Are These Bugs Really "Normal"?

Ripon K. Saha, Julia Lawall, Sarfraz Khurshid, Dewayne E. Perry

Are These Bugs Really "Normal"?

Ripon K. Saha* Julia Lawall† Sarfraz Khurshid* Dewayne E. Perry*
*The University of Texas at Austin, USA
†Inria/LIP6/UPMC/Sorbonne University, France
ripon@utexas.edu, Julia.Lawall@lip6.fr, {khurshid, perry}@ece.utexas.edu

***Abstract*—Understanding the severity of reported bugs is important in both research and practice. In particular, a number of recently proposed mining-based software engineering techniques predict bug severity, bug report quality, and bug-fix time, according to this information. Many bug tracking systems provide a field "severity" offering options such as "severe", "normal", and "minor", with "normal" as the default. However, there is a widespread perception that for many bug reports the label "normal" may not reflect the actual severity, because reporters may overlook setting the severity or may not feel confident enough to do so. In many cases, researchers ignore "normal" bug reports, and thus overlook a large percentage of the reports provided. On the other hand, treating them all together risks mixing reports that have very diverse properties. In this study, we investigate the extent to which "normal" bug reports actually have the "normal" severity. We find that many "normal" bug reports in practice are not normal. Furthermore, this misclassification can have a significant impact on the accuracy of mining-based tools and studies that rely on bug report severity information.**

***Index Terms*—Bug Severity, Bug Tracking System, Mining Software Repositories**

I. Introduction

Bug tracking systems are among the most frequently used resources for research in mining software repositories [1]–[8]. They are also often used in developing new techniques for automated software engineering such as automatic bug triaging [9], bug assignment [10], bug-fix time prediction [11], [12], severity prediction [13], bug prioritization [14], and bug localization [15], [16]. A critical element of much of this work is understanding the importance of the bug reports found in these bug tracking systems. As this information is difficult to accurately infer, and may depend on the priorities and point of view of the bug reporter, studies typically rely on the "severity" label provided in the bug report [17]. While the labels vary by project, they typically amount to some variant of *Severe*, *Normal*, and *Minor*.

In many bug tracking systems, Normal is provided as the default. This may raise questions about the validity of a Normal severity label. Indeed, the person who files a bug report may be an ordinary user who has no expertise in the implementation of the affected software, or even no technical expertise at all. Such a person may find it difficult to accurately assess the severity of a bug. Thus, the bug reporter might not fill in the severity field, leaving it at its default Normal value. As a result, studies that use the severity field to investigate if there exists any relationship between bug severity and factors such as bug-fix time, amount of discussion, etc. are open to criticism that the results found in the Normal case may be invalid, as Normal may not reflect the actual severity. Simply excluding the Normal reports, however, may distort the results in the opposite direction, if the Normal reports represent a large percentage of the available data.

These issues have been highlighted in a number of research studies. For example, in their two studies of severity prediction, Lamkanfi et al. [13], [18] excluded all the normal bugs stating: "*In our case, the normal severity is deliberately not taken into account. First of all because they represent the grey zone, hence might confuse the classifier. But more importantly, because in the cases we investigated this "normal" severity was the default option for selecting the severity when reporting a bug and we suspected that many reporters just did not bother to consciously assess the bug severity"*. Similarly, Tian et al. [19] excluded Normal bugs stating: "*Following the work of Lamkanfi et al., we do not consider the severity label normal as this is the default option*". Along the same lines, in the submission of our previous work on "long lived bugs" [17], when we drew our conclusions that most long lived bugs were important and adversely affected users' normal working experience, all three reviewers expressed their concerns:

Reviewer 1: *"since you used data from the severity field, I would suggest to discuss the fact that the level of this field could be somewhat subjective."*

Reviewer 2: *"In most cases, 'Normal' is the 'default' value of the severity, thus most of the users reporting a bug leave the default value since they simply don't know or are not interested in precisely defining the value."*

Reviewer 3: *"Firstly we have to agree with the finding that eclipse severity is meaningful. If it is then you can cite other work that shows it to be meaningful, otherwise this claim does not hold up."*

A researcher is thus faced with a dilemma: either include information that may be unreliable, or discard potentially valuable information. To the best our knowledge no study has investigated either the amount of noise in the severity data (except *w.r.t.* enhancements [2], [20]) or the amount of value in this information.

In this paper, to better understand how severity information can be used, we investigate the following hypotheses, summarizing the apparent current consensus, as reflected by the above citations:

H1: Normal bugs do not reflect the actual severity level.

H2: Bug reports are mislabeled as Normal because reporters do not bother to change the default (Normal) severity.

Furthermore, we investigate the reasons for these problems and their impact in a representative software-engineering application. Our analysis is carried out in the context of four systems from the Eclipse product family. These are open source systems, that have publicly available bug databases, and that have been used in a number of previous software engineering studies [13], [17]–[19]. Our findings indicate that:

- Around 80% of the bugs reported in the studied software projects are classified as Normal. Excluding them from any automatic software engineering techniques could substantially distort the results.
- A manual reclassification of 500 Normal bugs in the studied software projects by pairs of students showed that 65% of the Normal bugs are not normal. Indeed, almost 25% of the Normal bugs are severe. These results support Hypothesis 1.
- Contradicting Hypothesis 2, we find the main reason for misclassifications in the Normal bugs is not that it is the default severity level. Rather, this field is very subjective and thus users may follow different criteria. Indeed, the pairs of students provided different opinions for more than half of the Normal bugs. We provide a taxonomy of the most common rationales used in these dissimilar assessments.
- The presence or absence of Normal bugs in training and test sets can significantly affect the actual and measured effectiveness of automatic software engineering techniques that rely on bug severity information. In our experiment with a basic bug severity predictor, we find that misclassification in the training data can reduce the accuracy of the severity prediction considerably. On the other hand, a tool accuracy excluding Normal bugs from both training and testing data is likely to be an overestimation if the tool is intended to be used on unlabeled data containing Normal bugs.

We conclude that while the classification of Normal reports is not very accurate, excluding them from software engineering studies can significantly distort the results.

The rest of this paper is organized as follows. Section II describes bug tracking systems and the various relevant features that they provide. Section III presents our research questions and our dataset. Sections IV through VII consider our research questions. Finally, Section VIII analyzes threats to validity, Section IX presents related work, and Section X concludes.

