Investigating Bug Report Changes in Bugzilla

Felipe Emerson de Oliveira Calixto, Franklin Ramalho, Tiago Massoni and José Manoel Ferreira
Departamento de Sistemas e Computação, Universidade Federal de Campina Grande, Campina Grande, Paraíba, Brazil

Keywords: Bug Report, Tracking Bug Report Systems, Bugzilla.

Abstract: Bug report change behavior in bug tracking systems may help pinpoint negligence or misunderstanding when submitters fill in bug report information. This study investigates bug report changes in several projects within Mozilla's Bugzilla to identify which fields in a report change the most, which bug profiles receive more changes and the relationship between these changes. We found that the most changed fields are *flagtypes.name* and *cc*. Reports are often modified when they indicate a valid bug, with medium to high priority and severity. Moreover, there are moderate to high correlations between changes in the following field pairs: *product-component*, *priority-severity*, and *platform-op_sys*. We believe these results are relevant to indicate which submitter's skills must be enhanced to improve the bug-tracking process.

1 INTRODUCTION

Users play an essential role in identifying bugs during a software lifecycle, as a system is not free of failures, even after its initial release. Typically, systems with a large number of users receive many bug reports daily. For example, Mozilla receives around 307 new reports daily (Fan et al., 2020). For tracking and monitoring the status of reported bugs, there are specific systems such as Bugzilla and JIRA, among others. Such systems allow the submitter, either the developer or end user, to create and track reports. These reports may describe requests for new features/improvements, but most of them are for bugs (Valdivia Garcia and Shihab, 2014).

For developers to find and fix the bug, the most helpful report contains valuable information usually described in the fields, such as *affected product*, *affected component*, *priority*, *severity*, *bug classification* and *bug type*, among others. Nevertheless, it is hard to guarantee that reporters will provide all the needed information; a few studies have found that relevant fields are often neglected or incorrectly filled (Bettenburg et al., 2007; Bettenburg et al., 2008). Incomplete reports can be due to a lack of knowledge or attention in the case of end users.

For example, Zimmerman et al. (Bettenburg et al., 2008) investigated the overall quality of bug reports. In one of the stages of their study, they conducted a survey focused on developers and reporters, seeking to identify, among other things, which features (i) have been previously reported, (ii) are the most difficult for the reporter to provide, and (iii) are considered most valuable by developers.

As a result, they identified a contrast between what developers consider most useful and what the authors provide, suggesting that this may be related to the difficulty of providing specific information. Bug report fields may not be filled or correctly informed because the person who fills a report is often an end user of the system who may need more technical knowledge regarding the various characteristics of a bug. Consequently, developers' understanding of the report and the time until the bug is resolved may be affected. Soltani (Soltani et al., 2020) found that the lack of crucial fields like steps-to-reproduce and stack traces can impact the bug report resolution time by up to 70%.

The bug-tracking process is subject to constant changes during the lifecycle. We found that bug reports have an average of 14.29 changes across their lifecycle. A bug report's life goes through numerous updates, from status changes (when the bug is confirmed or fixed, for example) to changes to correct information, such as the affected product. Other usual changes could happen to identify specific aspects of a bug. For instance, when a bug is blocking (blocking bug) the correction of others (blocked bug), when a bug is identified and validated (valid bug), or the bug described in the report is invalid (invalid bug). Also, when a bug report addresses a previously reported bug (duplicate bug) (Valdivia Garcia and Shi-

Calixto, F., Ramalho, F., Massoni, T. and Ferreira, J.
Investigating Bug Report Changes in Bugzilla.
DOI: 10.5220/0011847600003467
In *Proceedings of the 25th International Conference on Enterprise Information Systems (ICEIS 2023) - Volume 2*, pages 55-64
ISBN: 978-989-758-648-4; ISSN: 2184-4992

hab, 2014; Erfani Joorabchi et al., 2014; Rocha et al., 2016). Poorly reported bugs may affect those status changes, leading to rework and unnecessary cycles. Then, it is relevant to understand how these changes are related to the bug report fields often neglected or misfiled.

In this paper, we used Mozilla's Bugzilla historical data to investigate how report fields change over time, as they may indicate which fields are more likely to have inaccurate or insufficient information.

Furthermore, it helps us understand why some bug reports are changed more often.

We also investigate whether there is any relationship between field changes by analyzing the relationships between changes in pairs of fields. We consider the following research questions: (RQ1) Which fields change the most? (RQ2) What is the profile of the most changed bugs? (RQ3) Are there relationships between field changes in bug reports?

Bugzilla is one of the leading open-source software used/built for bug tracking and monitoring activities; it is used by companies/projects like Mozilla, RedHat, and Eclipse, among others. Briefly, users report bugs; these are triaged and assigned to a developer to fix them.

We chose Mozilla's Bugzilla as our dataset due to its availability and use in previous studies (Fan et al., 2020; Bettenburg et al., 2008; Valdivia Garcia and Shihab, 2014; Hooimeijer and Weimer, 2007; Erfani Joorabchi et al., 2014). The dataset has 690,817 bug reports, all of which are *RESOLVED*, and could be from any of Mozilla's projects. We identified changes in a total of 617 fields across the dataset.

