freedesktop.org (https://www.freedesktop.org/wiki/Distributions/Packaging/WhyUpstream/). Therefore, in our study, we are interested in how maintainers manage upstream packages in practice.

Figure 2 indicates the typical bug fixing process for upstream packages. We derive the process from the official documents in Debian (https://wiki.debian.org/UpstreamGuide) and Fedora (https://fedoraproject.org/wiki/Staying_close_to_upstream_projects) by an open-coding approach. The 1st author derived the initial version of the process, discussed this version with the 2nd and 3rd authors, then iteratively resolved any disagreements. A triager validates an upstream bug report and assigns it to the maintainer of the upstream package after the upstream bug was reported by a user. Unlike fixing bugs in in-house packages, fixing bugs in upstream packages includes additional stages: (1) the maintainer reports the bug to upstream developers if they are accessible, (2) the upstream developers fix the bug and release a new version, and (3) the maintainer tests the fix and integrates it into the package repository of her/his distribution.

In practice, however, the process is more complex. If the upstream project is discontinued or not responsive, maintainers need to fix the bug themselves by producing *local patches* (Adams et al. 2016). Furthermore, upstream developers might take a long time to fix the bug, while maintainers also produce a local patch to reduce the impact of the bug. When maintainers produce the patches to fix a bug, they need to maintain the patches until the upstream project ships a release incorporating the fix. In other words, maintainers have the additional patch maintenance effort in keeping the patches compatible with the subsequent distribution releases until the bug is fixed upstream.

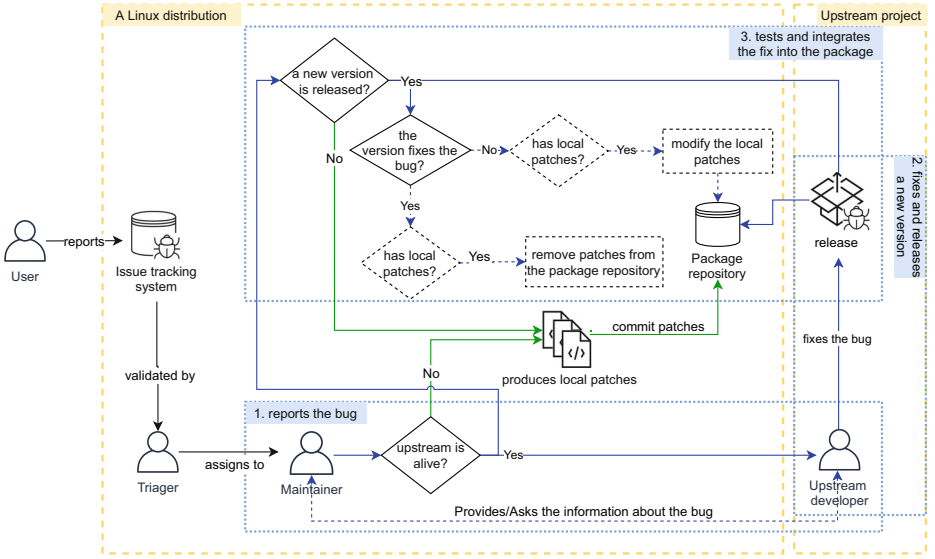


Fig. 2 An illustration of the bug fixing process for upstream packages. The three large blue boxes with the blue-background text indicate the three stages in which a maintainer in a distribution, according to the freedesktop.org guidelines, collaborates with upstream developers to fix a bug in their package. The blue arrow lines indicate the common steps. The green arrow lines represent the alternative reaction of a maintainer (e.g., upstream is discontinued or severe bugs). The dashed arrow lines and boxes represent optional steps. The yellow dashed boxes represent the instances/roles/actions inside a Linux distribution or an upstream project

2.2 Related Work on Upstream and Downstream (Distribution) Coordination

Prior studies investigated the potential problems and concerns regarding the coordination between upstream and downstream developers to fix bugs in upstream projects. For instance, Canfora et al. (2011) identified the prevalence of bug fixes between the FreeBSD and OpenBSD kernels. To fix bugs that occur between projects, developers in the two projects must communicate with each other to share the information of bugs. Ma et al. (2017) identified the difficulty for downstream developers to resolve upstream bugs in the scientific Python ecosystem, and observed that workarounds (i.e., temporary and local solutions) are common practices to reduce the long-lasting impact of upstream bugs. Our study investigates the cost of fixing bugs in upstream packages in terms of the number of developers, time, and local patches.

Ding et al. (2017) summarized four common workaround patterns that are used in practice to, on the one hand, require as few code changes as possible while, at the same time, being easy to remove such code changes afterwards. For example, replacing the buggy upstream method by using a similar method in the downstream project. We focus on the amount of effort for downstream developers to produce local patches to fix upstream bugs.

Zhang et al. (2018) observed that upstream bugs are often linked with longer developer discussions in the Ruby on Rails ecosystem. We also aim to understand the communication between upstream and distribution (downstream) developers by leveraging machine learning models to explain how collaborative activities affect the fixing of bugs in upstream packages. Ma et al. (2020) studied the impact of upstream bugs, such as buggy code hidden

in downstream projects until the failure inducing preconditions are satisfied. They proposed a technique to identify such impact in the downstream projects. They identified 1,132 downstream modules using 31 buggy upstream functions but only 47 out of these 1,132 modules have ever reported the buggy functionality in the past.

Another line of research related to upstream and downstream coordination is to port functions or code snippets in software projects from one to another. Porting software projects is close to our study since it is similar to the 3rd stage in Fig. 2, i.e., integrating and testing upstream fixes into a distribution. For instance, Ray and Kim (2012) studied the extent and characteristics of duplicate work happening in forked projects when cross-system porting from three BSD projects. They observed that porting happens periodically (at least 10 months) and the rate of porting does not reduce over time. At least 26% of active developers are involved in porting patches from other projects in the three BSD projects. Ray et al. (2013) characterized 5 categories (e.g., inconsistent variable renaming, porting code in a wrong operation) of porting errors from 113 and 182 porting errors in FreeBSD and the Linux kernel, respectively. Barr et al. (2015) developed an algorithm using genetic programming to transplant features in one software project into another automatically. The authors applied the algorithm to extract a feature from the donor project (source) and transplant it into the host project (target) through several experiments, including the VLC media player. However, our study focuses on the entire bug fixing process, including the coordination between upstream and distributions to fix high-severity bugs in upstream projects.

Different from the aforementioned studies, we study the prevalence of bugs in upstream packages within Linux distributions, which contain thousands of packages to better understand how maintainers manage upstream packages.

2.3 Related Work on Bug Fixing

Bug fixing is a complex, iterative, and costly process in distributed software development (Crowston and Scozzi 2008). Prior work investigated the amount of needed time and developers to fix bugs. Kim and Whitehead Jr (2006) observed that it takes a median of 200 days to resolve a bug in the ArgoUML and PostgreSQL projects. Duc et al. (2011) showed that, in three firm-involved open-source projects, firm-paid developers contribute higher to fix bugs and have a higher workload on average compared to volunteer developers.

Another popular line of work in this area is the analysis of the influential factors in the bug-fixing process. Weiss et al. (2007) observed that similar bugs take similar effort for developers to fix them in the JBoss project. Guo et al. (2010) studied two Microsoft products and found that the reputation of bug reporters affects the bug-fixing time. However, in five open-source projects, Bhattacharya and Neamtiu (2011) showed that the reputation of bug reporters has little influence on the bug-fixing time. Zhang et al. (2013) discovered that bugs with higher severity and priority are fixed faster, but the differences of bug fixing time between each of these bugs are large in three industrial projects.

The aforementioned studies extracted bug reports from a project and studied the bug fixing activities within the project, while our work extracts bug reports within a software ecosystem (i.e., Linux distribution). In other words, in our case, bugs exist in in-house and/or upstream projects in the Linux distribution. We also compare the cost of fixing bugs for maintainers and developers between upstream and in-house projects. To the best of our knowledge, no prior studies have examined the time for maintainers/developers to fix upstream bugs and investigated the corresponding cost.

3 Study Design

This section presents the design of our empirical study addressing the three research questions that we mentioned in the introduction. RQ1 studies the proportion and evolution of high-severity bugs in upstream and in-house packages that are fixed. RQ2 investigates the maintenance cost for upstream bugs that are fixed by distributions. Finally, to understand why bugs are fixed upstream, RQ3 examines the influential factors for a high-severity bug to be fixed upstream. Note that the specific methodology for each RQ is explained in Section 4.

3.1 Selection of Subject Systems

We select two Linux distributions, i.e., Fedora and Debian, based on their popularity and the ease of access to links between bugs and upstream for these two distributions. They leverage two of the most popular package managers in Linux. Fedora leverages *rpm* as its package manager, and Debian uses the *dpkg* package manager. According to the 2020 popularity ranking on distrowatch.com[2], Debian (6th) and Fedora (8th) are the two highest ranked root distributions, i.e., the source distributions from which derivative distributions (like Ubuntu, Mint, etc.) are forked to develop their own features independently.

In addition, Debian and Fedora follow different strategies of upstream bug management. Fedora is the upstream distribution of the commercial Red Hat distribution, which is its primary sponsor[3]. Therefore, Fedora follows the "upstream first" policy[4] from Red Hat to do software development (e.g., forwarding bugs to upstream, fixing bugs) side-by-side with upstream developers in the upstream projects. Only in specific cases (e.g., unresponsive upstream, release schedules), Fedora will fix bugs directly in the distribution and try to get patches accepted upstream to avoid future maintenance on those patches. By contrast, Debian depends on maintainers in its own community to maintain local patch repositories for packaged upstream packages, sending patches upstream whenever possible.

3.2 Data Collection

Figure 3 presents an overview of our study approach. We leverage two data sources for each of the studied distributions, i.e., the bug reports and the packages. Our studied period is from 2005 to 2019 to exclude the insufficient data of the early stage in each distribution and the latest data of 2020 (time of starting our study). The first release of Fedora was created from Red Hat in 2003 (https://fedoraproject.org/wiki/Releases/HistoricalSchedules), while the first release of Debian was made in 1996 (https://wiki.debian.org/DebianReleases). We archived our scripts and uploaded them to GitHub.[5]

3.2.1 Extract High-Severity Bugs

To obtain our dataset, we first extract the bug reports from the corresponding issue tracker of each studied distribution, as shown in Table 1, by either the relevant APIs that are offered by the issue trackers or custom crawlers. We only consider the high-severity (e.g., high,

[2] https://distrowatch.com/dwres.php?resource=popularity
[3] https://getfedora.org/en/sponsors/
[4] https://www.redhat.com/en/blog/what-open-source-upstream
[5] https://github.com/SAILResearch/suppmaterial-20-justina-upstream_bug_linux

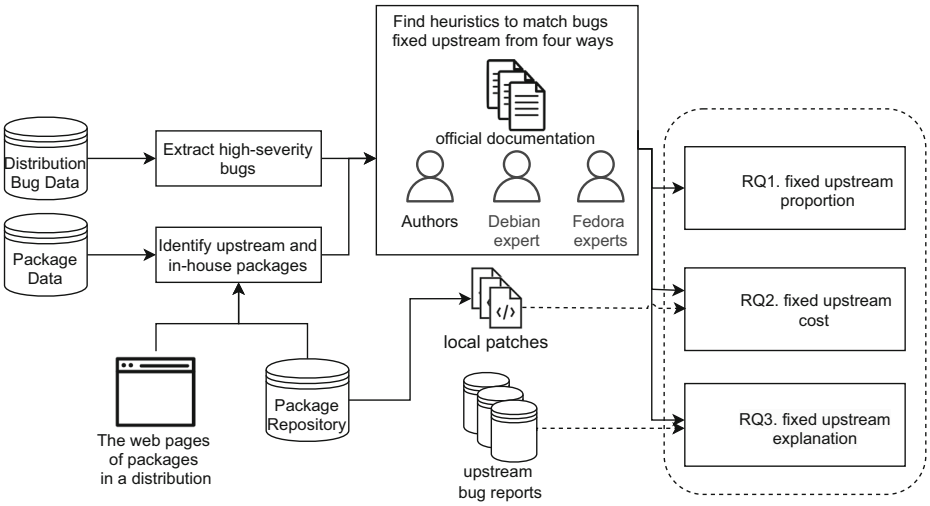


Fig. 3 An overview of our study approach. The solid arrow lines indicate the data sources used for all RQs. The dashed arrow lines indicate the particular data sources used for each specific RQ. We will describe the data sources in more details in the corresponding sections (RQ2 and RQ3)

urgent, critical) bugs because these bugs have a higher impact on users and are hence more important to fix (Menzies and Marcus 2008; Lamkanfi et al. 2010). We also identify duplicate bugs, similar to prior work (Bettenburg et al. 2008; Boisselle and Adams 2015), and remove them to avoid over-estimating the number of bugs and analyses in our work. Our dataset contains a total of 143,362 high-severity bugs across the studied distributions. The details of each distribution are presented in Table 1.

Note that even though an issue tracking system can also be used for feature requests, our dataset does not include the latter since Linux distributions apply a low-severity label or use other platforms for feature requests. In particular, Debian uses the *wishlist* value, which is a low-severity label, to indicate that a bug report is a feature request (https://www.debian.org/Bugs/Developer). Fedora does not track feature requests in its issue tracking system (https://docs.fedoraproject.org/en-US/fesco/).