II. Background

We first briefly present the notion of a bug tracking system, and the levels of severity that such systems commonly use.

A. Bug Tracking System

Generally project stakeholders maintain a bug database for tracking the bugs found in their projects. A bug database may collect bug reports from developers, testers, or ordinary users, according to the policies of the project. Widely used online bug tracking systems include Bugzilla, JIRA, and Mantis.[1] Different bug tracking systems may have different data structures and follow different life cycles of bugs. In this paper, we focus on data extracted from Bugzilla.

Any person who has a Bugzilla account for a given project can post a change request. A change request could be either a bug report or a request for a feature enhancement. In Bugzilla, both are represented similarly and are referred to as "bugs," with the exception that for a feature enhancement, the severity field is set to "enhancement". Generally, the bug reporter provides a bug summary, a bug description, the names of the suspected product and component, and an indication of the bug's severity. The bug reporter also specifies the software version, the platform and operating system where the bug was encountered, so that developers can easily reproduce the bug.

TABLE I
Bug Severity Values used by Eclipse

Severity	Definition
Blocker	Blocks development and/or testing work. No workaround exists.
Critical	Crashes, loss of data, severe memory leak.
Major	Major loss of function.
Normal	Regular issue, some loss of functionality under specific circumstances.
Minor	Minor loss of function, or other problem where easy workaround is present.
Trivial	Cosmetic problem such as misspelled words or misaligned text.
Enhancement	Request for enhancement.

B. Bug Severity

Each project that uses Bugzilla can define its own severity levels. Since we study the projects from the Eclipse product family, we discuss the severity levels defined by Eclipse community. According to Eclipse Bugzilla documentation,[2] the severity level can be one of the following: Blocker, Critical, Major, Normal, Minor, Trivial, or Enhancement. These values are intended to describe the impact of the reported bug on the operation of the software. The definitions of these values provided by the Eclipse documentation are given in Table I.

At the start of the bug fixing process, each project or component team leader triages `NEW` bug reports to determine whether the bug is really a bug and if the provided information is correct. In case of any inconsistencies, the bug triager can correct them or request more information from the person who originated the report. For example, during triaging, a bug may be moved to another component/product or the triager can adjust the severity level. The developer who fixes a bug can also adjust the severity level. If the severity field is changed, the Bugzilla report contains the history of the assigned values.

[1] http://www.bugzilla.org, https://www.atlassian.com/software/jira, https://www.mantisbt.org

[2] http://wiki.eclipse.org/Eclipse/Bug_Tracking

III. Study Setup

A. Research Questions

Our study investigates the following research questions:

RQ1. What proportion of the bugs are Normal in the bug repository?

Motivation: The first question of any empirical study is how large is the population that we want to study. Indeed, if the population is small, there may be little reason to worry about it. For this study, our population of interest is the set of bug reports having the severity level Normal.

RQ2. What proportion of bug reports classified as Normal are actually "normal"?

Motivation: This is one of the main research questions of our study. The fewer bug reports classified as Normal that are actually "normal," the more effect this will have on the validity of any study that somehow depends on the bug report severity classification. This research question also addresses Hypothesis 1: *Normal bugs may not represent the actual severity level*.

RQ3. What are the main sources of misclassifications?

Motivation: To reduce misclassifications, we first need to understand the reasons behind it. Delineating the common reasons for misclassifications will help researchers or practitioners deal with the problem more systematically. This research question also addresses Hypothesis 2: *Bug reports are mislabeled as Normal because reporters do not bother to change the default (Normal) severity level.*

RQ4. Can misclassification or exclusion of Normal bugs affect previous study results?

Motivation: We investigate whether the noise in Normal bugs can affect tool results. If there is no impact, we would have little reason to worry about the issue.

B. Subject Systems

Our study focuses on four open-source projects, JDT, CDT, PDE, and Platform, from the Eclipse product family.[3] JDT and CDT provide an Integrated Development Environment based on the Eclipse platform for developing Java applications and for developing C and C++ applications, respectively. The Plug-in Development Environment (PDE) provides tools to create, develop, test, debug, build and deploy Eclipse plug-ins, fragments, features, update sites and Rich Client Platform (RCP) products. Finally, the Eclipse Platform defines the set of frameworks and common services that make up the infrastructure required to support the use of Eclipse. These projects are widely used in the real world, and have also been extensively used in software engineering research [13], [17], [18]. Furthermore, although these projects belong to the same product family, they are from various domains.

We have used Lamkanfi et al.'s [21] bug dataset, obtained from the Eclipse Bugzilla database,[4] to obtain the bug information associated with these projects. This dataset includes all the bug reports and their histories from the project inception to March 2011 for these four projects. Table II describes the dataset in more detail. Although this dataset is few years old, it was part of the MSR data showcase in 2013 and similar datasets have been used in many studies [13], [17]–[19], [22]

TABLE II
Data Set

System	#Change Requests (Bugs+Enh.)	#Bugs	#Enhancements	#Bugs Fixed
JDT	46,308	38,520	7,788	18,873
CDT	14,871	12,854	2,017	7,260
PDE	13677	11,958	1,719	6,854
Platform	90,691	78,120	12,571	33,738
Total	165,547	141,452	24,095	66,725

IV. Proportion of Normal Bugs

In this section, we investigate our first research question: *What proportion of bugs have Normal severity level?*

A. Methodology

A straightforward methodology would be to just compare the number of reports labelled Normal with the total number of reports. However, a bug tracking system may contain many invalid and duplicate bug reports, as well as feature requests or enhancements. There are also some bug reports that developers think are not worth fixing. In Bugzilla, the *status* and *resolution* fields together keep track of the current status of each bug. More specifically, the status field holds at most one of the values: `UNCONFIRMED`, `CONFIRMED`, `IN_PROGESS`, `RESOLVED`, and `VERIFIED`. The resolution field holds at most one of the values: `FIXED`, `INVALID`, `WONTFIX`, `DUPLICATE`, and `WORKSFORME`. Using this information, we extracted all the unique and valid bug reports, specifically those whose status field was set to either `RESOLVED` or `VERIFIED` and the resolution field was set to `FIXED`. We note that, as our bug reports date from 2011 at the latest, almost all of the reports have either been classified as uninteresting (`INVALID`, `WONTFIX`, `DUPLICATE`, or `WORKSFORME`) or have been fixed. We also removed all the reports marked as enhancements. We use the resulting set of reports in this and all of the subsequent research questions. Then, we counted all the bugs for each severity level.