Our study revealed that changes typically occur in two types of fields: custom (fields whose name starts with *cf_* and are created by Bugzilla administrators) or non-custom (fields that exist by default), with the latter undergoing most changes, with an average of 12.24 (vs. 2.04) changes during the bug life cycle.

Also, on average, the fields that change the most are *cc* (users registered to follow up on a given report) and *flagtypes.name* (used to request information from a user). The bug reports with the most changes are related to valid bugs (with *FIXED* resolution), with medium or high priority (*P1*, *P2*, or *P3*), and medium or high severity (*S1*, *S2*, *S3*, *blocker*, *critical*, or *major*). Some products, such as *Infrastructure & Operations*, have a lower mean of changes than others.

When a reporter makes a single update to a bug report, several fields can also be changed. We thus explored the occurrence of changes in pairs of fields in the same update, that is, the number of changes occurring for two fields in the same update, through correlation. As a result, we could check whether a field tends to have more modifications alone or in pairs with another. We found evidence that there are pairs of fields with a median-strong correlation between their change occurrences, such as: between *platform* (0.94) and *op_sys* (0.86), between *product* (0.64) and *component* (0.69), and between *priority* (0.28) and *severity* (0.61).

The main contributions of this work are: (i) characterizing changes in bug reports, including an in-depth exploration of fields not extensively studied in prior research; (ii) a sample dataset that has already been filtered, a selection of features, and accompanying scripts; (iii) results that may help other studies better estimate the use of change features in their models; and (iv) an initial investigation into the simultaneous correlation of changes between pairs of fields, which can be used to develop tools that indicate which other field(s) may tend to change in conjunction with a given field, thus resulting in more informative bug reports. Helping reporters to complete bug reports more comprehensively can facilitate developers in fixing the bug. Overall, this work aims to support better bug reporting practices and assist in improving bug resolution outcomes.

We organized this document as follows. Section 2 discusses some concepts related to the Bugzilla dataset. Section 3 discusses related work. Section 4 presents the methodology of this study. Section 5 presents the results achieved. Section 6 exposes potential threats to research validity. Finally, Section 7 presents the conclusions and possible lines of research for future work.

2 BACKGROUND

In this section, we detail the definition of the Bugzilla fields addressed in this work and how Bugzilla structures the changes history of a bug report. Anvik (Anvik et al., 2005) presents a general context of how an open bug repository works.

2.1 Bug Report Fields in Bugzilla

Due to Bugzilla's large number of fields, we selected some of the most studied fields (Fan et al., 2020; Bettenburg et al., 2008; Valdivia Garcia and Shihab, 2014; Hooimeijer and Weimer, 2007; Erfani Joorabchi et al., 2014; Rocha et al., 2016; Gupta and Sureka, 2014). Below, we have the name, description, and possible values of the main fields covered in this study:

- *id*: numerical field used to identify a single bug report uniquely;

- *history*: all change records (presented in more detail in subsection 2.2);
- *resolution*: current resolution status of a report. It can take one of the following values: *FIXED*, *INVALID*, *INCOMPLETE*, *DUPLICATE*, *WORKSFORME*, *WONTFIX*, *INACTIVE*, or *MOVED*. When a report has been confirmed as a bug and fixed, the developer will set the resolution to *FIXED*. When a report describes a bug that has already been reported earlier, the report is resolved as *DUPLICATE*. The rest of the values refer to bugs that are not reproducible;
- *product*: the product affected by the bug. In Mozilla, there are, for example, the Firefox and Thunderbird products;
- *component*: indicates the component affected by the bug. Each component belongs to a specific product, and a product can have multiple components. For example, the Firefox product has the Menus component, and the Core product has the JavaScript Engine component;
- *priority*: defines how important a bug is to being fixed compared to others. The priority has values ranging from *P1* to *P5*, where *P1* refers to a bug with maximum priority and *P5* to a bug with very low priority;
- *severity*: describes the severity of a bug. Currently, Mozilla's Bugzilla has two severity scales. The first scale has values from *S1* to *S4*, which go from catastrophic to trivial severity. In addition, severity may have the *N/A* value for reports where a severity classification does not apply, for example, when a report is of the enhancement type. The second scale is more self-explanatory, as each value indicates the degree of severity (*blocker*, *critical*, *major*, *normal*, *minor*, *trivial*, *enhancement*).

2.2 Structure of a Bug Report's History

Figure 1 shows a UML Class Diagram with the history structure used by Bugzilla to record changes to a bug report - we can see other attributes of a bug report in Bugzilla's documentation. In short, each bug report has a *history* that is an array, which can be empty (when no changes are registered previously) or can contain multiple objects of type *ChangeSet*. *ChangeSet* consists of two attributes: *who* contains the email of the user who made the change; and *when* is the date on which the change was made. In addition, a *ChangeSet* is composed of *changes*, an array of *Change* objects. A *ChangeSet* must include at least one *Change*. Lastly, a *Change* object consists of the attributes: *field_name* (name of the field that someone changed), *added* (the value that someone added), and *removed* (the value that someone removed).