3.2.2 Identify Upstream and in-House Packages

We then extract the list of packages and their relevant information (e.g., name and category) for each release of a given distribution. We also classify packages into two groups based on

Table 1 Basic statistics about the studied Linux distributions from January, 2005 to the end of 2019

Distribution	ITS	# high-severity bugs	# upstream packages[3]	# in-house packages[3]
Fedora	Bugzilla[1]	37,205	35,139 (76%)	11,332 (24%)
Debian	Debbugs[2]	106,157	33,044 (86%)	5,366 (14%)

[1] https://bugzilla.redhat.com/

[2] https://www.debian.org/Bugs/

[3] The percentage is the # in the cell divided by the sum of the # of upstream and in-house packages

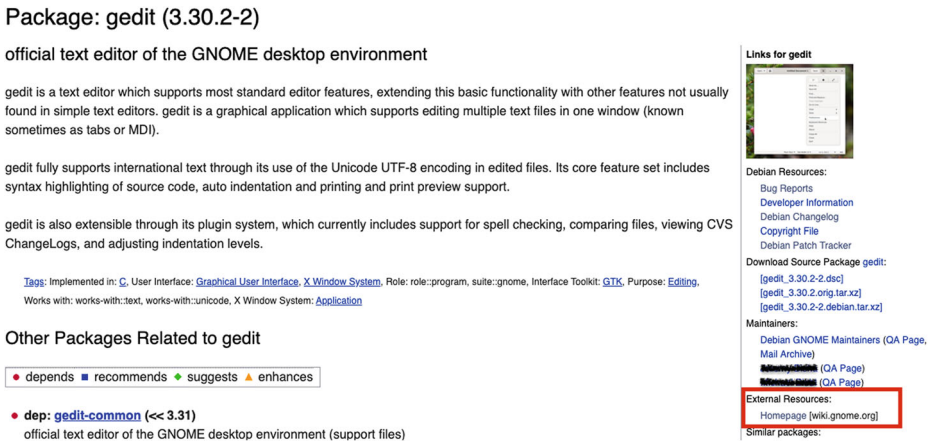


Fig. 4 An example of how to determine an upstream package. The *gedit* package is an upstream package since the red box indicates the source project, i.e., GNOME

the sources of packages: upstream (obtained directly from an upstream project) or in-house (developed by in-house teams).

For each package in a given distribution, we leverage the relevant web pages within the distribution or package repositories, when available, to identify whether there is an upstream project link. For example, Fig. 4 shows that *gedit* is an upstream package in Debian.[6] We identify 35,139 and 33,044 upstream packages in Fedora and Debian, respectively. After identifying upstream packages, we filter out 13,996 and 33,230 of the remaining packages in Fedora and Debian, respectively, for which we cannot find available web pages or repositories. We then assign the remaining packages as in-house packages.

To evaluate the precision of the above heuristic method that we use to determine upstream and in-house packages, we manually investigate 50 packages by randomly selecting 25 upstream packages and 25 in-house packages. For the 25 sample upstream packages, we leverage the upstream project link in each package to check whether the upstream developer team is active or not. We observe that all the upstream projects are available at the time when we write this paper, which indicates no false positives in our sample set.

For the 25 sample in-house packages, we search their package names to investigate whether there is an active upstream project somewhere on the Internet (e.g., GitHub), in which case our heuristic would not be correct. For each package, we investigate the metadata (e.g., description, contributors, folder structure) of the potential upstream projects carefully to identify the actual upstream project the package could belong to. After that, we determine that an upstream project is alive by its activities (e.g., commits, comments). We observe that 2 out of the 25 sample in-house packages were originally packaged from upstream projects. One of these two upstream projects is discontinued and thus distribution maintainers took ownership of that package, becoming an in-house package. The other one was customized by distribution maintainers, indicating it is an in-house package. Hence, our analysis of 50 packages indicates that the heuristic method achieves an accuracy of 100%.

[6] https://packages.debian.org/buster/gedit

4 Results

RQ1: What Percentage of Upstream Bugs are Fixed Upstream Versus Locally?

Motivation: Packages in a Linux distribution are either developed by in-house developers or packaged from the upstream projects, e.g., GNOME, Firefox, and the Linux kernel. In an ideal world, upstream developers provide new features and fix bugs, while distribution maintainers integrate new package versions into their distribution, making them interact seamlessly with the other packaged projects. Hence, the goal of this research question is to characterize the bugs from upstream packages. In particular, we investigate the evolution of reported and fixed high-severity bugs over time and the proportion of high-severity bugs that are fixed upstream.

Approach: To study high-severity bugs that are fixed upstream, we analyze the proportion of these bugs that are fixed upstream and those that are fixed locally in the two studied distributions, the evolution of reported and fixed bug reports in a distribution over time, and which application domains of packages are the most buggy. We compare the above analysis between the studied distributions.

Unfortunately, there is no unique way to identify whether a bug reported to a distribution has been fixed upstream or locally. For that reason, we have leveraged several heuristics, combined with manual validation of the heuristics by the authors, an actual Debian maintainer and three Fedora maintainers. Our interviewed maintainers are experienced individuals. We conducted the interview by asking four maintainers to avoid potential bias. More specifically, the Debian maintainer has been working in Debian for almost 10 years and currently maintains at least 35 packages. One of the three Fedora maintainers has been working in Fedora since its first release (i.e., 2007) and keeps contributing to upstream projects. This maintainer is an expert in several areas of package maintenance, having contributed to and being involved in the maintenance of dozens of packages. The second Fedora maintainer worked at Fedora for more than 6 years and is an expert in Python and auto deployment. The third Fedora maintainer has worked at Fedora for 4 years, specializing in the synchronization of package releases between upstream and Fedora.

Heuristics for Identifying Bugs that are Fixed Upstream We contacted the maintainers to figure out the potential sources that record the information of bugs fixed upstream. For example, the Debian maintainer suggested that *"the Debian BTS has features to track whether or not a bug is in the upstream portion of a package, and whether or not it is fixed upstream. See, in particular, the 'upstream' and 'fixed-upstream' tags on bugs."* and "*to see if [a bug] was fixed by a patch in the packaging.*". One of the Fedora maintainers suggested that *"BZ [numbers] help to track the fixed changes [in the changelog]."* Eventually, we settled on leveraging the tags in bug reports, the changelogs of packages, and the metadata of patches.

First, we leveraged the "fixed-upstream" tag of bug reports to identify a set of bugs that are fixed upstream in Debian (https://www.debian.org/Bugs/Developer), while in Fedora we look for bugs with the "UPSTREAM" label (https://fedoraproject.org/wiki/BugZappers/BugStatusWorkFlow). Using these heuristics, we identified 3,726 and 556 fixed-upstream bugs in Debian and Fedora, respectively. Since the "UPSTREAM" label in Fedora indicates that a bug has been reported to upstream and it will be integrated back into Fedora after the

```
6tunnel (0.10rc2-1) unstable; urgency=high

  * New upstream release (closes: #194851)
  * Bumped Standards-Version to 3.5.10
```

(a) An example of the changelogs for bug 194851 in Debian

```
* Tue Nov  1 2011 Tim Waugh <twaugh@redhat.com> 9.04-5
- Applied upstream fix for skipping "cached" outline glyphs (bug #742349).
```

(b) An example of the changelogs for bug 742349 in Fedora

Fig. 5 Two examples of bug fixes in the changelogs of upstream bug fixes in a package release in Debian and Fedora

upstream fixes it, we manually investigate whether bugs with the label were fixed upstream on a representative sample set of 227 fixed-upstream bugs (confidence level = 95%, confidence interval = 5%). We observe that 208 (92%) out of the 227 bugs were fixed upstream and 19 are unknown. The upstream bug reports of 10 bugs are not available, 4 bugs were reported to upstream developers by email, and 5 bugs do not contain the links to the respective upstream bug reports. Hence, we conclude that the heuristic has a high accuracy (at least 92%) to identify fixed-upstream bugs.

Second, we leverage the official guidelines maintainers should follow to compose the changelogs of a new package version. Specifically, Debian states that *"It is conventional that the changelog entry of a package that contains a new upstream version of the software looks like this: *New upstream release"* (https://www.debian.org/doc/manuals/developers-reference/pkgs.html#recording-changes-in-the-package) and adds the fixed bugs following a certain format (i.e., #id) (https://www.debian.org/doc/manuals/developers-reference/pkgs.html#upload-bugfix). Fedora also has a similar policy for maintainers to record updates to a new version and fixed bugs in a changelog (https://docs.fedoraproject.org/en-US/packaging-guidelines/#changelogs).

Based on these documented guidelines to identify fixed-upstream bugs, we inspect the changelogs (i.e., a total of 918 changelog entries) from a total of 50 randomly selected upstream packages (i.e., 25 packages in Debian and 25 packages in Fedora) to find potential patterns. We also randomly investigate the changelogs of the top 5 buggy upstream packages in both studied distributions (i.e., a total of 10 packages) to figure out additional patterns.

Finally, we derive the regular expression `"new upstream [\w ]*(?=release|version|fix)"` in Debian to find references to bugs fixed upstream, while in Fedora we would look for the keyword "upstream". Bug #194851 in Debian (Fig. 5a) and Bug #742349 in Fedora (Fig. 5b) illustrate these heuristics. Note that the bug-id in a changelog entry might indicate a low-/medium-severity bug or the reference bug-id to the upstream project (that may not be reported to the distribution) for traceability, which we filter out afterward from our dataset. Based on the heuristic, we identified a total of 5,180 (4,007 additional) fixed-upstream bugs in Debian and 68 additional fixed-upstream bugs in Fedora.

To validate this second heuristic, we manually investigate the comments of 75 randomly selected bugs from three popular packages (i.e., perl, apache2, and samba) in Debian. We observe that 20 out of the 75 bugs that are marked as fixed-upstream bugs by the second heuristic are the bugs that are reported to be fixed upstream, which are true positives. However, we found 3 out of the 75 selected bugs that are not marked as fixed-upstream bugs that should be classified into fixed-upstream bugs, suggesting a high precision (72/75 = 96%) and recall (20/23 = 87%) of the heuristic. The changelogs of these 3 bugs do not match any pattern in the heuristic. Moreover, we also note that the participants in 4 out of the

Table 2 Statistics of high-severity **upstream** bugs in each distribution

Distribution	# reported upstream bugs[1]	# fixed bugs[2]	# fixed upstream bugs[2]	# fixed locally bugs[3]
Fedora	32,530 (87.4%)	12,009 (36.9%)	624 (5.1%)	529 (4.4%)
Debian	88,300 (83.2%)	55,481 (62.8%)	7,733 (13.9%)	7,386 (13.3%)

[1]The percentage presents the percentage out of all high-severity bugs

[2]The percentage is computed relative to the cell on its left

[3]The percentage is computed relative to the # of fixed bugs

75 selected bugs were not able to determine whether the upstream fixed them or not. For instance, for those bugs no developer tested the fix of a bug or there were no comments for a long time (e.g., a few months or years) before the bug was closed and marked as fixed. Since we cannot classify bugs for which distribution maintainers themselves are not able to track the source of the fix, we do not consider these to be misclassifications of our heuristics.

On the other hand, since the heuristic only identifies a small number (i.e., 68) of fixed-upstream bugs in Fedora, we investigate whether maintainers follow the guidelines in practice. Although we observe 808 changelog entries containing bug-ids using the specific format, only 68 of them also contain the keyword "upstream". Furthermore, out of the 808 changelog entries in our dataset, only 121 refer to high-severity bugs.

Third, we also leverage the metadata in patches where maintainers tag the source (e.g., upstream, Debian) of a patch to identify fixed-upstream bugs. Debian suggests their maintainers follow the DEP-3 tagging guideline[7] and to use the *Origin* label to indicate the origin of the patch in the header. However, we observe that only 9.3% of patches in Debian used the DEP-3 format, in which we identify a total of 652 (133 additional) fixed-upstream bugs in Debian. Unfortunately, we cannot find a similar guideline in Fedora. Eventually, we obtain a total of 7,733 (13.9%) and 624 (5.1%) fixed-upstream bugs in Debian and Fedora, respectively (see Table 2).

Heuristics for Identifying Bugs that are Fixed Locally To identify bugs that are fixed locally, we derive heuristics based on bug reports and the changelogs of packages. First, we select all distribution bug reports that contain attached patch files, i.e., the attachments with one of the following extensions: .debdiff, .diff, or .patch in Debian. In Fedora, we leverage a similar heuristic based on the structured field (i.e., is_patch) of an attachment in Bugzilla. Using these heuristics, we identified 7,386 and 416 bugs that are fixed locally in Debian and Fedora, respectively. Second, we select changelog entries that contain at least one of the following keywords: work-around, workaround, temporar*, or patch and that do not contain the "upstream" keyword in Fedora. We identified 113 additional bugs that are fixed locally. Eventually, we obtain a total of 7,386 and 529 bugs that are fixed locally in Debian and Fedora, respectively (see Table 2).

We validate these two heuristics by manual investigation on the set of identified bugs. For the first heuristic, we investigate the comments of 25 randomly selected bugs in Debian and 25 randomly selected bugs in Fedora. We observe that 1 out of the 50 selected bugs was fixed upstream instead of locally, suggesting a high accuracy of 98% (49/50). Similar to the aforementioned cases of fixed-upstream bugs, for 2 out of the 50 selected bugs we were not able to determine whether distribution maintainers fixed it or not. For the 2nd

[7]https://dep-team.pages.debian.net/deps/dep3/

heuristic, we investigate the comments of 25 randomly selected bugs in Fedora and observe that in 1 out of the 25 bugs maintainers were not able to determine who fixed it. Due to the lack of precise information in the bug reports, we do not consider such cases to be misclassifications of our heuristic.

Bug Analysis To analyze the evolution of the reported bugs in in-house or upstream packages by year, we calculate the number of reported bugs for either group of packages divided by the total number of reported bugs (i.e., both in-house and upstream packages) in each year. To avoid the impact of the number of packages, we normalize the proportion of reported bugs by the number of active packages (that have identified bugs by our aforementioned heuristics) in each year. For example, when distribution maintainers integrate new upstream packages into their system, the number of reported bugs in upstream packages would exhibit a sudden spike, then drop afterwards. On the other hand, the actual count of reported bugs drops suddenly when maintainers remove packages along with their dependent ones.