B. Results

Table III provides detailed results regarding severity. For all of the considered systems, Normal is the dominant severity category, with 78-82% of the bug reports. The next most dominant category is Major, representing only 8-10% of the reports. Blocker bugs are rarest, at around 1%. The proportions of other types of bugs (Critical, Minor, and Trivial) are between 2% and 4% in most cases. Our results suggest that any research based on bug severity that ignores Normal bugs faces a severe threat to validity, since a large percentage of bug reports would likely not be taken into account.

[3]http://www.eclipse.org/jdt, https://eclipse.org/cdt, http://www.eclipse.org/pde, https://projects.eclipse.org/projects/eclipse.platform

[4]https://bugs.eclipse.org/bugs/

TABLE III
PROPORTION OF BUGS BY SEVERITY

System	Blocker	Critical	Major	Normal	Minor	Trivial	Total
JDT	116	572	1,647	14,856	1,090	592	18,873
	0.6%	3.0%	8.7%	78.7%	5.8%	3.1%	100%
CDT	83	155	698	5,946	288	90	7,260
	1.1%	2.1%	9.6%	81.9%	4.0%	1.2%	100%
PDE	64	220	567	5,631	246	126	6,854
	0.9%	3.2%	8.3%	82.2%	3.6%	1.8%	100%
Platform	424	1,306	3,535	26,289	1,245	939	33,738
	1.3%	3.9%	10.5%	77.9%	3.7%	2.8%	100%

V. ACTUAL SEVERITY OF "NORMAL"-LABELED BUGS

In the previous section, we saw that a large proportion of bugs are classified as Normal. However, we do not yet know if these Normal bugs are really normal according to the Eclipse Bugzilla definition. In this section, we investigate the second research question: *What proportion of bugs having Normal severity level is actually normal?*

A. Methodology

Since to the best of our knowledge, there is no clean dataset of bug reports that have actual severity levels, classifying bug reports using automated machine learning techniques would likely be inaccurate. Therefore, we conduct a manual investigation of their actual severity. Our methodology takes into account the fact that bug severity may be subjective, as well as the high cost of doing such an analysis.

1) Design: The severity field represents the impact of a given bug on the operation of the software, and thus it may be subjective. Indeed, even the Eclipse documentation mentions that the bug reporter's perspective on the severity can depend on how the bug reporter wants to use the software.[5] Therefore, to get reliable results, we have each bug report assessed by multiple users. In such a study, the cost depends on two factors: 1) the number of bug reports to be assessed, and 2) how many assessments are made.

To keep the cost reasonable, we made two decisions. First, we randomly selected a sample of 500 bug reports labelled Normal, representing 125 bug reports from each project, from within the last five years of our dataset, *i.e.*, from 2006 to 2011. Second, we recruited a group of assessors, such that each report would have at least two assessors. If the two assessors had different opinion about a given bug report, we analyzed both the report and the assessments to make a decision.

2) Users/Assessors Selection: All the assessors in our study are either graduate or undergraduate students of the University of Texas at Austin. We sent a general email to all the students in the senior "Software Engineering" class and to some graduate students in the software engineering track of the Electrical and Computer Engineering department. Good programming knowledge and substantial experience in working with the Eclipse IDE were requirements to participate in the study. Based on these criteria, we selected 6 graduate students and 4 senior undergraduate students. 9 of them have experience in industry either as an intern or as a full-time programmer. Table IV gives the students' qualifications in more detail.

[5]http://wiki.eclipse.org/Eclipse/Bug_Tracking

TABLE IV
STUDENT'S QUALIFICATIONS

Description	**Mean**	**Median**	**Min**	**Max**
Coding Experience (in Years)	6.1	5.0	3.0	13.0
Experience in Java (in Years)	5.0	4.0	2.0	11.0
Experience with Eclipse (in Years)	4.6	3.5	2.0	11.0
Industrial Experience (in Months)	10.0	9.0	2.0	24.0

3) Procedure: Our study was divided into two sessions: training and assessment. In the training session, we conducted a 30-minute tutorial. The tutorial included:

1) Providing a brief overview of our study,
2) Explaining a real Eclipse bug report and giving a brief overview of Bugzilla,
3) Showing how to submit a bug report in Eclipse, to show that Normal is the default severity level,
4) Explaining the definition of each severity level from the Eclipse documentation (Table I),
5) Showing a representative example in each severity level to deepen their understanding about bug severity,
6) Explaining the structure of the expected feedback.

We divided the 500 bug reports into five sets of 100 bug reports each. Sets 1-4 had 100 bug reports from the same project and Set 5 had 25 bug reports from each of the four projects. This strategy allowed most of the students to focus on a single project. We then assigned the students to five groups, pairing a graduate student with an undergraduate, when possible. The group members were not informed of each other's identity. Then we assigned each set of bug reports to a group randomly. The students were given 10 days to complete the assessments. We recommended to the students that they carefully read each part of the bug report, including at least the bug summary, the bug description, and the developers' comments, to make their decision. All of the students completed their task. After the study, each student was rewarded with a *$50 Amazon Gift Card*.

We note that the students have access to more information than the original bug reporter, as the students have access to the comments that were added after the original bug report was made. Our goal, however, is not to simulate the conditions under which bugs are reported, but to obtain accurate severity information. Furthermore, in the total of 1,000 assessments produced by the students, the students used the bug summary, bug description, and comments in 68%, 62%, and 39% of the cases, respectively. For only 15% of the bug reports did both students report that the comment information was helpful to make a decision, and for only 2% of the bug reports did the students only use the comments to make a decision. Therefore, for most bug reports, the students were able to make a decision based on the information available to the original reporter.

TABLE V
SIMILARITY OF STUDENTS' RESPONSE

System	Similar	Different	Proportion of Agreement
JDT	52	73	42%
CDT	62	63	50%
PDE	53	72	42%
Platform	43	81	35%
Total	210	289	42%

4) Feedback: We designed a Google form to receive students' feedback.[6] Our form contains: 1) the bug id, 2) the actual severity of the bug, 3) the specific reason for the decision (free text), 4) the parts (summary, description, and comments) of the bug report that helped make a decision, and 5) the assessor's name. All of the fields were required to submit a response. Students could provide "Not Sure" for the actual severity if they really were not sure about it.