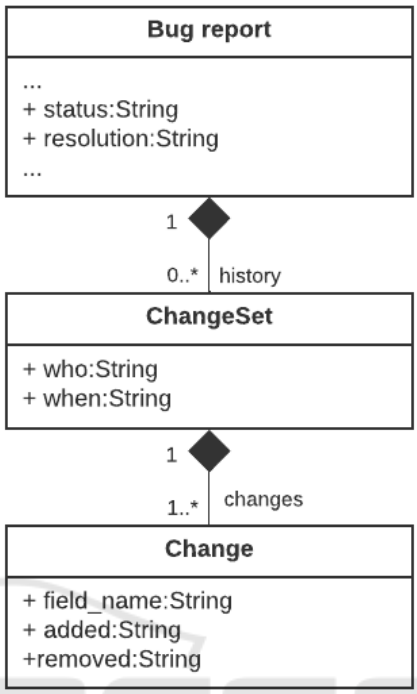

Figure 1: History structure.[1]

3 RELATED WORK

Zhang (Zhang et al., 2016) reviewed the literature about bug resolution and identified research lines on this topic.

The quality of bug reports has been the subject of several research studies. Zimmerman et al. (Bettenburg et al., 2008) built a tool to measure the quality of bug reports and suggest improvements to the reporter. Some work has focused on developing tools that help to increase the quality of bug reports: Song and Chaparro (Song and Chaparro, 2020) built BEE, a tool for structuring and analyzing bug reports, and Fazzini et al. (Fazzini et al., 2022) created EBug, a tool for assist reporters in writing steps-to-reproduce in mobile apps.

There is much research on applying models for classifying bug reports and/or predicting features. Lo et al.(Fan et al., 2020) designed a model to predict whether a bug report is valid. Hooimeijer and Westley (Hooimeijer and Weimer, 2007) built a model to predict when a bug will be triaged, given a certain amount of time. For this, they used three features

[1] https://wiki.mozilla.org/Bugzilla:BzAPI:Objects

related to changes: the number of severity changes, comment count, and attachment count. Shihab and Garcia (Valdivia Garcia and Shihab, 2014) also worked with prediction models to predict whether a bug is blocking or not, and they used the feature *priority has increased*, which tells if the priority has gone up after the initial report. Xiao et al. (Xiao et al., 2020) applied a deep neural network (DNN) model to predict duplicated bug reports.

Joorabchi et al. (Erfani Joorabchi et al., 2014) made a characterization of non-reproducible bug reports seeking to understand their frequency, nature, and cause of them. For this, they make a comparative analysis of the properties of non-reproducible bugs and their counterparts. Furthermore, they investigate their life-cycle patterns taking into account the changes in history in status and resolution. Regarding changes, Joorabchi's work only focuses on understanding the transitions that occur in status and resolution, while the entire focus of this work is to understand more generally (with more fields) the frequency of changes and the possible relationship between field changes.

Rocha et al. (Rocha et al., 2016) propose a study of bug workflows in Mozilla Firefox to understand developers' workflow better while dealing with bugs. To do so, they use the status' changes history of developing workflow graphs and using the resolution to compare workflows. Moreover, they compared workflows between developers with different levels of experience. Thus, this work focuses on analyzing only status changes. Gupta (Gupta and Sureka, 2014) used business process mapping tools and techniques to create a framework that generates runtime process maps from analyzing the changes history in bug reports - they used the fields: *status*, *resolution*, *assigned_to*, *qa_contact*, and *component*.

These studies either use some change features in their models or, by investigating the changes, focus on the status and resolution fields. In this study, we focus on the frequency of changes by considering more fields and the relationships between those changes, which serve as complementing evidence.

4 METHODOLOGY

This work aims to analyze the change history of bug reports reported in the Bugzilla database. Through an exploratory analysis of this dataset, the objectives are: (i) to identify which fields are most changed; (ii) the profile of the reports that have the most changes, and (iii) to determine whether there are relations between changes.

All study material is available at: https://github.com/felipeemerson/Bugzilla-mozilla-investigation.

4.1 Research Questions

The study was carried out in order to answer the following research questions:

RQ1. Which fields change the most?

RQ2. What is the profile of the most changed bugs?

RQ3. Are there relationships between field changes in bug reports?

4.2 Dataset

The dataset used in this study was Mozilla's Bugzilla due to its popularity and availability and because of several previous studies working with it (Fan et al., 2020; Bettenburg et al., 2008; Valdivia Garcia and Shihab, 2014; Hooimeijer and Weimer, 2007; Erfani Joorabchi et al., 2014). We defined the following filters:

- Report creation date range: between 01/01/2013 and 01/01/2022. It covers nine years, ensuring a good amount of bug reports;
- Status: *RESOLVED*. Using this status, we avoid getting current open bugs or invalid reports;
- Product: all. The dataset includes bug reports from several Mozilla projects.

In total, the dataset used has 690,817 bug reports.