Therefore, we calculate the normalized count of bugs (NCB) using the following equation to characterize the evolution of the proportion of bugs reported to maintainers per package and compare the evolution between in-house and upstream packages.

$$NCB_{report}(y) = \frac{BugCount_{report}(y)}{BugCount_{reportall}(y) * PackageCount(y)} \tag{1}$$

where:

y = year of interest
$BugCount_{report}$ = the number of bugs reported (in-house or upstream) in year y
$BugCount_{reportall}$ = the number of bugs in year y
$PackageCount$ = the number of active packages in year y

We also compare the evolution of upstream bugs that are recorded to be fixed by in-house teams and those that are recorded to be fixed by upstream developers. Similar to the reported bugs, we calculate the normalized count of fixed bugs (NCFB) using the following equation to capture the evolution of fixed bugs at the package level.

$$NCFB_{fixed}(y) = \frac{BugCount_{fixed}(y)}{BugCount_{fixedall}(y) * PackageCount(y)} \tag{2}$$

where:

y = year of interest
$BugCount_{fixed}$ = the number of upstream bugs fixed (locally or upstream) in year y
$BugCount_{fixedall}$ = the number of upstream bugs in year y
$PackageCount$ = the number of active packages in year y

Finally, we identify the categories of packages that more commonly contain high-severity bugs. Since Debian has a list of 59 package categories (https://packages.debian.org/stable/) and Fedora does not, we determine the package category for the upstream packages in Fedora using the following steps:

- Assign the package category of a package in Fedora when the name of the package matches exactly the name of a package in Debian.
- Assign the package category of a package in Fedora when the upstream source project of the package matches exactly that of a package in Debian.

– Based on our observations, we derive several heuristic rules from the naming convention of packages in Debian. For example, the names of packages in the localization package category contain the keyword "i18n" or "l10n". The names of packages related to the KDE desktop start with the prefix "kde-". We apply such heuristics to identify the package categories. The full list of our heuristics can be found in Appendix.

Results: 13.9% and 5.1% of upstream bugs in the studied distributions are explicitly recorded to be fixed upstream in Debian and Fedora, respectively, and 13.3% and 4.4% of upstream bugs are recorded to be fixed locally. The number of upstream bugs that are fixed upstream and those that are fixed locally are similar in both studied distributions (Fedora: 624 vs. 529, Debian: 7,733 vs. 7,386), while the percentages for Debian (13.9% and 13.3%) are much higher than for Fedora (5.1% and 4.4%).

As a corollary of our findings, surprisingly, for 73% and 90% of upstream bugs in Debian and Fedora, respectively, we are not able to automatically determine whether upstream developers or distribution maintainers produced the fixes. While distribution maintainers usually have this information in their minds, the fact that it is not explicitly recorded in bug reports or changelogs could complicate upstream bug management in the long run. Maintainers are not able to link several changes made for a particular bug. Furthermore, in the context of the software supply chain, missing such information exacerbates the risks of resolving security issues. For example, Heartbleed (i.e., CVE-2014-0160) was a severe security bug in the OpenSSL package. Debian maintainers cherry-picked a commit[8] made by two external contributors into their OpenSSL package repository on April 5, 2014, without the corresponding bug id. The commit directly changed source code in files (e.g. ssl/d1_both.c) instead of being recorded as explicit patches. Bug[9] #743883 (for Heartbleed) in Debian's issue tracker was closed on April 9. Shortly afterwards, Debian maintainers incorporated a new upstream release[10] on July 23, 2014, again without reference to the bug id. Since there is no corresponding bug id in these two commits, the only way for Debian maintainers to link both patches (direct fix and new upstream release) to bug #743883 would be searching for the common CVE-ID (CVE-2014-0160).

Note that in our analysis an bug fixed upstream refers to a bug that was fixed upstream first, before propagating the fix to the distributions, while a locally fixed bug refers to a bug that first was fixed locally in a distribution, before potentially propagating the fix upstream. The propagation process of an upstream fixed bug is automatic (as soon as the next release is sent to the package maintainers, the distribution will have access to the fix), while the propagation process of a locally fixed bug from a distribution to an upstream project is not guaranteed (e.g., in the case of distribution-specific modifications).

The normalized count of upstream bugs (NCFB) that are recorded to be fixed upstream has dropped over the years in Fedora, while the trend has been fluctuating in Debian. The proportion of bugs reported to be fixed upstream per year has been steady in both Fedora and Debian. For each year, a median of 5% and 14% of bugs across upstream packages are recorded to be fixed upstream with a standard deviation of 1.9 and 2.4 in Fedora and Debian, respectively.

[8] https://salsa.debian.org/debian/openssl/-/commit/96db9023b881d7cd9f379b0c154650d6c108e9a3
[9] https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=743883
[10] https://salsa.debian.org/debian/openssl/-/commit/2f21d2da895a465e151f4cb8f100040b897b5c1c

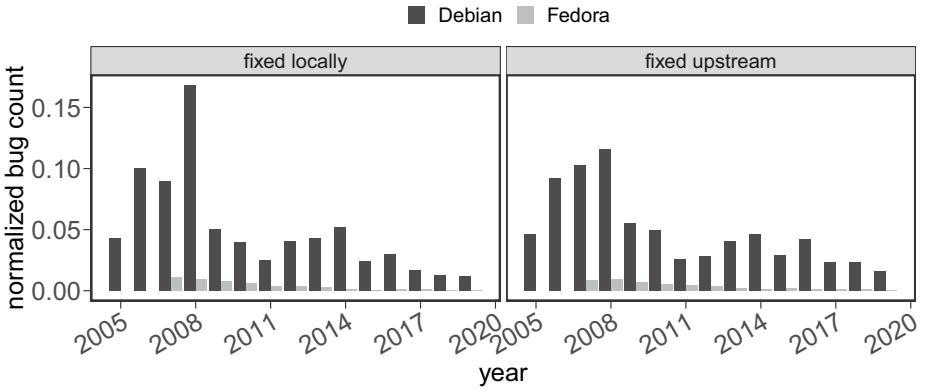


Fig. 6 The normalized counts of upstream bugs (relative to package count) **that are recorded to be fixed** locally (left) and upstream (right) have dropped in Fedora over time. Both trends continue to fluctuate in Debian

Given this stable trend of the proportion of bugs that are fixed upstream, we further analyze the trend at the package level using our normalized bug count (NCFB). Figure 6 shows the evolution of the normalized count of upstream bugs that are fixed upstream and locally over time, respectively. In both studied Linux distributions, these trends are similar since the proportion of bugs that are fixed upstream has been steady over time. However, Fedora has fewer bugs per upstream package that are fixed, compared to Debian, for both upstream and locally fixed bugs.

The normalized count of reported bugs (NCB) in upstream packages has also dropped over the years, with fluctuations in the studied distributions. Figure 7 shows the evolution of the normalized count of reported bugs in upstream and in-house packages across the studied distributions. The trend of the reported bugs in upstream packages (the right plot in Fig. 7) is similar to the corresponding trends for fixed bugs in Fig. 6, indicating that the proportion of the fixed bugs in upstream packages is consistent over the years. For example, from 2011 to 2014, both the trends of bugs that are fixed upstream and locally continue to increase in Debian (Fig. 6), which coincided with a similar trend for reported bugs in upstream packages (the right plot in Fig. 7). We also observe that the NCFB (Fig. 6) and NCB (Fig. 7) in Debian are larger than in Fedora. One possible reason for this is that they follow different release cycles of approximately 3 years[11] vs. 13 months[12], respectively. As such, the period for users to report bugs to a particular version of a package in a Debian release is longer than in a Fedora release.

Debian and Fedora have a similar catalog of buggy upstream packages. Figure 8 shows that 9 out of the top 10 buggy categories of upstream packages are common in the studied distributions. These 9 common categories of upstream packages account for 84% and 94% of upstream bugs in Debian and Fedora, respectively. However, the ranking of these categories of upstream packages is different between Debian and Fedora. For example, the libs category is the most buggy category in Fedora, while it is the 6th ranked category in Debian.

[11] https://www.debian.org/releases/
[12] https://fedoraproject.org/wiki/Fedora_Release_Life_Cycle

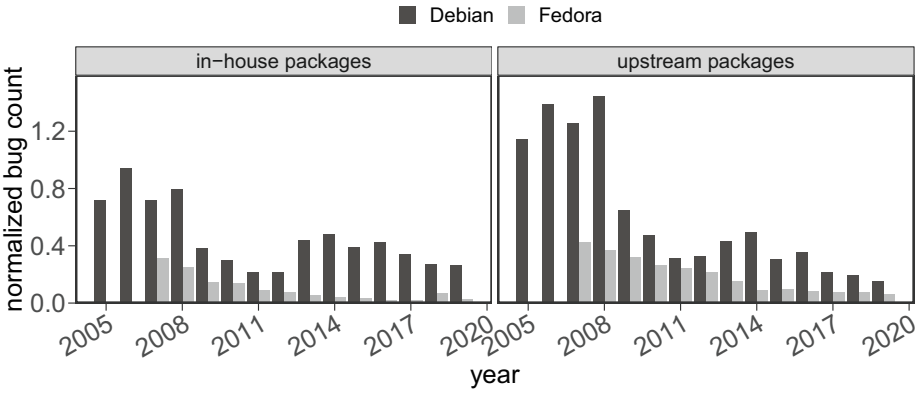


Fig. 7 Comparison between the evolution of the normalized count of **reported bugs** normalized by the package count for both in-house and upstream packages

Since packages in the libs category are common packages that are shared by many packages, Fedora ensures the quality of these packages to reduce the impact on other packages. One of the interviewed Fedora maintainers stressed "*the responsibility of the SRE's (Site Reliability Engineers) to make sure [the bugs in libs] are fixed for Fedora*." On the other hand, bugs in packages that are not in the libs category are reported to the corresponding upstream projects. Therefore, the fixes are tracked and developed upstream rather than in Fedora.

Summary of RQ1

A minority of at most 14% and 13% of upstream bugs are explicitly recorded as fixed by upstream developers or distribution maintainers, respectively, whereas for at least 73% of upstream bugs the origin of their fix is not explicitly recorded. The evolution of reported bug count and that of the number of bugs that are reported to be fixed upstream per upstream package has dropped over time in Fedora and Debian. In both studied distributions, the most buggy categories are libs and devel.

RQ2: What is the time and cost for fixing high-severity upstream bugs locally in distributions and maintaining such distribution fixes?

Motivation: The results in RQ1 show that only less than 14% of upstream bugs are explicitly recorded to be fixed upstream. However, distribution maintainers take responsibility for at least 13% of the remaining upstream bugs (with the 73% remaining cases in Debian not having explicit information w.r.t. who fixed them) even though they are not necessarily familiar with the upstream packages. For example, for maintainers who adopt an upstream project, Ubuntu states that *"You should have some experience with the upstream project - it really helps if you're working on something you know well."* (https://wiki.ubuntu.com/Upstream/Adopt) Lacking this experience increases the difficulty to come up with

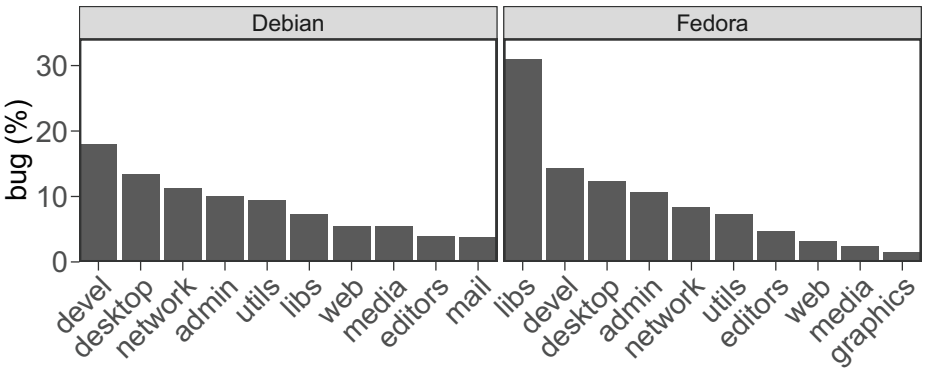


Fig. 8 The distribution of top 10 buggy categories of upstream packages. Fedora and Debian share 9 of 10 common buggy categories with different ranking orders. The categories that do not match are the **mail** category in Debian and the **graphics** category in Fedora

good fixes for the upstream bugs. As a consequence, downstream developers often propose workarounds (e.g., local patches) to reduce the impact of such bugs (Ma et al. 2017).

When a bug is successfully fixed by a distribution maintainer, the fix still needs to be submitted to the upstream project (https://www.freedesktop.org/wiki/Distributions/Packaging/WhyUpstream/). However, the fix might only be relevant to the maintainer's own distribution, or it might take time before the upstream project is able to validate the submitted fix. In such a case, distribution maintainers need to hold on to their bug fix changes as separate patches, i.e., diffs between the official upstream release and their modified version. Whenever upstream releases a new version of their project, the maintainer now needs to check if it incorporates his/her bug fix changes and, if not, whether the existing bug fix patch still applies cleanly (Ma et al. 2017). If not, the patch has to be fixed, and re-submitted online for (possible) inclusion in the next project release. Therefore, this is an additional cost for distribution maintainers to maintain the patches and keep them compatible with the following releases until distribution maintainers integrate the upstream releases including the bug fixes.

Considering the above challenges and the large number (i.e., 67,490) of fixed bugs in upstream packages (see Table 2), we are interested in gaining a deeper understanding of the involved costs of fixing upstream bugs locally.