B. Results

We got 1000 responses from the students, comprising two responses for each bug report. There were only 16 responses where at least one student was undecided and thus chose "Not Sure". For only one bug report, from the Platform project, were both responses "Not Sure" due to insufficient information. We investigated all these 16 bug reports and were able to assign a severity level in 15 cases. However, we were not able to assign any severity level to the bug report where both responses were not "Not Sure". We eliminated this report, leaving the responses for 499 bug reports for analysis.

Among the 500 bug reports, there were only 164 reports, *i.e.*, 33%, for which both students gave the same severity level. To refine the results, we focused on the difference between Normal and the other severity levels. Like other work [13], [18], we merged the Blocker, Critical, and Major categories into a higher-level category, *Severe*, and the Minor and Trivial categories into a higher-level category, *Non-Severe* (*NS*). We refer to two responses that are in the same higher-level category as *similar*. After merging the categories, as shown in Table V, we obtain 210 similar responses, amounting to 42% of the reports, leaving 289 where the students disagree. For Platform, the proportion of similar responses is only 35%. The highest rate of agreement (50%) is found for CDT. These results confirm that severity is highly subjective.

Given the high rate of dissimilarity among the severity levels provided by the students, we cannot use the results directly to obtain the proportion of Normal bug reports that are actually normal. Instead, we consider the results from three perspectives: *best case*, *worst case*, and *optimal*. For the best case, *i.e.*, the most optimistic view of the state of the software, we take the lowest level of severity between the two students' responses. For example, if one student categorized a bug as Major and another categorized it as Normal, we would consider the actual severity to be Normal. On the other hand, for the worst case, *i.e.*, the most pessimistic view of the state of the software, we take the highest level of severity between the two responses. Finally, for optimal case, we investigated all the 289 bug reports where students' responses differed and tried to come to a consensus. We read all the bug reports (summary, description, and comments) and students' responses (assigned severity level and specific reason for assigning that level). Then, we made a decision by either taking the one of the students' responses that we agreed with, which was possible in most cases, or assigned a different severity level.

[6]http://goo.gl/XP03JZ

TABLE VI
PROPORTION OF NORMAL BUGS AT EACH SEVERITY LEVEL FROM THE BEST AND WORST CASE PERSPECTIVES

System	Best Case				Worst Case			
	Sev.	Norm.	NS	Enh.	Sev.	Norm.	NS	Enh.
JDT	14	48	48	15	50	40	20	15
	11%	38%	38%	12%	40%	32%	16%	12%
CDT	14	66	21	24	45	52	4	24
	11%	53%	17%	19%	36%	42%	3%	19%
PDE	5	46	45	29	40	31	25	29
	4%	37%	36%	23%	32%	25%	20%	23%
Platform	4	38	54	28	36	36	24	28
	3%	30%	43%	22%	29%	29%	19%	22%
Total	37	198	168	96	171	159	73	96
	7%	40%	34%	19%	34%	32%	15%	19%

Tables VI and VII present the proportion of Normal reports that are classified by the students into the different high-level categories for the best, worst, and optimal cases. In all cases, the Enhancement columns are the same; since enhancements are not bugs, in both tables we use the optimal-case results. From the results, we can see that the proportion of Severe, Normal, and Non-Severe bugs could vary between 7%-34%, 32%-40%, and 15%-34% respectively, depending on how the results are calculated. More specifically, from Table VII, representing the optimal classification, we see that the actual proportion of Normal bugs among those originally labelled Normal is only 35%, and that 19% of the reports originally labelled as Normal are not reports of bugs at all. Furthermore, among the bugs originally labelled Normal, 24% are Severe and 22% are Non-Severe. For JDT, the proportion of Severe bugs is even higher, 33%. Among the 109 reports that are Non-Severe, 84 are Minor and only 25 are Trivial. Therefore, our overall results suggest that the dataset of Normal bugs has serious noise, validating Hypothesis 1: *Normal bugs may not represent the actual severity level*.

VI. SOURCES OF MISCLASSIFICATION

In the previous section, we showed that 65% of Normal bugs are not actually normal according to the definition of Eclipse severity levels. There may be numerous reasons for these misclassifications, from "leaving the severity field at its default value" to "too subjective to decide". In this section, we answer RQ3: *What are the main sources of misclassifications?*

A. Methodology

To understand the reasons for misclassifications, first we investigate the main severity levels that confused assessors.

TABLE VII
PROPORTION OF NORMAL BUGS AT EACH SEVERITY LEVEL FROM THE OPTIMAL CASE PERSPECTIVE

System	Same Response				Different Response				Total			
	Sev.	Norm.	NS	Enh.	Sev.	Norm.	NS	Enh.	Sev.	Norm.	NS	Enh.
JDT	13	19	20	0	28	21	9	15	41	40	29	15
	10%	15%	16%	0%	22%	17%	7%	12%	33%	32%	23%	12%
CDT	12	32	3	15	22	27	6	9	34	58	9	24
	10%	26%	2%	12%	18%	22%	5%	7%	27%	46%	7%	19%
PDE	4	16	25	8	12	30	9	21	16	46	34	29
	3%	13%	20%	6%	10%	24%	7%	17%	13%	37%	27%	23%
Platform	3	13	20	7	26	17	17	21	29	30	37	28
	2%	10%	16%	6%	21%	14%	14%	17%	23%	24%	30%	22%
Total	32	80	68	30	88	95	41	66	120	174	109	96
	6%	16%	14%	6%	18%	19%	8%	13%	24%	35%	22%	19%

TABLE VIII
DIFFERENT RESPONSE MATRIX

Blocker	0						
Critical	2	2					
Major	1	0	16				
Normal	6	3	43	67			
Minor	1	2	4	19	54		
Trivial	2	0	1	2	27	29	
Enhancement	4	1	1	6	19	11	13
	Not Sure	Blocker	Critical	Major	Normal	Minor	Trivial

Then, we read all the bug reports where students' responses differed by high-level category (Severe, Normal, or Non-Severe). In almost all of the cases, it was possible to determine why the student chose a particular severity level by reading the reasons the student provided.