4.3 Metrics

In this study, we apply the following metrics:

- *Total changes per bug report*. It allows us to check which bug reports change the most;
- *Percentage of bug reports that recorded at least one change per field*. The percentage of bug reports registered at least one modification in a given field. Provides clues about which changes happen more frequently;
- We reused the metric *total changes per field of a bug report*, which is used as a feature (in the severity field) in Hooimeijer and Westley's work (Hooimeijer and Weimer, 2007).

4.4 Procedure

We obtained the data used in this work in two stages: data collection and processing.

4.4.1 Data Collection

The data was collected using Bugzilla's REST API[2] using Python scripts. The API returns the data in JSON format. Due to the massive amount of data, the result was stored in the MongoDB database and is accessed using the mongo engine library.

4.4.2 Data Processing

In order to obtain the metrics values described in Section 4.2, we processed the data through Python scripts, and the results were written to JSON files. The files are used in notebook-type documents to produce the graphs and statistics to be analyzed. For this purpose, the pandas[3], matplotlib[4], and seaborn[5] libraries were used.

4.5 Fields

The fields explored in this study are:

1. ***id***: unique bug report identifier;
2. ***history***: the changes history;
3. ***resolution***: it indicates as the bug report was resolved;
4. ***product***: affected product;
5. ***severity***: bug severity level;
6. ***priority***: bug priority level.

The fields *id*, *history*, and *resolution* were downloaded between 06/03/2022 and 06/04/2022; While *product*, *severity* and *priority* were added between 07/19/2022 and 07/31/2022. The second part of downloads was necessary due to the later identification of their requirement with their final values to answer RQ2. Furthermore, we detected that a bug report, with id 1604167, now requires access authorization, which led it to be left out of the specific analyses involving the fields added later.

5 RESULTS

The results achieved with this study provide evidence to answer the research questions proposed in Section 4. We discuss this in this section.

[2]https://bmo.readthedocs.io/en/latest/api/index.html
[3]https://pandas.pydata.org/
[4]https://matplotlib.org/
[5]https://seaborn.pydata.org/

Table 1: Number of fields that registered changes by type.

Fields		Value	
Custom fields	Related to product	578 (93.68%)	544 (88.17%)
	Not related to product		34 (5.51%)
Non-custom fields		39 (6.32%)	

5.1 RQ1. Which Fields Change the Most?

To answer RQ1, we identify which fields were subject to at least one change in the entire dataset (Subsection 5.1.1). After checking that most fields are custom, we investigated their prevalence (Subsection 5.1.2). In Subsection 5.1.3, we present the general statistics of changes to understand better the results obtained. In Subsection 5.1.4, we present the answer to RQ1.

5.1.1 Total Fields with at Least One Change

As shown in Table 1, we identified 617 fields that registered at least one change in the dataset. Looking at the field names, we noticed that 578 fields start with the prefix *cf_*, used to identify custom fields, which Bugzilla administrators create to meet some demands that existing fields do not meet.

Among the custom fields, most are fields related to versions of a specific product and start with the prefixes *cf_status* or *cf_tracking*, e.g., *cf_status_firefox101*, *cf_status_thunderbird_103* and *cf_tracking_seamonkey237*. There are 39 non-custom fields, only 6% of the total.

5.1.2 Custom Fields that Have a Higher Percentage of Presence

Considering only the custom fields that are not related to product versions, we have highlighted in Table 2 that the *cf_last_resolved* field (last date on which the report was considered resolved) has 100% presence, meaning that it is updated at least once in every bug report in the dataset. Fields such as *cf_has_regression_range* (it says if a report has a regression interval), *cf_crash_signature* (it saves the fault signature), among others, have a presence below 2.48%, which means modifications involving these fields occur in very few reports. These low values may be due to the fact that only a developer can modify custom fields. Furthermore, most are related to a specific product version, being used for a short time. For example, Mozilla Firefox has new releases monthly.

5.1.3 Total Changes per Bug Report

Table 3 presents the values of the total changes by bug report for custom and non-custom fields and the two

Table 2: The five most frequent custom fields with changes.

Field	Presence percentage
cf_last_resolved	100%
cf_blocking_b2g	2.48%
cf_qa_whiteboard	1.68%
cf_has_regression_range	1.61%
cf_crash_signature	0.96%

Table 3: Statistics of the total changes.

	Mean	Median	SD	Max
Custom fields	2.04	1	2.42	94
Non-custom fields	12.24	8	15.20	1793
All fields	14.29	10	16.25	1796

types together. On average, a bug report has 14.29 changes over its lifecycle, with most changes being made to non-custom fields. Non-custom fields have a high standard deviation (SD) compared to custom fields, indicating that they have a higher value dispersion. As for the maximum values recorded, non-custom fields had up to 1793 changes in a single bug report, an exceptional outlier value compared to the mean and median. This difference between the number of changes can be explained by the factors already mentioned in subsection 5.1.2 (most custom fields are fields used for a short period) and changes in custom fields that occur in very few bug reports.