Approach: As in Section 2.1 (Fig. 2), the bug fixing process for upstream packages includes collaboration between maintainers and upstream developers. We focus on the bugs that our heuristics in RQ1 are able to identify as fixed upstream or locally to estimate the cost of fixing an upstream bug. We leverage the following metrics that prior studies have widely used to study bug fixing: needed time for users waiting for fixes (Zhang et al. 2013; Kim and Whitehead Jr 2006; Bhattacharya and Neamtiu 2011), the number of participants in the bug fixing process (Bhattacharya and Neamtiu 2011; Canfora et al. 2020), the number of comments (Giger et al. 2010; Zhou et al. 2015; Zhang et al. 2012), the number of reference links (e.g., URLs to design documents or other bug reports) mentioned in

comments (Canfora et al. 2020), and the characteristics of local patches (Weimer 2006; Bhattacharya and Neamtiu 2011). We then compare the difference of the costs among upstream bugs that are fixed upstream vs. locally, and in-house bugs that are fixed locally.

We compare the needed time for users waiting for fixes among three types of bugs, i.e., bugs in upstream packages that are fixed upstream vs. locally, and bugs in in-house packages. However, fixing upstream bugs includes several different processes in certain circumstances, as shown in Fig. 2. We cannot measure the time from when upstream developers start fixing a bug to when maintainers know about the fix (although they might not yet have access to it). Therefore, similar to prior work (Zhang et al. 2013; Kim and Whitehead Jr 2006; Bhattacharya and Neamtiu 2011), we use the duration between when a bug is opened and when it is marked as fixed in the bug report of a given distribution.

We normalize three of our selected metrics to reduce the correlation between these metrics. In particular, we normalize the number of participants and the number of comments by the needed time to fix. The number of reference links mentioned in comments is normalized by the number of comments.

Additionally, maintainers produce local patches when these patches (fixes) are not immediately accepted upstream (Adams et al. 2016). Hence, maintainers maintain these patches locally for the subsequent distribution releases until the upstream fixes that particular bug. Local patches can be managed by different tools and stored in dedicated patch repositories. We extract the patches for each package from the corresponding package repository in the studied distributions.

We analyze how many patches are maintained in one distribution release, how long a patch file survives in terms of the number of distribution releases (i.e., a patch is removed when the upstream incorporates the fix), and the code churn of each patch file. Specifically, Debian maintainers use *Quilt* to control the series of patches related to the upstream source (https://perl-team.pages.debian.net/howto/quilt.html). In contrast to regular version control systems like git, tools like Quilt do not physically merge patches into a code base, but treat patches as first-class entities. In other words, patches can easily be applied, removed, swapped with other patches, etc. Fedora maintainers leverage the *git diff* command to track the changes of the upstream source for a bug as a patch (https://docs.fedoraproject.org/en-US/Fedora_Draft_Documentation/0.1/html/Documentation_Guide/sect-workflow-patching.html).

Results: In Debian, locally fixed upstream bugs make users wait for 39% less time than upstream bugs that are fixed upstream, but require users to wait for 52% more time than locally fixed in-house bugs. Figure 9 compares the user-waiting time for in-house bugs, upstream bugs that are fixed locally, and upstream bugs that are fixed upstream. In Debian, bugs that originate from upstream packages take longer to be fixed by the upstream developers themselves than being fixed locally. This inefficiency is mostly due to the communication overhead with the upstream projects . Since the upstream packages are widely used through Linux distributions (those "distribute" open source projects to end users), bugs from upstream packages typically are reported through them before being propagated upstream (Davies et al. 2010). After reporting to the upstream, maintainers need to wait for the upstream developers to respond to and fix the bug.

More specifically, in Debian, users have to wait a median of 38 days for local fixes of the upstream bugs, while they wait a median of 62 days for fixes that are from upstream and being integrated into Debian. The result of the Wilcoxon signed-rank test shows that

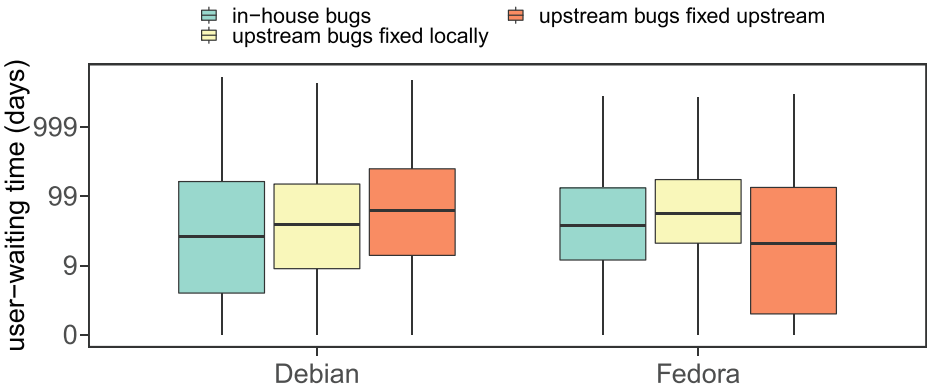


Fig. 9 Upstream bugs that are fixed locally (the yellow boxplots) take a longer time compared to in-house bugs (the green boxplots) across the studied distributions. In Debian, upstream bugs that are fixed locally (the yellow boxplot) take a shorter time compared to those that are fixed upstream (the orange boxplot)

the difference between bugs that are fixed upstream and those that are fixed in-house is statistically significant (i.e., p-value $\ll 0.01$, $\alpha = 0.01$), with a small effect size (0.22). On the other hand, users also wait longer to get local fixes for upstream bugs (i.e., a median of 38 days) than in-house bugs (i.e., a median of 25 days), with p-value $\ll 0.01$ ($\alpha = 0.01$).

In Fedora, locally fixed upstream bugs make users wait for 49% and 175% more time than locally fixed in-house bugs and upstream bugs that are fixed upstream, respectively. The reason why users wait a relatively shorter time for upstream bug fixes that are from upstream than local fixes is not that those bugs were actually fixed faster, but because maintainers mark the bug report as fixed once they report the bug to upstream or know that upstream developers have published the fix for the bug. Maintainers do not need to wait for upstream developers releasing a new version and integrating the version into their distribution manually before marking a bug as fixed. Antiya (https://release-monitoring.org/), a service to monitor the upstream releases in Fedora, will automatically integrate the new upstream releases of packages into Fedora (https://fedoraproject.org/wiki/Upstream_release_monitoring). Of course, users would still need to wait the entire time, suggesting that the results for Fedora might be more akin to those in Debian.

For example, a bug report related to taking screenshots when the screen was locked in Fedora, was closed by a maintainer who stated *"Fixed upstream, the 3.14.1 release in next week will bring the fix to Fedora."*.[13] In the corresponding upstream bug report[14], before posting the comment in Fedora, the maintainer posted 13 comments to help resolve the bug, such as testing the patches committed by other developers, pointing out the drawbacks of those patches, and proposing solutions and changes to fix the bug. Finally, the bug was resolved according to his suggestions and changes.

The number of participants per day and the number of comments per day when fixing upstream bugs locally are similar to when fixing in-house bugs and fixing upstream bugs upstream. Figures 10 and 11 indicate that fixing an upstream bug locally does not

[13] https://bugzilla.redhat.com/show_bug.cgi?id=1150444
[14] https://bugzilla.gnome.org/show_bug.cgi?id=737456

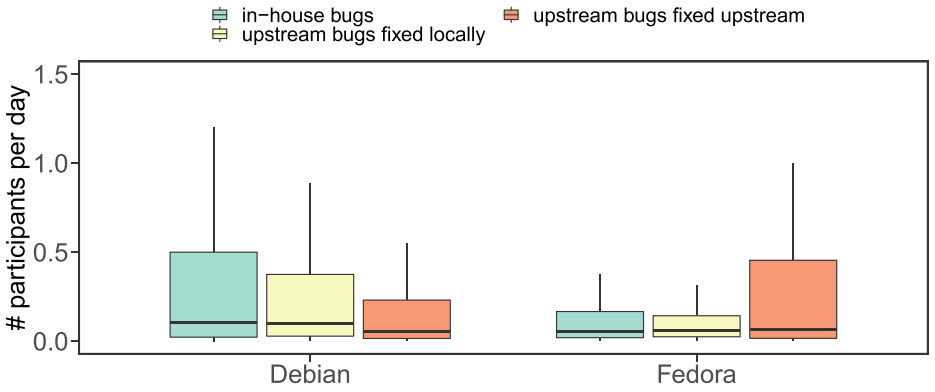


Fig. 10 The number of participants per day to fix an upstream bug locally (yellow) is equal to the number of participants per day to fix an in-house bug (green) across the studied distributions. Note that these boxplots remove the top 1% of data points for better visualization

require more participants per day, nor more discussion (comments) per day, compared to fixing in-house bugs and fixing upstream bugs upstream. One possible explanation is that upstream bugs that are fixed locally occur due to certain builds or modifications in a given distribution. For example, a Debian maintainer mentioned *"Actually we had to put some handlers back in after upstream removed them because the keys were not generated in the Debian kernel."*, which made the keystroke events be reported twice on certain platforms.[15]

In Debian, distribution maintainers record more links to reference documents when fixing upstream bugs locally than when fixing in-house bugs, but fewer links than for upstream bugs that are fixed upstream. In Fedora, the number of reference links per comment is similar among all three types of bugs. The reference links in comments indicate whether fixing a bug needs more additional information (e.g., design documents, requirements). Fedora follows the "upstream first" policy[16] that encourages its maintainers to discuss solutions of a bug in the upstream bug tracker, where the larger community can vet the solutions, as discussed in Section 3. As such, Fedora maintainers tend to comment on and discuss bugs in the upstream bug tracker rather than the distribution bug tracker. Figure 12 shows that the number of links to reference documents when fixing upstream bugs locally is larger than when fixing in-house bugs in Debian (Wilcoxon test: p-value $\ll 0.01$, $\alpha = 0.01$ and the negligible effect size). For example, an upstream bug[17] that is fixed locally and an in-house bugs[18] have 1.33 (4 links among 3 comments) and 0.43 (3 links among 7 comments) reference links per comment, respectively.

Maintainers maintain equal or fewer local patches per package for upstream packages within one distribution release compared to in-house packages. More specially, both Debian and Fedora maintain a median of 2 patches per upstream package, while they maintain a median of 3 and 2 patches per in-house package, respectively. As shown in Fig. 13,

[15] https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=586924
[16] https://www.redhat.com/en/blog/what-open-source-upstream
[17] https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=489553
[18] https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=424456

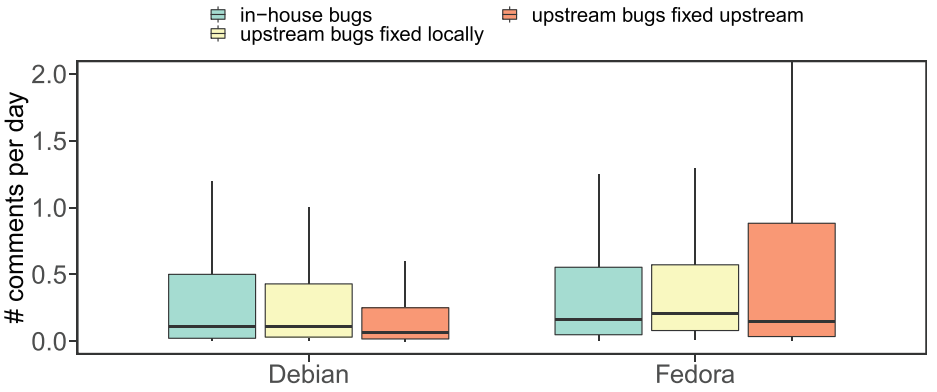


Fig. 11 The number of comments per day in an upstream bug is similar to the number for an in-house bug. Note that these boxplots remove the top 1% of data points for better visualization

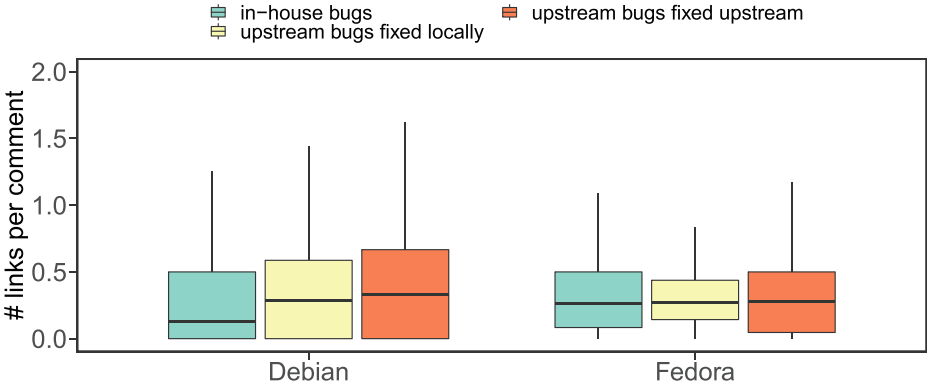


Fig. 12 The number of reference links per comment in an upstream bug that is fixed locally (yellow) is larger than the number for an in-house bug (green) in Debian, though they are similar in Fedora. Note that these boxplots remove the top 1% of data points for better visualization

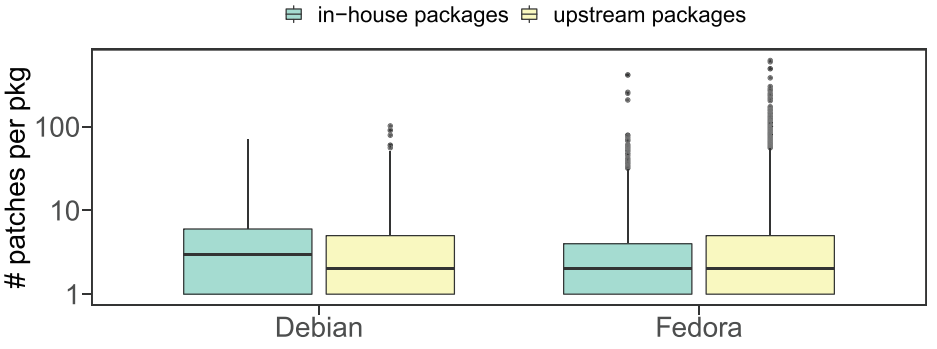


Fig. 13 The number of patches per upstream package in one release of a given distribution is smaller than the number of patches per in-house package in Debian, while the number is equal for upstream and in-house packages in Fedora, though there are more outliers for upstream packages

930 upstream packages (outliers) require a higher number of local patches than in-house packages. According to the guidelines of upstream package management in Linux distributions (https://www.freedesktop.org/wiki/Distributions/Packaging/WhyUpstream/), when the upstream bugs are critical (e.g., security issues or severe impacts), the distribution maintainers can produce local patches for such bugs to reduce their effects. For such a short-term fix, distribution maintainers potentially face a long-term maintenance effort. They need to check the compatibility of their patches with each subsequent release of upstream packages, until the upstream projects release a new version containing the bug fixes. On the other hand, in-house packages are not necessarily developed by in-house developers from scratch, they can take ownership of an orphaned upstream package or customize a package themselves as an in-house package (as mentioned in Section 3.2). Hence, in-house developers can also produce patches for bugs in in-house packages.