To categorize the common reasons for different responses, we analyzed all the bug reports, following an open-ended taxonomy. For a given bug report, first we identified the high-level reason for the difference, and then we checked whether the reason fits into any of the existing categories. If it did not, we created a new category. We provide concrete examples for each category below, to better understand the categorization procedure. We selected examples that: i) cover the range of subject systems, and ii) have a summary or description in the bug report that is concise enough to present in the paper.

B. Results

Table VIII shows the number of bug reports for each pair of dissimilar responses. For example, the value in the rightmost cell indicates that for 13 bug reports, one student's response is Trivial but the other's response is Enhancement. From the results we see that students were mostly confused between the Normal and Critical, Normal and Major, and Minor and Normal severity levels. Interestingly, Normal bugs are present in each confusion pair. Therefore, it is evident that even after careful assessment, users can be confused between Normal and other categories. The following summarizes our taxonomy of common reasons for the different responses.

1) Focusing on different aspects of a bug report: A bug report may describe several aspects of a bug. Different persons can focus on different aspects, causing them to map the bug to different severity levels. Consider the following example:

Platform # 210946, Description: *A caught Throwable is not written to the Eclipse log. It is just written on the console.*

One student thought that this bug is Severe since it involves losing data from the Eclipse log. Their rationale was that since information is normally written to both the console and the log, the user may close the console and rely only on the log file. Such a user would not even realize that some data were missing. On the other hand, the other student assigned a severity level Minor thinking that there is an easy workaround, since the Throwable is at least written on the console.

2) Same aspect but different perception: In some bug reports both students focused on the same aspect of the bug but their perception of it was different. For example:

CDT # 332915, Summary: *[tracepoint] Refreshing the Trace Control view blocks the UI thread.*

Bug Description: *We've noticed that when heavily using the tracepoint interface, deadlocks can happen due to the UI thread being blocked. Once [*sic*] case is that the refresh operation of the Trace Control view is done within a Query, which locks the UI thread.*

One student responded, "One feature enabled ends up impeding another feature - even though both features work in isolation." Thus, she chose Severe. The second student responded, "GUI and refresh are on the same thread". Thus, it is a regular issue and the student assigned a Normal severity.

3) Different acceptance or tolerance level: Sometimes users may have the same perception about the problem but a different level of tolerance to deal with it. For example:

Platform # 172321, Summary*[Commands] [GTK] Handler activation in editor when a dialog is closed is delayed*

One student thought that this is a "loss of functionality (delay time of feature) only on linux platform" and thus tagged it as Normal. The other student responded, "delay issue for activating handler is a major issue to me". So he chose Major.

4) Impact: Some bugs can seem to fall into the category Minor if the definitions of severity levels are strictly followed. However, the impact of the bug can be annoying enough to make the bug Normal or even Severe. For example:

JDT # 231887, Summary: *[actions] cannot refresh working sets through Package Explorer*

Bug Description: *Steps To Reproduce: 1. Import some Java projects and put them into some working sets. Change the Top Level Elements in the Package Explorer to Working Sets 2. Externally modify some of the files from different working sets 3. Select the working sets in the Package Explorer, right-click, and choose Refresh. Nothing happens. (Verify by opening files that have been modified - instead of opening the file, you get the "This file is out of sync" editor) 4. If you expand all the working sets and refresh the individual projects, it works.*

From the bug summary and description we can see that the user has provided a workaround. However, every time the user changes something, he must refresh each project related to the

working set, which is annoying. Thus, one student marked the bug as Severe saying that it hinders the workflow. However, the other student responded, "Easy workaround. Not so much important bug." We also noticed these dissimilar responses when keyboard shortcuts do not work properly (e.g., Platform # 262593). Again, there is in principle an easy workaround, using menu commands, but some users may be so used to keyboard-shortcuts that they do not feel comfortable with the menu, causing them to view the bug as Severe.

5) Different cost of the same bug: development perspective vs. release perspective: This is the most frequent category in our sample, especially for those dissimilar responses where students are confused between Critical and another category. In our study, in most cases, when a report indicates a program crash, *e.g.*, due to a null pointer exception, the students marked it Critical, which is appropriate according to the definition of the severity levels. Examples include:

JDT # 325523, Summary: *NPE when deleting resource*

PDE # 275921, Summary: *NPE with update classpath*

However, in some cases, the students analyzed the bug from a developer's perspective and marked it as Normal. For example, they said that this was an easy fix or occurred in infrequent cases. Indeed, when an exception is thrown during development and testing, this can be considered as normal since this is a common mistake that developers can fix quickly. However, if a stable version crashes for a given task and the user must wait for the next update to get it resolved, the effect is a lot costlier than the development scenario.

6) Bug vs. Enhancement: Whether a given issue is a bug or an enhancement is subjective, and is thus itself a reason for misclassification. Furthermore, even if a reporter correctly classifies an enhancement, s/he may end up representing it incorrectly in the Bugzilla database due to the tricky Bugzilla configuration used by Eclipse. Indeed, in the configuration of Bugzilla used by the Eclipse projects there is no separate field for distinguishing bugs from enhancements. Instead, Enhancement is just one possible bug severity level. Therefore, if a change request is an enhancement, the reporter should set the severity label to Enhancement regardless of the request's importance. In practice, however, we found many cases where reporters marked enhancements as Major, Normal, or Minor depending on their perceived importance, thus implicitly misclassifying the change request as a Bug, not as an Enhancement. We discuss some examples:

Platform # 185067, Summary: *[KeyBindings] New Keys pref page: cannot sort 'User' column*

PDE # 330943, Description: *[plug-in registry] View initialization takes too much time*

In the first example, the issue reporter is asking for sort functionality to be added to the "new keys" preference page. Since what is asked for is a new feature, it is an enhancement, not a bug, even if the reporter finds it inconvenient or inconsistent that the feature is not currently available. Likewise, in the second example, there is no error in "View initialization." Instead, the reporter is requesting that the performance be improved. However, in the students' assessment, one response for each bug was Minor, since the students thought that these issues would not affect users much.

C. Discussion for Hypothesis 2

We now investigate the existing Hypothesis 2, whether "leaving the severity field at its default value" is the main reason for severity misclassification. For this, we try to infer possible motivations from the experiences of our student assessors and from the bug report characteristics themselves.