5.1.4 Total Changes per non-Custom Field

The previous subsection shows that non-custom fields cover most of the total changes. Table 4 shows that only three fields reached a non-zero median: *cc*, *resolution*, and *status*; these last two are fields with 100% presence in the entire base due to the restriction of all bug reports in the dataset to have the *RESOLVED status* and because it is necessary to inform the resolution field. So, apart from status and resolution, the fields with the highest percentages are *cc* (it is used for users to register and receive notifications about the report), *flagtypes.name* (it is used to ask a user for information), and *assigned_to* (it is filled in when a developer is assigned to fix a bug).

As for the maximum values of changes, two fields that indicate relationships between bugs appear among those with the highest maximum values: *depends_on* (list of bugs that block the current one) with 1770 and *blocks* (list of bugs that are blocked by the current one) with 353. Among the fields with the lowest maximum values, we can highlight *regressed_by* (list of bugs that introduced the current one) with only 5, which is another field that indicates relationships between bugs.

In summary, the *cc* and *flagtypes.name* fields stood out in the various scenarios, showing that they are among the most modified and present. For *cc*, the likely reason for the results is that users signing up to track bug reports is very common. As for *flagtype.name*, the request for information from one user to another is common for several reasons: a developer asking for new information about the described bug or a user asking a developer for analysis, among other situations. Moreover, in both cases, the more complex and/or urgent the bug is, the more users tend to participate in the bug. Consequently, more changes occur in those fields.

5.2 RQ2. What Is the Profile of the Most Changed Bugs?

Grouping changes by the final value of a field can give clues to understanding which values are related to more changes. For this subsection, we calculate statistics only using modifications in non-custom fields (because they are the ones with the most modifications, as seen in subsection 5.1). We use the *resolution*, *priority*, and *severity* fields to group the data. Due to base restriction, we do not use status, where all bugs have the final status *RESOLVED*. In addition, we perform grouping by the *product* field, considering the most popular products within the dataset.

5.2.1 Grouping by Final Resolution

Due to the download time range described in section 4.5, 06 (six) bug reports were removed from the analysis as they were reopened within the range and had their final resolutions removed.

Table 5 introduces the total changes grouped by the final resolution. As we can see, it shows that the resolution with the most changes is *FIXED* with 15.13 changes on average (this means that bug reports, where the last value registered in resolution was *FIXED*, have 15.13 changes on average). In comparison, *INACTIVE* and *WONTFIX* have around ten changes on average, while *DUPLICATE*, *INCOMPLETE* and *INVALID* have around 8. There is a difference of up to 50% between the average modifications from *FIXED* to the others. This result may indicate that invalid reports tend to be identified with a lower degree of change or that valid bug reports have a longer life cycle, given that after a bug is identified, its report can still present changes that help in the correction of the same.

The highest modification values were recorded in *WONTFIX* with 1793, *INVALID* with 1692, and *FIXED* with 1369. Were these values recorded in the

Table 4: Results summary of the total changes in non-custom fields.

Highest Means	Highest Medians	Maximum Values	Lowest Maximum Values	Most Present Fields
flagtypes.name (2.70)	*cc* (1)	*depends_on* (1,770)	*attachments.isprivate* (4)	*resolution* (100%)
cc (2.34)	*resolution* (1)	*cc* (773)	*restrict_comments* (4)	*status* (100%)
status (1.34)	*status* (1)	*flagtypes.name* (547)	*regressed_by* (5)	*cc* (77.43%)
resolution (1.16)	*product* (0)	*comment_tag* (519)	*bug_mentor* (5)	*flagtypes.name* (48.30%)
comment_tag (0.69)	*component* (0)	*blocks* (353)	*op_sys* (5)	*assigned_to* (34.90%)

Table 5: Total changes grouped by final resolution.

Value	Mean	Median	SD	Max
FIXED	15.13	10	17.88	1369
INVALID	7.86	6	10.38	1692
INCOMPLETE	8.24	6	8.74	860
DUPLICATE	8.18	6	7.32	362
WORKSFORME	9.97	8	9.87	580
WONTFIX	10.39	7	14.86	1793
INACTIVE	10.42	8	12.62	247
MOVED	8.74	6	8.50	99

bug reports with id 838081[6], 950073[7], and 1243581[8], respectively. By examining them, we found they were used as a hub for bug reports related to a specific project. The bug report with id 838081 centralized a Product Backlog (list of requirements) of Firefox's Metro interface, which was never released. The report with id 950073 centralized Firefox Desktop reports. Furthermore, the report with id 1243581 was related to the Stylo project. In this way, related reports were concentrated using the *depends_on* field, and new bug reports were added and removed if they were resolved. In summary, none of the three bug reports were linked to any specific bug, and there may be more bug reports in the same situation.

5.2.2 Grouping by Final Priority

Concerning the total changes grouped by the final *priority*, we can see, In Table 6, that the highest averages of changes by *priority* are *P1* with 19.11 (bug reports with *P1* as final value registered in *priority* have an average of 19.11 changes), *P2* with 17.69, and *P3* with 15.40, but these values have a standard deviation of up to 20.71 changes. Thus, the median can be a better value to illustrate how these changes occur, where priorities *P1*, *P2*, and *P3* have a median between 11 and 13 modifications, while *P4* and *P5* have 7 and 6, respectively. There is a difference in total changes of up to about twice between low-priority and high-priority bug reports. Furthermore, as the *priority* field is not required, there is a high number of bug reports (478542) that do not have a defined *priority* (represented by the value "–"), which have a median of 8 changes.