However, surprisingly we observe that the maintenance cost of the patches for upstream packages is slightly lower than for in-house packages in terms of the number of patches per package, with a small effect size (Wilcoxon test: p-value $\ll 0.01$, $\alpha = 0.01$).

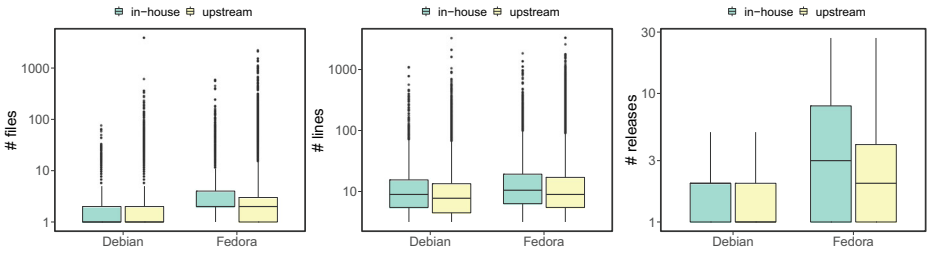
Maintainers remove local patches for upstream packages faster than for in-house packages in terms of the number of distribution releases. Figure 14 shows the distribution of code churn and survival time of each patch file in one distribution release. The difference in survival time of patch files between upstream and in-house packages is statistically significant in both Fedora and Debian, with a p-value $\ll 0.01$ (Wilcoxon test: $\alpha = 0.01$). The Cliff's Delta effect size indicates that the magnitude of the difference is small (0.34) and median (0.45) in Fedora and Debian, respectively. As patches are produced to fix bugs and ensure the stability of a distribution, these patches will be removed afterwards, i.e., the survival time is shorter when the upstream projects release new bug fixes. In contrast, local patches for in-house packages are maintained much longer in terms of the number of distribution releases.

Summary of RQ2

Although the cost of fixing upstream bugs upstream is similar to the cost of fixing them locally in terms of the number of participants per day and the number of comments per day, the cost of fixing upstream bugs upstream is larger than the cost of fixing them locally, in terms of the time users have to wait for bug fixes and the number of reference links per comment. Debian maintainers mention more links to references per upstream fixed bug than per locally fixed bug. Moreover, when maintainers produce local patches for bugs, the cost of producing such patches per upstream package is lower than per in-house package in terms of code churn.

RQ3: Which high-severity bugs are most likely to be fixed upstream?

Motivation: While observations in RQ2 indicate that distribution maintainers need less information to fix upstream bugs locally than upstream, they still need more information than to fix bugs of in-house packages. Similarly, users wait longer for local fixes to upstream bugs than for in-house packages. While compared to bugs fixed upstream, maintainers need less information to fix upstream bugs locally. Therefore, for a distribution as a whole, it is important to understand the characteristics of fixing bugs upstream. Specifically,



(a) #changed files (b) #changed lines (c) Survival time

Fig. 14 The code churn and survival time of each patch file for the upstream and in-house packages in each Fedora and Debian release. Across the studied distributions, the maintenance effort for the upstream packages is lower than for the in-house packages in (a), (b) and (c)

we aim to understand what types of bugs are fixed upstream and whether the interaction between upstream and downstream influences bugs being fixed upstream. Therefore, we can offer suggestions for maintainers to reduce the maintenance burden and identify additional opportunities for assigning bugs to upstream.

Approach: We are interested in the key factors that influence the likelihood that a bug will be fixed upstream. Figure 15 illustrates an overview of the process to build our explanatory models. First, we select a dataset that consists of a balanced number of bugs that are fixed upstream and those that are fixed locally, with one dataset for Debian and one for Fedora. In particular, we randomly select 500 bugs that are fixed upstream and 500 bugs that are fixed locally from the dataset in Table 2, obtaining two datasets of 1,000 high-severity bugs (**Model Dataset**).

Feature Collection Then, we extract a total of 27 features in 6 dimensions regarding bugs, packages, and collaboration between upstream and distributions. Similar to prior work (Shihab et al. 2013; Ohira et al. 2012; Xia et al. 2014), we extract the common factors (e.g., description, bug-fixed time) of a bug report to characterize bugs (Shihab et al. 2013; Zimmermann et al. 2010; Xuan et al. 2012), reporters (Fan et al. 2018; Just et al. 2008; Xuan et al. 2012; Marks et al. 2011), and bug fixing activities (Mockus et al. 2002; Jeong et al. 2009). For example, Fan et al. (2018) found that reporter's experience is an important indicator that determines a valid bug or not. To understand the relation between upstream

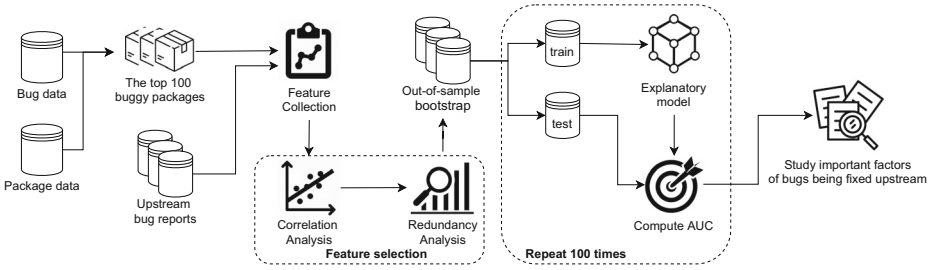


Fig. 15 Overview of RQ3 model construction process

and downstream, we add several novel dimensions along with several metrics and highlight them in bold in Table 3. We now describe the dimensions with novel metrics in more detail.

Bug Fixing History Since upstream projects are the source of upstream packages, maintainers report a bug to the upstream project before adding the reference link of the upstream bug report in the bug report. We leverage the number of reference links to the upstream bug reports to indicate such behavior. Furthermore, an upstream bug can affect many distributions and their users in the Linux ecosystem. Therefore, a similar upstream bug can already be reported in the issue tracking systems of different distributions. We add two metrics regarding the similarity of previous bug reports either between upstream and distributions, or between peer distributions (see Table 3).

Additionally, since upstream packages may affect in-house packages or vice versa, we add the number of reference links to the related in-house bug reports to indicate the interference between upstream and in-house packages. Furthermore, bug triage is an essential step in the bug fixing process but a time-consuming task (Hu et al. 2014). We add one novel metric, i.e., the number of bugs that are fixed upstream before the current bug report in a distribution, to measure the performance of the package maintainers with respect to upstream bugs.

Internal Collaboration Activities The number of comments in a bug report is related to its fixing time (Hooimeijer and Weimer 2007). Hence, we further extract the URLs in the comments as an indicator to reflect the maintenance effort of maintainers to search the relevant information (e.g., design documents, bug reports in other bug trackers). Then, we classify these URLs into internal links (referring to internal documents) and external links (referring to external documents). When maintainers discuss the fix for a bug, they may post the links regarding the relevant information (e.g., design doc, similar bugs or solutions) in the comments of the bug report. Therefore, we use the number of external and internal links to measure the communication activities. In addition, according to RQ1, bugs in upstream packages might be addressed by maintainers instead of upstream developers. We measure the number of maintainers involved with fixing upstream bugs based on the comments of a bug report. We identify those maintainers via their email address, which contains the domain name of their distribution (e.g., debian.org) (https://www.debian.org/doc/manuals/developers-reference/index.en.html, https://fedoraproject.org/wiki/Join_the_package_collection_maintainers#Create_a_Fedora_Account).

External Collaboration Activities In general, an upstream bug is first reported by a user of a distribution before maintainers forward the upstream bug to upstream developers. While upstream developers are fixing the bug, they often lack the information and context of a specific bug occurring downstream (Ma et al. 2017) and require additional information (e.g., logs from the user, tests in certain circumstances). Maintainers coordinate with them to provide the required information, even from the user who reported the bug. We use four metrics extracted from the comments of the local and upstream bug reports to reflect the interaction between upstream and downstream (see Table 3). Due to the variety of upstream bug trackers (e.g., GitHub, Bugzilla), it is a time-consuming task to collect all the bug reports of upstream projects. Hence, we analyze the distribution of upstream bug trackers and only consider the upstream projects using Bugzilla and Debbugs (i.e., Debian bugs) as their bug trackers. In the selected 100 upstream projects, 73% of the upstream reference links in bug reports link to such bug trackers.

Table 3 The dimensions of features in our models. The feature names in bold are new features with respect to the relation between upstream and distribution fixing bugs

Feature name	Type	Description
Bug Characteristics		
#Description	Numeric	The number of words in the description of the bug
Severity	String	The severity of the bug
Priority	String	The priority of the bug
Reporter Characteristics		
Is internal reporter	Boolean	States whether the reporter is an in-house developer. In-house developers are determined based on the developer's email that contains the domain name of a distribution (e.g., debian.org) (https://www.debian.org/doc/manuals/developers-reference/index.en.html, https://fedoraproject.org/wiki/Join_the_package_collection_maintainers#Create_a_Fedora_Account)
Reporter experience	Numeric	The number of bugs reported by the same developer before this bug
Reporter experience to the package	Numeric	The number of bugs reported to the same package by the same developer before this bug
Bug Fixing Activities		
Fixed time	Numeric	The time duration in the number of days between when the bug was reported and when the bug is fixed
Triaged time	Numeric	The time duration in the number of days between when the bug was reported and when developers started to fix the bug
Bug Fixing History regarding the current bug		
#Related in-house bug reports	Numeric	The number of related in-house bug reports that were reported before the current bug
#Referred upstream links	Numeric	The number of reference links to the upstream bug reports
Peer similarity score	Numeric	The cosine similarity between the bug report and peer-distributions' bug reports that were reported before the current bug
Similarity score[1]	Numeric	The cosine similarity between the current bug and the corresponding upstream bug reports
Package category	String	The application domain of the package

Table 3 (continued)

Feature name	Type	Description
#Fixed bugs	Numeric	The number of fixed bugs in the package before the current bug report
#Fixed-upstream bugs	Numeric	The number of bugs that are fixed upstream in the package before the current bug report
#Participants	Numeric	The number of participants across the bug report history before the current bug report
Internal Collaboration Activities		
#Comments	Numeric	The number of comments in the bug report
#External links in comments	Numeric	The number of unique external reference links that developers posted in the comments
#Internal links in comments	Numeric	The number of unique internal reference links that developers posted in the comments
%Internal comments	Numeric	The proportion of comments added by distribution maintainers or developers in the bug. Distribution maintainers and developers are determined based on the developer's email that contains the domain name of a distribution (e.g., debian.org) (https://www.debian.org/doc/manuals/developers-reference/index.en.html, https://fedoraproject.org/wiki/Join_the_package_collection_maintainers#Create_a_Fedora_Account)
#Maintainers	Numeric	The unique number of maintainers contributing to the bug by posting comments
External Collaboration Activities		
Has upstream developers	Boolean	States whether upstream developers contributed to the bug by posting comments
#External developers	Numeric	The unique number of external developers contributing to the bug by posting comments
%External comments	Numeric	The proportion of comments added by external developers in the bug. When the email of a developer does not contain the domain name of a distribution, we determine the developer is an external developer (https://www.debian.org/doc/manuals/developers-reference/index.en.html, https://fedoraproject.org/wiki/Join_the_package_collection_maintainers#Create_a_Fedora_Account)
Has distribution maintainers[1]	Boolean	States whether distribution maintainers attached comments to the upstream bug report
#Comments from distribution maintainers[1]	Numeric	The number of comments posted by distribution maintainers in the upstream bug report
#Comments upstream[1]	Numeric	The number of comments in the upstream bug report

[1] The features are computed from the corresponding upstream bug report if it is available

Model Construction and Performance Assessment Finally, we build logistic regression models to understand the relation between the studied metrics and the possibility of getting a bug fixed upstream (see Fig. 15). Our models classify upstream bugs into being fixed upstream or in-house. Since the results in RQ1 show that each Linux distribution has separate characteristics, we build a model for each Linux distribution instead of one model for all the studied distributions.

Feature Selection After extracting the metrics, we apply correlation and redundancy analysis, similar to prior work (Lee et al. 2020; Rajbahadur et al. 2021; da Costa et al. 2018), before constructing our models. The correlation analysis (Lee et al. 2020; Rajbahadur et al. 2021) step removes highly correlated variables (whose Spearman correlation is $> 80\%$), since correlated variables can affect the interpretation of the model. While the correlation analysis removes correlated variables, it does not detect all the redundant variables, i.e., variables that can be predicted with more than one other independent variable. Therefore, we then conduct a redundancy analysis (McIntosh et al. 2016; da Costa et al. 2018) that identifies for each metric whether it can be predicted by other metrics. The redundancy analysis is performed by the `redun` function[19] from the `Hmisc` package in R. The discarded metrics are marked with the dagger symbol (†) in Table 4.