Experiences of the student assessors: All the 500 bug reports in our manual investigation are marked as Normal in the Eclipse Bugzilla. However, for 65% of these reports, the student assessors had a different opinion of the severity from the original labeller. We try to understand why differences of opinion can occur by studying the differences of opinion that occurred within our manual study. Specifically, for 58% of the reports, each of the students evaluating the report assigned it a severity in a different higher-level category (e.g., severe, normal or NS). Careful investigation into the students' responses reveal that most of these discrepancies were due to the subjective nature of the assessment. There are indeed many factors to consider, such as minor or major loss of functionality, frequent or infrequent use cases, the convenience of any workaround, etc. Each of these factors is itself subjective, and our analysis in Section VI-B shows that different choices by the students often resulted from their putting different weights on these different subjective criteria. As the students were told to pay careful attention to the evidence available in choosing a bug severity, but still often came up with different labels, it is possible that the original reporters were also paying attention to the choice of severity, but made choices that were different than those of the students.

Furthermore, we observe that our optimal strategy classifies only 3 of the selected 500 Normal reports as Blocker and only 25 of these reports as Trivial, implying that most of the differences of opinion between the original reporter and the students was among the severity levels closer to Normal. These differences are again likely to be more subjective.

Report characteristics: We also performed a simple automatic analysis on all the fixed bug reports to get an idea about the proportion of bugs that should be Trivial but are categorized as Normal. We found that many of the reports classified as Trivial by our optimal strategy contain the keywords *typo*, *spell*, and *documentation*, either in the report or in the students' comment, and that these words appear rarely in the other reports in our sample. Text search of the summary and description of all the fixed Normal bugs in the complete dataset showed that only 1% of such bug reports contain these words. Again, we find little overlap between levels that are far apart, suggesting that misclassifications are between similar categories for which the differences are more subjective.

VII. Misclassification or Exclusion of Normal Bugs: Do They Matter?

In this section, we investigate RQ4: *Can misclassification or exclusion of Normal bugs affect previous study results?*

A. Methodology

As we already noted, many previous studies use the bug severity field as a feature in various techniques such as bug-fix time prediction, modeling bug report quality, severity prediction, etc. A number of previous studies have ignored Normal bug reports, on the assumption that Normal does not correctly reflect the severity level. In this study, we have confirmed this assumption. However, a tool that has not been trained on Normal bugs may subsequently give meaningless results on Normal input. Such a tool is thus unusable in a real-world setting unless accurate severity information is already available. Therefore, we investigate two phenomena: 1) whether there can be any effect of misclassification on previous study results, and 2) whether there can be any impact on the results if the Normal bugs are eliminated from the study.

To investigate the impact of these phenomena, we chose bug severity prediction as a representative application. Generally a bug severity prediction algorithm takes a set of bug reports known to be from various categories (e.g., Severe and Non-Severe) as training data and uses the properties inferred from this training data to predict the severity of bug reports in a test set. Lamkanfi et al. [13] showed that taking into account bug summaries is sufficient to get accurate results. Since our objective is to investigate the effect of misclassification or exclusion of Normal bugs on accuracy, not to propose new techniques for bug severity prediction, we have just implemented a simple approach. Our bug severity prediction system takes a set of bug summaries labelled with severity as a training set and predicts the severity of an input report represented by its summary. Our predictions are coarse-grained: Severe, Normal, and Non-Severe. We use Mallet's implementation of Naive Bayes out-of-the-box as our underlying classifier. Then we measure accuracy in terms of the proportion of correctly classified items. Specifically, the accuracy of our classifier is $m/n * 100\%$ if it classifies m instances correctly out of n instances. For a comprehensive description of bug severity prediction, please see [13], [18], [19].

Training and Test Set Creation: We distinguish between a *clean* dataset, in which we have good confidence that the severity labels are accurate, and a *noisy* dataset, in which it is not known whether the severity labels are accurate. Either kind of dataset furthermore may or may not contain Normal bugs. This leads to four training sets, TR_{clean}, TR_{noisy}, $TR_{clean}-Normal$, and $TR_{noisy}-Normal$, having various permutations of these properties. We train our severity prediction algorithm on each of these training sets, resulting in four instances of the tool. To assess the impact of noisy data and of excluding Normal bugs, we then test each of these tool instances on a clean test set, TE, and compare the accuracy of the resulting predictions with the known labels.

A challenge in our experimental methodology is obtaining sufficient clean data. Indeed, our training and test sets must respect a number of constraints. First, the training set and the test set should be disjoint. Furthermore, previous research has shown that to avoid bias due to the over-prevalence of data in one class, all of the training sets (clean or otherwise) and the test set should have the same number of bugs at each severity level [23]. Specifically, if one class has few instances in the training set, any learning algorithm would know less about that class. Likewise, if a one class is very dominant in a test set, the evaluation result would be biased toward that class. For example, in a two-class (C_A and C_B) test set , if one class (C_A) represents 90% of instances and if a naive classifier says all the instances are C_A, its accuracy would be still 90%. Finally, Lamkanfi et al. [13] have found that a training set of 500 bug reports in each category gives stable results.

In addressing our previous research questions, we have manually investigated only 500 bug reports, and among them 120, 174, and 109 are classified as Severe, Normal, and Non-Severe, respectively. Using this dataset, and respecting the constraint that there should be the same number of bugs at each severity level, we can obtain a data set of at most only slightly over 300 elements. Concretely, we take the 100 most recent reports in each severity level, resulting in a dataset of 300 elements. As this does not satisfy the requirement of 500 reports in each category, we cannot use this as a training set. Thus, we use it as the test set, TE.

For the training sets, we need a low-cost way to obtain more data with reliable severity labels. For this, we focus on the bug reports in which the severity information has been changed at least once in any way. This may not result in a completely clean set, but it should be acceptable, since developers, who we assume to be experts in the software, have reviewed those bug reports and adjusted their severity level. Starting from the year 2006, we take the first 500 bug reports having this property in each severity category. For example, the training set of Normal bugs consists of those bug reports that ended up being Normal after one or more severity changes. The resulting set of 1500 reports makes up TR_{clean}. To create TR_{noisy}, we follow the same procedure, without the requirement of a change in the severity information. Then, from TR_{clean} and TR_{noisy}, we obtain $TR_{clean}-Normal$ and $TR_{noisy}-Normal$, respectively, by removing the Normal reports. As we have taken the training data TR from the start of the time period and the test data TE from the end of the time period, they are likely disjoint. We have furthermore verified this in practice.