Table 6: Total changes grouped by final priority.

Value	Mean	Median	SD	Max
–	11.41	8	14.28	1793
P1	19.11	13	20.71	544
P2	17.69	13	19.61	595
P3	15.40	11	17.21	1003
P4	8.44	7	9.06	860
P5	8.73	6	10.50	797

The results suggest that reports with medium or high priority tend to present more changes than reports with low priority.

5.2.3 Grouping by Final Severity

Unlike *priority*, whose values are on a scale from *P1* to *P5*, *severity* has two different scales, one from *S1* to *S4* and another one whose classification is made by categories (*critical*, *normal*, *enhancement*, *blocker*, *major*, *minor* and *trivial*). Apparently, the *S1* to *S4* scale was added later, and both coexist, leaving it up to the user to choose.

Analyzing the values in Table 7, we have that for the first scale, *severity S4* has the lowest average of changes with 8.39, while *S1*, *S2*, and *S3* have 16.68, 19.09, and 15.32 changes, respectively. Considering the other scale, *blocker*, *critical*, and *major*, which are the highest levels of *severity*, also have the highest averages of changes with 17.19, 15.38, and 15.87, respectively.

As we can see, reports with medium to high severity tend to have more modifications than reports with low severity.

[6]https://bugzilla.mozilla.org/show_bug.cgi?id=838081
[7]https://bugzilla.mozilla.org/show_bug.cgi?id=950073
[8]https://bugzilla.mozilla.org/show_bug.cgi?id=1243581

Table 7: Total changes grouped by final severity.

Value	Mean	Median	SD	Max
–	8.81	7	6.98	190
N/A	12.12	9	11.48	300
S1	16.68	11	14.05	83
S2	19.09	15	14.67	129
S3	15.32	13	11.32	348
S4	8.39	6	7.64	157
blocker	17.19	11	20.03	530
critical	15.38	11	13.97	254
enhancement	11.53	11	3.32	23
major	15.87	11	16.64	244
minor	11.72	9	9.31	137
normal	12.30	8	15.81	1793
trivial	12.05	9	21.50	842

Table 8: Total changes grouped by final product (top 10 most popular products).

Value	Mean	Median	SD	Max
Core	14.53	10	18.01	1369
Firefox	11.17	8	13.55	767
Firefox OS Graveyard	13.90	9	16.40	576
Testing	10.80	7	13.21	326
DevTools	14.18	10	14.94	423
Infrastructure Operations	6.15	4	6.27	335
Toolkit	14.34	10	16.43	530
Firefox for Android Graveyard	13.95	10	14.82	433
Thunderbird	11.53	9	11.39	280
Firefox Build System	14.08	10	15.80	545

5.2.4 Grouping by Final Product

In the dataset used in this study, there are 163 different products, so we chose to group the 10 (ten) most popular ones, the products with the highest number of reports identified in the dataset. Observing Table 8, we have the product *Infrastructure & Operations* with the lowest average of all, which is about 2 (two) times lower compared to the other products. Therefore, it may be possible that depending on the context of a product, bug reports that affect it have fewer changes than reports of other products.

5.3 RQ3. Are There Relations Between Field Changes in Bug Reports?

As it is possible to register several changes at once, the presence of a change may appear simultaneously with another change, or the occurrence of a change may be related to another change. Thus, there may be correlations between the total of changes and the simultaneous occurrence of changes between pairs of fields, i.e., when we have an occurrence of changes in both two fields in the same *ChangeSet*. In this context, we consider the following fields for analysis: *status*, *resolution*, *assigned_to*, *product*, *component*, *priority*, *severity*, *summary*, *platform*, and *op_sys*. The choice was made because they are fields explored in other studies (Fan et al., 2020; Bettenburg et al., 2008; Valdivia Garcia and Shihab, 2014; Hooimeijer and Weimer, 2007; Erfani Joorabchi et al., 2014; Rocha et al., 2016; Gupta and Sureka, 2014).

Correlation between *status* and *resolution*. Figure 2 shows a very high correlation of 0.85 between the number of changes in resolution and status fields, which would be expected. Whenever the resolution field changes, the status field will also change together. That is because a change in resolution only occurs in two situations: (1) when the bug has been resolved, and then the status will change to *RESOLVED* along with resolution; (2) when the bug is reopened and then removes the resolution value and changes the status to *REOPENED*. However, not always when the status changes resolution will change along because the status can change in other situations like when a bug report is created (*NEW*) or assigned (*ASSIGNED*) to a developer.

Correlation between *platform* and *op_sys*. There is a very high correlation of 0.82 between *platform* and *op_sys*, where *platform* refers to the device architecture (x86, ARM, etc.), while *op_sys* refers to the operating system. It is still possible to explore the correlation of simultaneous occurrences of these two fields with their total number of changes. Figure 3 shows that the correlation between the total *platform* changes and the simultaneous occurrence between the two is 0.94, even higher than the previous correlation. The correlation between *op_sys* and the simultaneous occurrence is 0.86, which is still high. So, *platform* changes rely more on *op_sys* than vice versa.