AUC We assess the performance of our models using the area under the curve (AUC), which is calculated by 100 out-of-sample bootstrap iterations with replacement (Thongtanunam et al. 2016; McIntosh et al. 2016). AUC is a better measurement for evaluating the performance of a classification algorithm (Ling et al. 2003), since, unlike accuracy, AUC is not impacted by class distributions. Moreover, AUC evaluates performance across a range of thresholds instead of requiring one specific threshold like precision, recall, and F-Measure do. In each bootstrap iteration, as shown in Fig. 16, we draw a bootstrap sample set from the **Model Dataset** with replacement for each of the studied distributions. The train set is re-sampled with replacement from the bootstrap sample, while the test set consists of the data points from the bootstrap sample that are not in the train set. The AUC of our models is calculated by the test set, which enables to compute true positive rates and false positive rates on unseen data.

To assess the performance of our models across the studied distributions, we train a model on one dataset and assess its performance on the other dataset. For example, we train a model on the Fedora dataset and test this model on the Debian dataset. Note that we remove the features that do not have the corresponding ones across the studied distributions. We use the commonly occurred features which are the majority of our extracted features.

Log-likelihood ratio tests In addition, we use a log-likelihood ratio (LR) test (Huelsenbeck and Crandall 1997) to evaluate the performance of our regression models fit to the data at the dimension level. Similar to prior work (Thongtanunam and Hassan 2020; Lee et al. 2020), we compute the overall $\text{LR}\chi^2$ to represent the performance of fit for the model using all the studied metrics (i.e., the full model) against the null model (i.e., a model without any metrics). We then compute the $\Delta\text{LR}\chi^2$ to indicate the performance of fit for the model with the metrics of interest against the model without using the metrics of interest.

[19] https://www.rdocumentation.org/packages/Hmisc/versions/4.2-0/topics/redun

Table 4 Statistics of our regression models. The goodness of fit for the metrics of interest ($\Delta LR\chi^2$) is shown in a proportion to the overall $LR\chi^2$

		Fedora	Debian
Overall $LR\chi^2$		526***	211***
AUC		0.87	0.73
AUC (across distributions)		0.63	0.81
Bug Characteristics	$\Delta LR\chi^2$	31***(6%)	17***(8%)
#Description	χ^2	5.27* (+)	0.86^o (+)
Severity	χ^2	0.09^o	**15.22****
Priority	χ^2	**22.79****	-
Reporter Characteristics	$\Delta LR\chi^2$	1.1(0.2%)	12***(6%)
Is Internal reporter	χ^2	0.79^o (-)	5.25 * (-)
Reporter experience	χ^2	0.01^o (-)	4.05* (-)
Reporter experience to the package	χ^2	0.15^o (-)	†
Bug Fixing Activities	$\Delta LR\chi^2$	17***(3%)	1(0.5%)
Fixed time	χ^2	**16.41***** (+)	0.97^o (+)
Triaged time	χ^2	0.09^o (+)	-
Bug Fixing History	$\Delta LR\chi^2$	81***(15%)	92***(44%)
#Related in-house bug reports	χ^2	1.91^o (+)	†
#Referred upstream links	χ^2	†	†
Peer similarity score	χ^2	4.25* (-)	1.35^o (-)
Similarity score	χ^2	**21.88***** (+)	**54.44***** (+)
Package category	χ^2	†	†
#Fixed bugs	χ^2	†	†
#Fixed-upstream bugs	χ^2	**19.24***** (+)	8.17** (+)
#Participants	χ^2	**13.90***** (-)	0.34^o (-)
Internal Collaboration Activities	$\Delta LR\chi^2$	201***(38%)	23***(11%)
#Comments	χ^2	7.21** (-)	†
#External links in comments	χ^2	6.77** (+)	7.84** (+)
#Internal links in comments	χ^2	**46.14***** (-)	**9.32**** (-)
%Internal comments	χ^2	0.01^o (+)	5.25* (-)
#Downstream developers	χ^2	**16.68***** (-)	4.12^o (+)
External Collaboration Activities	$\Delta LR\chi^2$	8*(2%)	15**(7%)
Has upstream developers	χ^2	2.67^o (+)	0.11^o (+)
Has distribution contributors	χ^2	†	†
#External developers	χ^2	†	**13.49***** (-)
%External comments	χ^2	5.06* (-)	3.96* (+)
#Comments from distribution contributors	χ^2	†	†
#Comments upstream	χ^2	†	†

Statistical significance: *** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$, ^{o} p ¿ 0.05

† Discarded due to the correlation or redundancy analysis

- Metrics are not available in the distribution

Bold numbers indicate that correlation is significant, taking into account Bonferroni correction

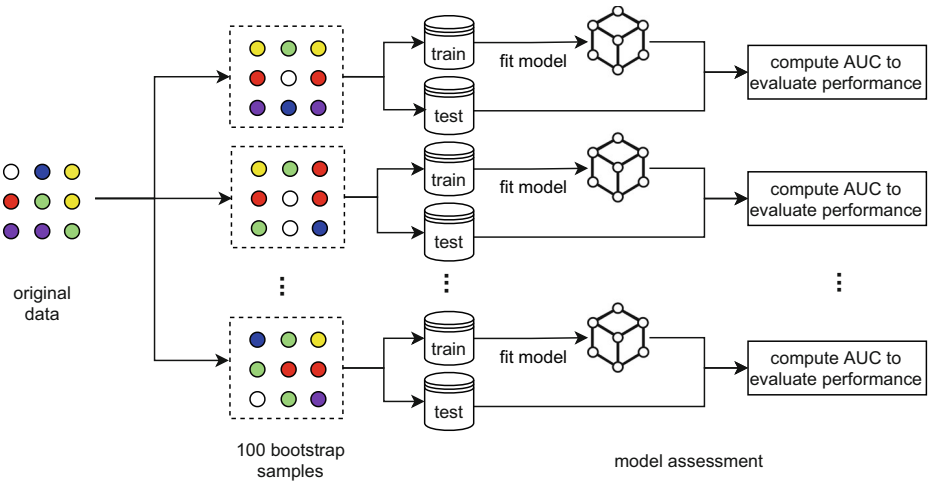


Fig. 16 The model building and validation process of the models used in RQ3

Association analysis We use Wald statistics to analyze the explanatory power (Wald χ^2) of each studied metric to the fit of the regression models. The larger the Wald χ^2 value, the higher the explanatory power of a metric to the model (Harrell Jr et al. 1984). In addition, we use the regression coefficients of the regression model to examine the direction of the association of the studied metrics.

Results: Our logistic regression models reach a median AUC of 0.87 and 0.73 to explain the likelihood that a bug will be fixed upstream in Fedora and Debian, respectively. We note that our models for the two studied distributions are remarkably good compared to a random model (AUC of 0.5). The high AUC values in modeling Fedora and Debian suggest that our models have a high explanatory power when studying bugs being fixed upstream, hence, we analyze our models' features in more details.

The Fedora model achieves an AUC of 0.63 to explain the likelihood that a bug will be fixed upstream for Debian, and the Debian model achieves an AUC of 0.81 for Fedora. These AUC values indicate the generalizability of our models for understanding bugs being fixed upstream. An upstream project can be part of multiple distributions, suggesting that a potential issue can occur in a common set of upstream packages across different distributions. The generalizability of our models across the studied distributions indicates that bug fixes from upstream projects can propagate across distributions, thus sharing common bug fixing patterns.

Dimension Level The dimension of bug fixing history has the strongest association with the likelihood that a bug will be fixed upstream in Debian, while the dimension ranks 2nd in Fedora. Table 4 shows that 4 out of 5 metrics and 2 out of 4 metrics that capture the history of a fixed bug contribute a significant amount of explanatory power to our Fedora and Debian models, respectively. Moreover, Table 4 shows that the $\Delta LR\chi^2$ values for our studied metrics account for 15% and 44% of the overall $LR\chi^2$ in the Fedora and Debian models, respectively. This result suggests that a bug being fixed upstream tends

to be related most to its fixing history in the ecosystem. For example, the number of bugs that are fixed upstream before the current bug reflects the extent of support from upstream developers to resolve their bugs in distributions.

On the other hand, **the dimension of internal collaboration activities between upstream projects and distribution maintainers accounts for the highest percentage (i.e., 38%) of the overall LRχ^2 in the Fedora model, while the dimension accounts for 11% (2nd ranked) in the Debian model**. Since developers are geographically distributed, they have several communication channels (e.g., emails, Internet Relay Chat (IRC) channels) (Panichella et al. 2014) to discuss and make decisions in software development other than the bug report comments. Therefore, future research may study whether collaboration activities occur in another communication channel. In addition, the strategy of upstream package management in Fedora is to mainly develop and fix bugs upstream, as mentioned in Section 3. Internal collaboration activities could be a negative indicator of a bug being fixed upstream.

Feature Level The similarity score between the upstream and distribution bug reports is highly associated with the likelihood that a bug will be fixed upstream (p-value $<$ 0.001). Table 4 indicates that there is a significant positive correlation between the cosine similarity of the upstream and distribution bug reports and the likelihood that a bug will be fixed upstream. In particular, the similarity score of the upstream and distribution bug reports contributes a relatively large amount of explanatory power in the Fedora (10%) and Debian (38%) models. This result indicates that the more likely a similar bug was reported in the past, the more likely this bug will be fixed upstream. This result suggests that maintainers should take into account identifying/reusing similar bug reports upstream before requesting an upstream bug, which could save unnecessary effort. This result is also in line with the observation of the guidelines for working with upstream in Debian that *"If the package has an upstream bug tracker then searching it for similar reports can be useful"* (`?debianbugtriage`).

The number of bugs that are previously fixed upstream in one upstream package is significantly associated with a bug being fixed upstream (p-value $<$ 0.01). Table 4 shows that the number of previously fixed bugs from upstream packages has a positive association with the possibility of a bug being fixed upstream. This result indicates that some upstream projects might be used to interacting more closely with distributions, being more likely to continue such interactions. On the other hand, it might not be likely to expect upstream projects without such interactions to suddenly start doing this by themselves.

The number of internal links in comments is significantly associated with a bug being fixed upstream (p-value $<$ 0.01). Table 4 shows that the number of internal links in comments has a negative association with the possibility of a bug being fixed upstream. This result suggests that maintainers discuss how to resolve a bug by posting additional internal references in comments, the less likely the bug will be fixed upstream. Similarly, the number of external links in comments has a positive association with the possibility of a bug being fixed upstream. The more maintainers refer to external links to discuss the fix for a bug, the more likely the bug will be fixed upstream.

Additionally, there is a gap in the explanatory power of the number of internal links in comments between Debian (6%) and Fedora (21%). One possible explanation is the difference in strategy of Debian and Fedora to manage upstream packages, as discussed in Section 3. Fedora prefers to vet solutions in upstream projects rather than distribution bug

reports. Therefore, the number of internal links in comments has a larger amount of explanatory power, and a negative association with a bug being fixed upstream in the Fedora model. In other words, the more internal reference links in a Fedora's bug report, the more likely the bug is not related to the source code of the upstream project. By contrast, the number of external links in comments has a similar amount of explanatory power in both models.

Summary of RQ3

Our regression models achieve a median AUC of 0.87 and 0.73 for the Fedora and Debian datasets, respectively. The dimension of bug fixing history has a significant association with the likelihood that a bug will be fixed upstream in the Debian (44%) model, while the dimension of internal collaboration activities is the largest in the Fedora (38%) model. The similarity score between upstream and distribution bug reports, the number of internal reference links, and the number of bugs that previously have been fixed upstream in a package share a significant association with the likelihood that a bug will be fixed upstream (p-value < 0.01).

5 Discussion

This section discusses the implications of our findings. We focus on the interaction across distributions and upstream projects (Section 5.1) and distribution maintainers of upstream packages (Section 5.2).

5.1 Implications for the Interaction Across Distributions and Upstream Projects

There is a need for tool support to explicitly track the bug fixing process across upstream and distributions. In RQ1, we observe that for the majority (at least 73%) of upstream bugs it is not clear who fixed them. The two popular and established Linux distributions that we studied use a mixture of different practices (e.g., certain fields such as labels or tags in a bug, but do not enforce these practices) to record that a bug is related to upstream and has been fixed upstream. Since such a recording mechanism is not a mandatory feature in the studied distributions, maintainers might not record the information of bug fixing before closing a bug. Missing such information increases the difficulty of maintainers to trace the progress of fixing an upstream bug and the difficulty of new maintainers to learn the history of managing a package. Furthermore, in case of vulnerabilities, it is important to have full traceability information.

Therefore, distributions should establish a means to unambiguously record this information across upstream projects and their distribution, even though upstream packages are packaged and distributed by multiple Linux distributions, corresponding to an N-to-N relation of traceability links between bug reports of packages and distributions. Trying to establish a centralized database, while the most intuitive solution, might not be feasible in practice, since it is unclear where to locate the database and who should be responsible for maintaining it. Furthermore, an upstream project usually focuses on bugs reported in its own issue tracker. Since a given upstream project can be packaged and distributed by

several distributions, the upstream developers need substantial additional effort to record the corresponding distribution bug ids in their issue tracker and maintain this information. This is especially true since each distribution has its own bug repository technology (making automation less effective) and upstream projects usually lack human resources and are relatively small.

On the other hand, the issue of software supply chain attacks has led to increasing adoption of different metadata formats for specifying a project's Software Bill of Materials (SBOM[20]). Technologies like SPDX (Software Package Data Exchange) allow to (semi-)automatically generate the SBOM information for a given project or distribution, including lists of components, dependencies, etc. This SBOM information is shipped in a given release of a software product. Some SBOM technologies even allow to specify which patches were installed, which bug fixes (for which bugs), etc., which seems to be a promising means for both upstream projects and distribution alike to formally specify the traceability information relative to their product in a scalable manner. Of course, this does not solve the issue of recovering traceability links for historical releases.