Finally, we note that all of the datasets contain bug reports from all four subject systems.

B. Results

We present our results in terms of the confusion matrix and accuracy. Precision, recall, and F-measure can all be calculated from the confusion matrix. Table IX shows the impact of misclassification on the accuracy of the severity predictor trained on the full clean dataset TR_{clean}. In the results, each row is the number of predicted results for a given category. For example, the first row represents the 100 actual Severe bugs in TE. Of these, 57 are predicted to be Severe, 26 are predicted to be Normal, and 17 are predicted to be Non-Severe. Based on the actual severity level of bug reports in TE, the

TABLE IX
ACCURACY FOR SEVERITY PREDICTION CLASSIFIER TRAINED FROM TR_{clean}

Actual \ Predicted	Severe	Normal	Non-Severe	
Severe	57	26	17	Accuracy: 49%
Normal	35	38	27	
Non-Severe	22	27	51	

Accuracy: 29% if we consider all the bug reports in TE to be Normal, as indicated in the bug repository.

TABLE X
ACCURACY FOR SEVERITY PREDICTION CLASSIFIER TRAINED FROM $TR_{clean} - Normal$

Actual \ Predicted	Severe	Normal	Non-Severe	
Severe	75	.	25	Accuracy: 47%
Normal	45	.	55	
Non-Severe	33	.	67	

Accuracy: 71% if we exclude Normal bugs from TE

accuracy of our classifier is 49%. However, if we consider all the bug reports in TE to be Normal, as they are classified in bug repository, then the accuracy is only 29%.

We next perform the same experiment, but where all Normal bugs have been removed from the clean training set, producing $TR_{clean} - Normal$. Since the training dataset contains no Normal data, no bugs in the test set TE will be classified as Normal (see Table X) and the accuracy of the resulting classifier is only 47%. However, if we also exclude Normal bugs from TE, as done in previous studies, the accuracy increases to 71%. Therefore, the accuracy reported in the existing literature is likely overestimated if the tool is intended to be used on unlabeled data that may contain Normal bugs.

Next, we repeat both experiments for the case of noisy training data. First, we consider TR_{noisy}, containing all categories of bugs. Table XI shows that we obtain an accuracy of 41% for TE with this training data. This value is 8% less than that obtained when we trained our classifier with TR_{clean}. Therefore, misclassification in the training data can reduce the accuracy of the severity prediction considerably.

Next, we consider $TR_{noisy} - Normal$, in which the reports labelled Normal have been removed. As compared to the use of the clean training set $TR_{clean} - Normal$, the accuracy only slightly declines, from 47% to 45% (Table XII), which is less than the decline between the results obtained using TR_{clean} and TR_{noisy}. Furthermore, as compared to TR_{noisy}, the accuracy actually improves, from 41% to 45%. This result

TABLE XI
ACCURACY FOR SEVERITY PREDICTION CLASSIFIER TRAINED FROM TR_{noisy}

Actual \ Predicted	Severe	Normal	Non-Severe	
Severe	34	49	17	Accuracy: 41%
Normal	36	40	24	
Non-Severe	22	29	49	

TABLE XII
ACCURACY FOR SEVERITY PREDICTION CLASSIFIER TRAINED FROM $TR_{noisy} - Normal$

Actual \ Predicted	Severe	Normal	Non-Severe	
Severe	63	.	37	Accuracy: 45%
Normal	52	.	48	
Non-Severe	29	.	71	

Accuracy: 67% if we exclude Normal bugs from TE

indicates that the most noise is in Normal bugs. Finally, the reported accuracy again increases greatly if our test set does not include Normal bugs, reaching 67%. But knowing in advance whether a bug is Normal is not a reasonable assumption for the input of a bug severity prediction tool.

Therefore, our overall results suggest that both misclassification and exclusion of Normal bugs may significantly affect any results based on bug severity.

VIII. THREATS TO VALIDITY

Construct Validity: The set of bug reports is the only artifact used in our study; bug reports are generally well understood. We have also used well known metrics in our data analysis such as proportion and classification accuracy, which are straightforward to compute. We use a publicly available dataset, which enables the replication of this study. Therefore, we argue for a strong construct validity.

Internal validity: A bug report may be filed by either an Eclipse developer or a real user. The students who participated in our study are not involved in Eclipse development. However, they frequently use Eclipse for their own software development, e.g., research and class projects. Furthermore, 9 out of the 10 students had previous experience in industrial software development. Therefore, we believe that the students have the background necessary to assess bug severity.

All the bug reports used in our study are extracted from Bugzilla. There are many other bug tracking systems such as Jira, Mantis, etc. Since the severity levels may vary across projects and bug tracking systems, we may get different results for the bug reports in other bug tracking system.

To assess the severity of bugs, the students manually analyzed bug reports. To delineate the common reasons for misclassification, we also manually analyzed bug reports and students' responses. There might have been some unintentional misinterpretations during the manual verification due to the lack of domain knowledge. However, we held extensive discussions to minimize this threat.

To construct a clean training set, we selected bug reports whose severity had been changed at least once. Although we did not manually check that these bug reports have the actual level of severity, we believe the dataset should be fairly accurate since either bug triagers or developers examined those bug reports and adjusted the severity level accordingly.

External Validity: Eclipse may not be representative of all software. Still, it has been used in a number of studies, and so the conclusions drawn from it are at least applicable to

those studies. Furthermore, we find similar results across the different Eclipse sub-projects.

Since manual investigation of bug reports is expensive, we investigated only 500 bug reports. We plan to conduct a more large-scale manual investigation in the future. Still, it has been shown that a sample of size 500 has suffient power to detect all but the smallest effects [24]. In our manual investigation, each report was assessed by two students. More assessments may further increase the confidence in the results. However, we separately investigated all the dissimilar responses. Therefore, this threat should have little impact on our results.

IX. Related Work

Our work is related to the study of bug reports, and more specifically bug severity. We review some recent work that has relied on this information. Our work specifically investigates a source of bias or noise in bug reports. We also review some work that has investigated other such issues.