Correlation between *product* and *component*. The correlation between the total *product* number of changes and total *component* changes is 0.41, considered a moderate correlation. However, Figure 4 shows that the correlation between the number of changes that co-occur in the *component* and *product* fields is 0.69, and of the *product* with the simultaneous occurrence is 0.64. That is a high correlation between a change in the *component* field coinciding with *product* and vice versa.

Correlation between *priority* and *severity*. The correlation between the number of *priority* changes and the number of *severity* changes individually is 0.18, a shallow value. Nevertheless, as seen in Figure 5, when the simultaneous occurrence is considered, there is a high correlation of 0.61 between *severity* and its simultaneous occurrences and a low correlation between *priority* and its simultaneous occurrences. *Severity* occurs more often, accompanied by *priority* than the reverse.

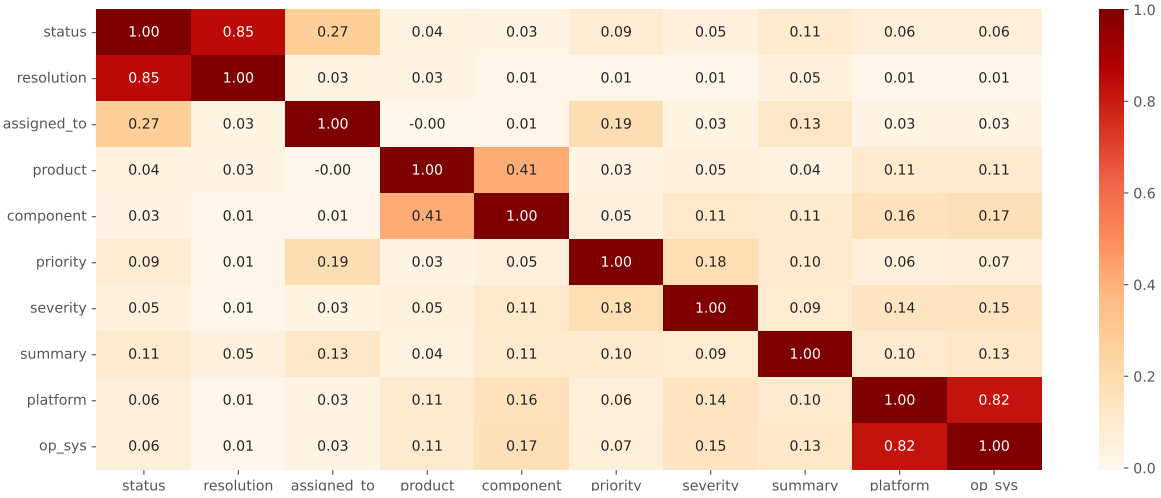

Figure 2: Correlations between the number of field changes.

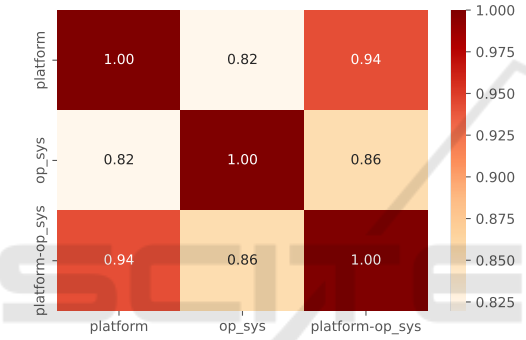

Figure 3: Correlation between *platform*, *op_sys*, and their simultaneous occurrences.

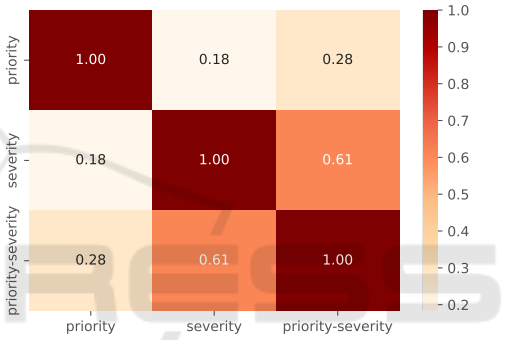

Figure 5: Correlation between *priority*, *severity*, and their simultaneous occurrences.

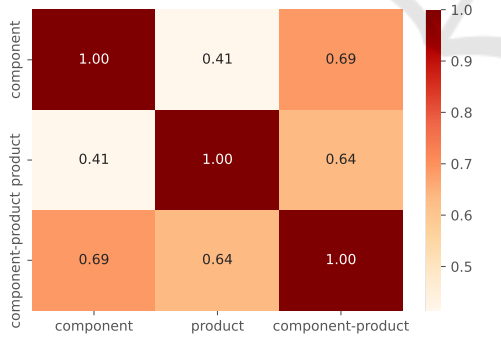

Figure 4: Correlation between *product*, *component*, and their simultaneous occurrences.