Active collaboration between upstream developers and distribution maintainers should be encouraged in order to fix bugs and better distribute fixes to the needed users. In RQ3, our explanatory models indicate that the similarity score between upstream and distribution bug reports and the number of previously fixed bugs in upstream projects are associated with a higher likelihood of a bug being fixed upstream. While upstream is the best place to perform quality assurance (https://www.freedesktop.org/wiki/Distributions/Packaging/WhyUpstream/), sending more bugs to upstream and fixing them there requires contributions from both sides (i.e., upstream and distributions).

First, an upstream project might not have an explicit bug tracker or different bug tracking technology than the distribution where distribution maintainers report their bugs to and search for the old bugs of the upstream project. In such a case, the collaboration activities between upstream and distributions happen in other channels (e.g., emails), which is difficult for people who are less involved with tracking or reviewing the fixing history.

Second, distribution maintainers require providing sufficient information (e.g., logs, configurations) of a bug for upstream developers when they validate whether the bug is valid in the upstream project or specific to a given distribution.

Furthermore, even if the upstream project would be interested in fixing a bug, the fix typically only applies directly to a new version of the upstream project. The results in RQ2 indicate that both Debian and Fedora maintain a median of 2 patches per upstream package. Since distributions offer their users a certain period of support, distribution maintainers need a backport fix for the old versions, especially for security fixes. For example, Debian maintainers had to backport the fix for a security bug[21] in the *glib2.0* package from the upstream release 2.60.0 to the old version 2.58.3 in Debian.

In addition, it would be essential for distributions to collaborate together more systematically in terms of exchanging fixes to common bugs. When bug fixes are made by one distribution instead of the upstream project, these fixes will not be shared with the other distributions until the upstream project integrates the fix in its next release, unless distribution maintainers proactively search for peer fixes in other distributions. For example, a bug[22] in

[20] https://www.ntia.gov/SBOM

[21] https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=931234

[22] https://bugzilla.redhat.com/show_bug.cgi?id=772257

the *augeas* package was resolved in Fedora more than one month earlier than when a similar bug[23] was reported in Debian. At the same time, the bugs in an upstream package might affect several distributions without them knowing this, hence precious time could be saved by closer collaboration between peer distributions.

5.2 Implications for Distribution Maintainers

There is a need for tool support to scan existing bug reports in the ITS of the upstream project or peer distributions for similar reports that potentially were fixed upstream. Since an upstream package can be part of several distributions, a bug in the upstream package in a given distribution might have been reported by the maintainers in other distributions, and potentially already have a fix. Our models suggest that the similarity score between a previously reported upstream bug report and a distribution bug report is an important factor for a bug being fixed upstream (Table 4). In our dataset, there are 9% and 6% of the high-severity bugs in Debian and Fedora, respectively, with a median similarity score of 0.87 and 0.88 to their correlated upstream bug report. These numbers suggest a high similarity score of bug reports between upstream projects and distributions, even though the actual proportion of bugs with such high similarity is low ($<10\%$). Since we only computed the similarity score for bugs that contain a reference link to their respective upstream bug report, there might be a larger set of bugs with such a high similarity score to an upstream report, unknown to maintainers, and hence, without such information being recorded explicitly. As such, using techniques of duplicate bug report detection could help find such cases.

Although there are some existing techniques to detect duplicate bug reports (Boisselle and Adams 2015; Alipour et al. 2013), these tools typically focus on detection between individual software systems in a 1-to-1 relation. In Linux distributions, such tools might not be able to detect similar bug reports between upstream projects and distributions, which is an N-to-N relation (N upstream projects vs. N distributions carrying them). In addition, after an upstream project ships a release, the package version of the upstream release is usually changed by maintainers to meet their distribution versioning guideline, i.e., adding the version number used in a given distribution. The upstream version of the release reflects several different version numbers in several distributions. On the other hand, a distribution might skip a certain package version of the upstream release. Hence, the complexity of mapping package versions from distributions to upstream projects increases the difficulty of identifying similar bug reports across distributions and upstream.

There is a need for tool support to integrate the web context of the reference links in the comments of a bug report and provide a summary of these links to speed up fixing upstream bugs. Having more external (less internal) links in bug comments is associated with the possibility of a bug being fixed upstream (Table 4). Since the maintainers of an upstream package can leave or join over time, a new maintainer needs to learn how to fix bugs in the package and might take care of an unfixed bug from other retired maintainers. In addition, anyone who is interested in fixing bugs can contribute to the comments of a bug. The links in the comments provide an additional channel for gathering information in the bug fixing process, which can be a source of additional context about a bug. The results in RQ2 indicate that Debian maintainers need more links to reference documents when fixing upstream bugs locally than when fixing in-house bugs (Fig. 12). An external link can

[23] https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=731132

refer to information related to the bug (e.g., the configuration setting is disabled by default) in the upstream package. In contrast, the internal links in the comments of a bug report indicate that the upstream bug might be related to other packages in a given distribution (e.g., conflicts with other packages or the distribution-specific process). To increase the ability of fixing upstream bugs, the need of tool support to retrieve and summarize the links in comments helps maintainers to have comprehensive information related to a bug.

6 Threats to Validity

6.1 Construct Validity

Although we only study the high-severity bugs in two studied distributions, our dataset contains a large number of bugs in total (i.e., 143,362). High-severity bugs typically indicate fatal errors and even crashes, while low-severity bugs represent the effect of such bugs is low on the functionality of a software system (Lamkanfi et al. 2010). Although the severity field of a bug might be filled out randomly (Lamkanfi et al. 2010), our findings from high-severity bugs reveal the lower-bound of involved cost of fixing upstream bugs in-house, since lower-severity bugs also need to be addressed.

In addition, developers might assign the severity label of a bug based on their own knowledge. However, Debian and Fedora have been in the top 10 popular Linux distributions over the past decade, according to the distrowatch.com rankings. They have derived and documented a mature policy of assigning severity labels (https://www.debian.org/Bugs/Developer, https://fedoraproject.org/wiki/BugZappers/BugStatusWorkFlow) for their maintainers to fix bugs and maintain their popularity. Note that a bug could be tagged as severe initially but later converted into a low severity bug, or vice versa. Since our work aims at studying bugs that really were severe, the final status of a bug, captured the latest bug characteristics, suffices for us.

We classify packages in a distribution into upstream and in-house packages based on the reference link to the source upstream project in related web pages of a package or package repositories, when available. Since an in-house package can be packaged from an upstream project a long time ago and the upstream project is no longer maintained, the link to the upstream project could have been removed. It is challenging to derive a heuristic to discriminate between such a type of package and in-house packages even though such a type of package differs from in-house packages that are developed from scratch. In our manual study sample in Section 3.2, we observe that 1 (4%) out of the 25 in-house packages was such a type of package. Future work can study such a type of package and evaluate its impact.

Although we aim to identify bugs that are fixed upstream or locally in a comprehensive manner, like any heuristic-based approach, we may miss other scenarios of upstream fixed bugs and local fixed bugs. For instance, a bug labeled as fixed upstream may have a root cause related to the different packaging processes between upstream and the distribution, where maintainers need to customize the upstream project (e.g., modify locations for configuration files) (Adams et al. 2016). When maintainers cannot customize the upstream project by themselves, they need to ask the upstream to adjust its packaging process (e.g., add configurations, switch to a new build system) to fix bugs in the packaging process. Since the bugs in the packing process only happen when maintainers package the upstream project to fit their distribution policy, we do not consider the case of customization for upstream packages and do not investigate the logs of the packaging process.

Since the studied distributions provide their maintainers flexibility on how to deal with packages, maintainers decide whether or not to reference relevant commits in their distribution's bug reports and changelogs, or to include bug ids in the commit messages of their version control system. For example, Debian's *openssh* repository[24] contains 8,492 commits between 2005 and 2019 (i.e., our studied period), 769 (9%) of which included a total of 212 Debian bug ids, with 28 out of these 212 bugs being high-severity bugs. Since Debian has 180 high-severity bug reports for *openssh* in its bug repository, this means that only 28 (16%) out of 180 high-severity bugs were explicitly recorded in commit messages. Upstream developers have the same freedom w.r.t. including bug ids in the commit messages of their version control system.

Furthermore, a bug fix can be made by a maintainer of another distribution, which is neither a local fix nor an upstream fix. For example, bug 851052[25] was a flickering issue while inputting characters and happened in several distributions (e.g., Ubuntu). A Ubuntu contributor made a commit directly to the Debian repository to fix the bug.

6.2 Internal Validity

For the measures of issue discussions (i.e., comments) and user waiting time for fixes, we only examine the number of comments per day, the number of links per comment and user waiting time between upstream bugs and in-house bugs. The experience of issue commenters and issue difficulty are not considered in our study. Nevertheless, we focus on high-severity bugs to partially mitigate these issues and provide the overall insight of the costs of fixing upstream bugs for distribution maintainers.

In RQ3, the regression models present the relation between the possibility of a bug that is fixed upstream and a set of software metrics. We build our explanatory models to gain an understanding of how distribution maintainers submit bugs to upstream and how these bugs are fixed upstream successfully. Although we do not use a mixed-effect model, our models reach high AUC values, i.e., 0.87 and 0.73 for Fedora and Debian, respectively. There might be other factors that influence a bug that is fixed upstream. Hence, we encourage future research to add more features to improve the explanatory power of the model and conduct qualitative studies to understand how upstream and downstream developers collaborate together to fix a bug.

6.3 External Validity

We perform a case study on two Linux distributions. Although Debian is commonly used as a case study of prior research (Herraiz et al. 2011; Claes et al. 2015; Boisselle and Adams 2015; Davies et al. 2010) and Fedora is one of the popular Linux distributions[26], the results may not generalize to all Linux distributions. Moreover, Fedora and Debian represent the roots of their families. Since derivative distributions can inherit packages from Fedora and Debian, maintainers in derivative distributions can also report bugs in those packages to Fedora and Debian. Maintainers fix such bugs that slip through into derivative distributions for downstream developers in their families. Hence, additional replication studies are required in order to characterize the practices and challenges of upstream package management in the Linux ecosystem.

[24] https://salsa.debian.org/ssh-team/openssh/-/commits/master
[25] https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=851052
[26] https://distrowatch.com/index.php?dataspan=2020

7 Conclusion

In this paper, we investigate 143,362 high-severity bugs across two Linux distributions to understand the prevalence of upstream bugs and the cost of fixing them for distribution maintainers in practice.

Our analysis shows that upstream bugs are dominant across the studied distributions, yet only less than 14% of these bugs are explicitly recorded fixed by upstream and at least 13% of these bugs are recorded fixed by the distribution. Our findings suggest that the vast majority (at least 73%) of upstream bugs are unknown who fixed them. In other words, there is a lack of traceability between high-severity bugs and their fixes, potentially impacting maintenance activities by other contributors than a package's maintainer.

Our measurements for upstream bugs indicate that the cost of fixing an upstream bug locally requires a longer needed time for users to wait for fixes and more reference links per comment, compared to bugs in in-house packages, but less than for upstream bugs fixed upstream. However, the number of participants per day and the number of comments per day are similar. In addition, the patches made for the upstream packages are simpler than the patches for the in-house packages in terms of code churn.

Finally, we build regression models to understand the factors that influence bugs being fixed upstream. We suggest that maintainers search for bug fixing history in the upstream projects before starting to fix a bug. Since searching the fixing history of bugs increases the effort for maintainers, we suggest to leverage techniques of duplication bug detection. Furthermore, the similarity score of upstream and distribution bug reports, the number of reference links, and the number of bugs that have previously been fixed upstream in a package are important indicators of a bug being fixed upstream.

Appendix

Table 5 shows the heuristics derived from the naming convention of packages in Debian and their associated categories.

Table 5 The heuristics and their categories

Category	Heuristics
Admin	The package name contains one of the following keywords: abrt, grubby, dnf, yumex
Database	The package name contains the keyword "mysql"
Devel	The package name starts with the keyword "perl-" or "python-", or ends with the keyword "-devel", or contains one of the following keywords: -java-, glibc, anaconda, golang.
Desktop	The package name starts with the keyword "kde-"
Editor	The package name contains one of the following keywords: document, documentation, -doc, javadoc
Fonts	The package name contains one of the following keywords: font
Libs	The package name contains one of the following keywords: library, -lib, mesa, lroax
Localization	The package name contains the keyword "i18n" or "l10n"
Network	The package name contains the keyword "networkmanager" or "freeipa"

Acknowledgments We would like to thank the Debian and Fedora maintainers that graciously provided us feedback. Furthermore, special thanks to Rahul Bajaj and the anonymous reviewers for their insightful

comments. The findings and opinions in this paper belong solely to the authors, and are not necessarily those of Huawei. Moreover, our results do not in any way reflect the quality of Huawei software products.

Declarations

Conflict of Interests The authors declare that they have no conflict of interest.