6 THREATS TO VALIDITY

Internal Threats to Validity. The procedures performed in the collection and/or processing phase may be possible sources of errors. Therefore, we checked each procedure more than once. Another factor is that the data is constantly being updated, possibly reopening bug reports and, consequently, having new updates. Considering this, we downloaded change histories from the bug reports in a short period (3 days). However, it was later necessary to download more fields, which led to 6 bugs being reopened in between.

External Threats to Validity. This study focused only on Mozilla's Bugzilla dataset, so the results may not be valid for other datasets, whether they are open source or not.

7 CONCLUSIONS

The present study identified that about 85% of the modifications occur in non-custom fields. Except for *cf_last_resolved*, custom fields have less than 3% presence in bug reports changes. Therefore, they may not be promising as features. *Cc* and *flagtypes.name* fields are the most modified. Among these, only *cc* was used in studies (Valdivia Garcia and Shihab, 2014), and (Erfani Joorabchi et al., 2014).

We have found that bug reports in Mozilla's

Bugzilla tend to have a higher average of changes when they are valid bugs with medium-high priority and/or medium-high severity. This information can help developers better estimate the effort needed to track and fix bugs.

Concerning the relations between changes, we identified that the correlation between field pair modifications could be promising. For example, *platform* and *op_sys* fields present a robust correlation (0.94 and 0.86, respectively) between the simultaneous occurrence of changes in them. Most *platform* changes occurred together with *op_sys* changes and vice versa. The *product* and *component* fields show a moderate correlation in both cases.

Future work could evaluate the use of the *flagtypes.name* field as a feature in models or tools. In addition, researchers could investigate which other fields and their respective values affect the amount of change in bug reports. A comparative study involving multiple datasets could further generalize the results. Future research could explore additional fields to identify new promising pairs that correlate with changes and their influence on bug report resolution.

REFERENCES

Anvik, J., Hiew, L., and Murphy, G. C. (2005). Coping with an open bug repository. In *Proceedings of the 2005 OOPSLA Workshop on Eclipse Technology EXchange*, eclipse '05, page 35–39, New York, NY, USA. Association for Computing Machinery.

Bettenburg, N., Just, S., Schröter, A., Weiß, C., Premraj, R., and Zimmermann, T. (2007). Quality of bug reports in eclipse. In *Proceedings of the 2007 OOPSLA Workshop on Eclipse Technology EXchange*, eclipse '07, page 21–25, New York, NY, USA. Association for Computing Machinery.

Bettenburg, N., Just, S., Schröter, A., Weiss, C., Premraj, R., and Zimmermann, T. (2008). What makes a good bug report? In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, SIGSOFT '08/FSE-16, page 308–318, New York, NY, USA. Association for Computing Machinery.

Erfani Joorabchi, M., Mirzaaghaei, M., and Mesbah, A. (2014). Works for me! characterizing non-reproducible bug reports. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, page 62–71, New York, NY, USA. Association for Computing Machinery.

Fan, Y., Xia, X., Lo, D., and Hassan, A. E. (2020). Chaff from the wheat: Characterizing and determining valid bug reports. *IEEE Transactions on Software Engineering*, 46(5):495–525.

Fazzini, M., Moran, K. P., Bernal-Cardenas, C., Wendland, T., Orso, A., and Poshyvanyk, D. (2022). Enhancing mobile app bug reporting via real-time understanding of reproduction steps. *IEEE Transactions on Software Engineering*, pages 1–1.

Gupta, M. and Sureka, A. (2014). Nirikshan: Mining bug report history for discovering process maps, inefficiencies and inconsistencies. In *Proceedings of the 7th India Software Engineering Conference*, ISEC '14, New York, NY, USA. Association for Computing Machinery.

Hooimeijer, P. and Weimer, W. (2007). Modeling bug report quality. In *Proceedings of the Twenty-Second IEEE/ACM International Conference on Automated Software Engineering*, ASE '07, page 34–43, New York, NY, USA. Association for Computing Machinery.

Rocha, H., de Oliveira, G., Valente, M. T., and Marques-Neto, H. (2016). Characterizing bug workflows in mozilla firefox. In *Proceedings of the XXX Brazilian Symposium on Software Engineering*, SBES '16, page 43–52, New York, NY, USA. Association for Computing Machinery.

Soltani, M., Hermans, F., and Bäck, T. (2020). The significance of bug report elements. *Empirical Software Engineering*, 25:5255–5294.

Song, Y. and Chaparro, O. (2020). Bee: A tool for structuring and analyzing bug reports. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2020, page 1551–1555, New York, NY, USA. Association for Computing Machinery.

Valdivia Garcia, H. and Shihab, E. (2014). Characterizing and predicting blocking bugs in open source projects. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, page 72–81, New York, NY, USA. Association for Computing Machinery.

Xiao, G., Du, X., Sui, Y., and Yue, T. (2020). Hindbr: Heterogeneous information network based duplicate bug report prediction. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*, pages 195–206.

Zhang, T., Jiang, H., Luo, X., and Chan, A. T. (2016). A literature review of research in bug resolution: Tasks, challenges and future directions. *The Computer Journal*, 59(5):741–773.