References

Debian - bug triage wiki (online). https://wiki.debian.org/BugTriage. Last accessed: 2020-12-01

Debian - information regarding the bug processing system for package maintainers and bug triagers (online). https://www.debian.org/Bugs/Developer. Last accessed: 2020-12-01

Debian - list of package categories (online). https://packages.debian.org/stable/. Last accessed: 2020-12-01

Debian - managing packages (online). https://www.debian.org/doc/manuals/developers-reference/pkgs.html#recording-changes-in-the-package. Last accessed: 2020-12-01

Debian - managing packages (online). https://www.debian.org/doc/manuals/developers-reference/pkgs.html#upload-bugfix,. Last accessed: 2020-12-01

Debian - quilt for debian maintainers (online). https://perl-team.pages.debian.net/howto/quilt.html. Last accessed: 2020-12-01

Debian developer's reference (online). https://www.debian.org/doc/manuals/developers-reference/index.en.html. Last accessed: 2020-12-01

Debian releases (online). https://wiki.debian.org/DebianReleases. Last accessed: 2020-12-01

Fedora - anitya - upstream release monitoring system (online). https://release-monitoring.org/. Last accessed: 2020-12-01

Fedora - bug status workflow (online). https://fedoraproject.org/wiki/BugZappers/BugStatusWorkFlow. Last accessed: 2020-12-01

Fedora - creating a patch (online). https://docs.fedoraproject.org/en-US/Fedora_Draft_Documentation/0.1/html/Documentation_Guide/sect-workflow-patching.html. Last accessed: 2020-12-01

Fedora - fedora engineering steering committee (online). https://docs.fedoraproject.org/en-US/fesco/. Last accessed: 2020-12-01

Fedora - join the package collection maintainers (online). https://fedoraproject.org/wiki/Join_the_package_collection_maintainers#Create_a_Fedora_Account. Last accessed: 2020-12-01

Fedora - upstream release monitoring (online). https://fedoraproject.org/wiki/Upstream_release_monitoring. Last accessed: 2020-12-01

Fedora historical schedules (online). https://fedoraproject.org/wiki/Releases/HistoricalSchedules. Last accessed: 2020-12-01

Fedora packaging guidelines (online). https://docs.fedoraproject.org/en-US/packaging-guidelines/#changelogs. Last accessed: 2020-12-01

Staying close to upstream projects (online). https://fedoraproject.org/wiki/Staying_close_to_upstream_projects. Last accessed: 2020-12-01

Ubuntu - adopt an upstream (online). https://wiki.ubuntu.com/Upstream/Adopt. Last accessed: 2020-12-01

Upstream guide (online). https://wiki.debian.org/UpstreamGuide. Last accessed: 2020-12-01

Upstream guidelines for linux distributions (online). https://www.freedesktop.org/wiki/Distributions/Packaging/WhyUpstream/. Last accessed: 2020-12-01

Adams B, Kavanagh R, Hassan AE, German DM (2016) An empirical study of integration activities in distributions of open source software. Empirical Software Engineering (EMSE'16) 21(3):960–1001

Alipour A, Hindle A, Stroulia E (2013) A contextual approach towards more accurate duplicate bug report detection. In: 2013 10th Working Conference on Mining Software Repositories (MSR'13). IEEE, pp 183–192

Anbalagan P, Vouk M (2009) On predicting the time taken to correct bug reports in open source projects. In: 2009 IEEE international conference on software maintenance (ICSM'09). IEEE, pp 523–526

Barr ET, Harman M, Jia Y, Marginean A, Petke J (2015) Automated software transplantation. In: Proceedings of the 2015 international symposium on software testing and analysis (ISSTA'15), pp 257–269

Bettenburg N, Premraj R, Zimmermann T, Kim S (2008) Duplicate bug reports considered harmful... really? In: 2008 IEEE international conference on software maintenance (ICSM'08). IEEE, pp 337–345

Bhattacharya P, Neamtiu I (2011) Bug-fix time prediction models: can we do better? In: proceedings of the 8th working conference on mining software repositories (MSR'11), pp 207–210

Boisselle V, Adams B (2015) The impact of cross-distribution bug duplicates, empirical study on debian and ubuntu. In: 2015 IEEE 15th international working conference on source code analysis and manipulation (SCAM'15). IEEE, pp 131–140

Canfora G, Cerulo L, Cimitile M, Di Penta M (2011) Social interactions around cross-system bug fixings: the case of freebsd and openbsd. In: Proceedings of the 8th working conference on mining software repositories (MSR'11), pp 143–152

Canfora G, Di Sorbo A, Forootani S, Pirozzi A, Visaggio CA (2020) Investigating the vulnerability fixing process in oss projects: Peculiarities and challenges. Computers & Security 99:102067

Claes M, Mens T, Di Cosmo R, Vouillon J (2015) A historical analysis of debian package incompatibilities. In: 2015 IEEE/ACM 12th working conference on mining software repositories (MSR'15). IEEE, pp 212–223

Crowston K, Scozzi B (2008) Bug fixing practices within free/libre open source software development teams. Journal of Database Management (JDM'08) 19(2):1–30

da Costa DA, McIntosh S, Treude C, Kulesza U, Hassan AE (2018) The impact of rapid release cycles on the integration delay of fixed issues. Empirical Software Engineering (EMSE'18) 23(2):835–904

Davies J, Zhang H, Nussbaum L, German DM (2010) Perspectives on bugs in the debian bug tracking system. In: 2010 7th IEEE working conference on mining software repositories (MSR'10). IEEE, pp 86–89

Ding H, Ma W, Chen L, Zhou Y, Xu B (2017) An empirical study on downstream workarounds for cross-project bugs. In: 2017 24th Asia-Pacific software engineering conference. IEEE, pp 318–327

Duc AN, Cruzes DS, Ayala C, Conradi R (2011) Impact of stakeholder type and collaboration on issue resolution time in oss projects. In: IFIP international conference on open source systems. Springer, pp 1–16

Fan Y, Xia X, Lo D, Hassan AE (2018) Chaff from the wheat: Characterizing and determining valid bug reports. IEEE Transactions on Software Engineering (TSE'18) 46(5):495–525

Giger E, Pinzger M, Gall H (2010) Predicting the fix time of bugs. In: Proceedings of the 2nd international workshop on recommendation systems for software engineering, pp 52–56

Guo PJ, Zimmermann T, Nagappan N, Murphy B (2010) Characterizing and predicting which bugs get fixed: an empirical study of microsoft windows. In: Proceedings of the 32Nd ACM/IEEE international conference on software engineering-Volume 1 (ICSE'10), pp 495–504

Harrell Jr FE, Lee KL, Califf RM, Pryor DB, Rosati RA (1984) Regression modelling strategies for improved prognostic prediction. Statistics in Medicine 3(2):143–152

Hauge O, Ayala C, Conradi R (2010) Adoption of open source software in software-intensive organizations–a systematic literature review. Inf Softw Technol 52(11):1133–1154

Herraiz I, Shihab E, Nguyen ThanhHD, Hassan AE (2011) Impact of installation counts on perceived quality: A case study on debian. In: 2011 18th working conference on reverse engineering. IEEE, pp 219–228

Hooimeijer P, Weimer W (2007) Modeling bug report quality. In: Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering (ASE'07), pp 34–43

Hu H, Zhang H, Xuan J, Sun W (2014) Effective bug triage based on historical bug-fix information. In: 2014 IEEE 25th international symposium on software reliability engineering. IEEE, pp 122–132

Huelsenbeck JP, Crandall KA (1997) Phylogeny estimation and hypothesis testing using maximum likelihood. Annual Review of Ecology and Systematics 28(1):437–466

Jeong G, Kim S, Zimmermann T (2009) Improving bug triage with bug tossing graphs. In: Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering (FSE'09), pp 111–120

Just S, Premraj R, Zimmermann T (2008) Towards the next generation of bug tracking systems. In: 2008 IEEE symposium on visual languages and human-centric computing. IEEE, pp 82–85

Kim S, Whitehead Jr EJ (2006) How long did it take to fix bugs? In: Proceedings of the 2006 international workshop on Mining software repositories (MSR'06), pp 173–174

Lamkanfi A, Demeyer S, Giger E, Goethals B (2010) Predicting the severity of a reported bug. In: 2010 7th IEEE working conference on mining software repositories (MSR'10). IEEE, pp 1–10

Lee D, Rajbahadur GK, Lin D, Sayagh M, Bezemer C-P, Hassan AE (2020) An empirical study of the characteristics of popular minecraft mods. Empirical Software Engineering (EMSE'20) 25(5):3396–3429

Li J, Conradi R, Slyngstad OPN, Bunse C, Khan U, Torchiano M, Morisio M (2005) An empirical study on off-the-shelf component usage in industrial projects. In: International conference on product focused software process improvement. Springer, pp 54–68

Ling CX, Huang J, Zhang H (2003) Auc: a better measure than accuracy in comparing learning algorithms. In: Conference of the canadian society for computational studies of intelligence. Springer, pp 329–341

Ma W, Chen L, Zhang X, Feng Y, Xu Z, Chen Z, Zhou Y, Xu B (2020) Impact analysis of cross-project bugs on software ecosystems. In: Proceedings of the ACM/IEEE 42nd international conference on software engineering (ICSE'20), pp 100–111

Ma W, Chen L, Zhang X, Zhou Y, Xu B (2017) How do developers fix cross-project correlated bugs? a case study on the github scientific python ecosystem. In: 2017 IEEE/ACM 39th international conference on software engineering (ICSE'17). IEEE, pp 381–392

Marks L, Zou Y, Hassan AE (2011) Studying the fix-time for bugs in large open source projects. In: Proceedings of the 7th international conference on predictive models in software engineering (PROMISE'11), pp 1–8

McIntosh S, Kamei Y, Adams B, Hassan AE (2016) An empirical study of the impact of modern code review practices on software quality. Empirical Software Engineering (EMSE'16) 21(5):2146–2189

Menzies T, Marcus A (2008) Automated severity assessment of software defect reports. In: 2008 IEEE international conference on software maintenance (ICSM'08). IEEE, pp 346–355

Merilinna J, Matinlassi M (2006) State of the art and practice of opensource component integration. In: 32nd EUROMICRO conference on software engineering and advanced applications (EUROMICRO'06). IEEE, pp 170–177

Mockus A, Fielding RT, Herbsleb JD (2002) Two case studies of open source software development: Apache and mozilla. ACM Transactions on Software Engineering and Methodology (TOSEM'02) 11(3):309–346

Ohira M, Hassan AE, Osawa N, Matsumoto K (2012) The impact of bug management patterns on bug fixing: A case study of eclipse projects. In: 2012 28th IEEE international conference on software maintenance (ICSM'12). IEEE, pp 264–273

Panichella S, Bavota G, Di Penta M, Canfora G, Antoniol G (2014) How developers' collaborations identified from different sources tell us about code changes. In: 2014 IEEE international conference on software maintenance and evolution (ICSME'14). IEEE, pp 251–260

Rajbahadur GK, Wang S, Ansaldi G, Kamei Y, Hassan AE (2021) The impact of feature importance methods on the interpretation of defect classifiers. IEEE Transactions on Software Engineering (TSE'21)

Ray B, Kim M (2012) A case study of cross-system porting in forked projects. In: Proceedings of the ACM SIGSOFT 20th international symposium on the foundations of software engineering (FSE'12), pp 1–11

Ray B, Kim M, Person S, Rungta N (2013) Detecting and characterizing semantic inconsistencies in ported code. In: 2013 28th IEEE/ACM international conference on automated software engineering (ASE'13). IEEE, pp 367–377

Shihab E, Ihara A, Kamei Y, Ibrahim WM, Ohira M, Adams B, Hassan AE, Matsumoto K (2013) Studying re-opened bugs in open source software. Empirical Software Engineering (EMSE'13) 18(5):1005–1042

Stol K-J, Babar MA, Avgeriou P, Fitzgerald B (2011) A comparative study of challenges in integrating open source software and inner source software. Inf Softw Technol 53(12):1319–1336

Storey M-A, Zagalsky A, Figueira Filho F, Singer L, German DM (2016) How social and communication channels shape and challenge a participatory culture in software development. IEEE Transactions on Software Engineering (TSE'16) 43(2):185–204

Thongtanunam P, Hassan AE (2020) Review dynamics and their impact on software quality. IEEE Transactions on Software Engineering (TSE'20)

Thongtanunam P, McIntosh S, Hassan AE, Iida H (2016) Revisiting code ownership and its relationship with software quality in the scope of modern code review. In: Proceedings of the 38th international conference on software engineering (ICSE'16), pp 1039–1050

Van Den Berk I, Jansen S, Luinenburg L (2010) Software ecosystems: a software ecosystem strategy assessment model. In: Proceedings of the Fourth European conference on software architecture: companion volume, pp 127–134

Weimer W (2006) Patches as better bug reports. In: Proceedings of the 5th international conference on Generative programming and component engineering (GPCE'06), pp 181–190

Weiss C, Premraj R, Zimmermann T, Zeller A (2007) How long will it take to fix this bug? In: Fourth international workshop on mining software repositories (MSR'07). IEEE, pp 1–1

Xia X, Lo D, Wen M, Shihab E, Zhou B (2014) An empirical study of bug report field reassignment. In: 2014 software evolution Week-IEEE conference on software maintenance, reengineering, and reverse engineering (CSMR-WCRE'14). IEEE, pp 174–183

Xuan J, Jiang H, Ren Z, Zou W (2012) Developer prioritization in bug repositories. In: 2012 34th international conference on software engineering (ICSE'12). IEEE, pp 25–35

Zhang F, Khomh F, Zou Y, Hassan AE (2012) An empirical study on factors impacting bug fixing time. In: 2012 19th working conference on reverse engineering. IEEE, pp 225–234

Zhang H, Gong L, Versteeg S (2013) Predicting bug-fixing time: an empirical study of commercial software projects. In: 2013 35th international conference on software engineering (ICSE'13). IEEE, pp 1042–1051

Zhang Y, Yu Y, Wang H, Vasilescu B, Filkov V (2018) Within-ecosystem issue linking: a large-scale study of rails. In: Proceedings of the 7th international workshop on software mining, pp 12–19

Zhou B, Neamtiu I, Gupta R (2015) A cross-platform analysis of bugs and bug-fixing in open source projects: Desktop vs. android vs. ios. In: Proceedings of the 19th international conference on evaluation and assessment in software engineering (EASE'15), pp 1–10

Zimmermann T, Premraj R, Bettenburg N, Just S, Schroter A, Weiss C (2010) What makes a good bug report? IEEE Transactions on Software Engineering (TSE'10) 36(5):618–643

Affiliations

Jiahuei Lin[1] · Haoxiang Zhang[2] · Bram Adams[1] · Ahmed E. Hassan[1]

Jiahuei Lin
jhlin@cs.queensu.ca

Bram Adams
bram.adams@cs.queensu.ca

Ahmed E. Hassan
ahmed@cs.queensu.ca

[1] Software Analysis and Intelligence Lab (SAIL), Queen's University, Kingston, Ontario, Canada

[2] Centre for Software Excellence at Huawei Canada, Kingston, ON, Canada