Bug reports are one of the main artifacts in software maintenance research. They have been used to understand various phenomena about software bugs and to design tools to help developers in the bug fixing process. Bettenburg et al. [8] investigated what kind of information developers think is the most helpful in a bug report. They also investigated the extents and reasons of duplicated bug reports [7]. Bortis et al. [9] proposed to tag bug reports automatically to help with bug triaging. Tian et al. [14] proposed a machine-learning based approach for assigning a priority to each bug report. Anvik et al. [10] and Shokripour et al. [25] proposed approaches for automatic assignment of bug reports to developers. Saha et al. [15] and Zhou et al. [16] proposed approaches for automatic bug localization. Huo et al. [26] investigated the role of expert and non-expert knowledge in bug reports and its impact on the results of bug localization tools.

Bug Severity is one of the key features of a bug report, to understand the bug's importance. Researchers have used this attribute in numerous software engineering studies. Menzies and Marcus [27] and Lamkanfi et al. [13], [18] proposed a text mining and machine learning based approach to predict bug severity. Tian et al. [19] also predicted bug severity, based on information retrieval. Bhattacharya et al. [28] proposed a graph-based approach to estimate bug severity. Hooimeijer and Weimer [29] used bug severity to investigate and model bug report quality. They concluded that self-reported severity is an important factor in the model's performance. Giger et al. [11] and Zhang et al. [12] used bug severity (with several other bug report features) to predict bug-fix time. Saha et al. [17] used severity to understand the importance of long lived bugs.

Bias or noise in bug-relevant datasets is a well-known problem in software engineering research. Bird et al. [30] investigated the potential biases in defect datasets in terms of bug features and commit features. They evaluated a popular defect prediction algorithm and showed that bug feature bias (e.g., unequal proportion of bug reports in terms of bug severity and developers' reputation) affects the performance of the algorithm. Later Nguyen et al. [31] confirmed that the bias in the bug-fix dataset exists not only in open-source projects but also in the datasets of commercial projects. Kim et al. [32] proposed an algorithm to detect such noisy instances in bug datasets so that they can be eliminated. However, these studies did not investigate noise in bug severity. Rahman et al. [33] showed that consideration of the sample size of a dataset is as important as bias in the dataset. Antoniol et al. [20] showed that not all the bug reports in bug tracking systems are actually bugs. Later, based on a comprehensive manual investigation on 7,000 issue reports, Herzig et al. [2] reported that one-third of the bugs in the issue tracking systems are not actually bugs and this misclassification affects bug prediction algorithms. Kochhar et al. [34] investigated the potential biases in the dataset of mappings between bug reports and corresponding fixed files, and described their impact on bug localization.

X. Conclusion

In this paper, we have studied the mislabeling of Normal bugs, and the impact that it can have on tools that rely on bug severity. Based on the studied software projects, we confirm that the bugs labeled Normal are often not normal according to the bug repository criteria. Furthermore, we find that the inclusion or exclusion of these reports, as well as their consideration as Normal bugs or according to their actual severity, can have a major impact on the accuracy of tools that rely on bug severity values. This raises a real dilemma for the software engineering researcher. Normal reports are very prevalent, around 80% of the reports in our study, but cannot be relied on and are damaging to tool evaluations.

A partial solution is to create a clean dataset. Indeed, our results show that a bug severity prediction tool gives better results when trained on clean data than when trained on noisy data. We have proposed two approaches to creating a clean dataset: manual inspection and selecting only reports where the severity has been changed after the original submission. The former, however, is time-consuming, and the latter is more approximate. The wide use of bug reports by the software engineering community thus suggests that the community may want to invest resources into creating larger clean datasets.

We have also observed that misclassification of bugs and enhancements is a severe problem, which also may affect many studies. It appears that distinguishing enhancements from bugs through the severity field is not effective, because it does not allow the user to express the urgency of the enhancement request. Users could be less tempted to create noisy data if bug tracking systems such as Bugzilla would provide a dedicated field to separate bugs from enhancements. Our future work includes manually validating more Normal bug reports to create a large-scale clean dataset, and improving the state-of-the-art of severity prediction algorithms.

Data. Our data and results are publicly available at: http://www.riponsaha.com/data/severity-assessments.csv.

Acknowledgments. We would like to thank the students who participated in the bug severity labeling study. This work was funded in part by the National Science Foundation (NSF Grant Nos. CCF-1117902 and CCF-0845628).

REFERENCES

[1] P. J. Guo, T. Zimmermann, N. Nagappan, and B. Murphy, "Not my bug! and other reasons for software bug report reassignments," in *CSCW*. ACM, 2011, pp. 395–404.

[2] K. Herzig, S. Just, and A. Zeller, "It's not a bug, it's a feature: How misclassification impacts bug prediction," in *ICSE*. IEEE Press, 2013, pp. 392–401.

[3] T.-H. Chen, M. Nagappan, E. Shihab, and A. E. Hassan, "An empirical study of dormant bugs," in *MSR*. ACM, 2014, pp. 82–91.

[4] E. Shihab, A. Ihara, Y. Kamei, W. Ibrahim, M. Ohira, B. Adams, A. Hassan, and K.-i. Matsumoto, "Studying re-opened bugs in open source software," *Empirical Software Engineering*, vol. 18, no. 5, pp. 1005–1042, 2013.

[5] G. Canfora, M. Ceccarelli, L. Cerulo, and M. Di Penta, "How long does a bug survive? an empirical study," in *ICSM*. IEEE Computer Society, 2011, pp. 191–200.

[6] F. Zhang, F. Khomh, Y. Zou, and A. E. Hassan, "An empirical study on factors impacting bug fixing time," in *ICSM*. IEEE Computer Society, 2012, pp. 225–234.

[7] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim, "Duplicate bug reports considered harmful...really?" in *ICSM*, 2008, pp. 337–345.

[8] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, "What makes a good bug report?" in *ESEC/FSE*. ACM, 2008, pp. 308–318.

[9] G. Bortis and A. van der Hoek, "Porchlight: A tag-based approach to bug triaging," in *ICSE*. IEEE, 2013, pp. 342–351.

[10] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in *ICSE*. ACM, 2006, pp. 361–370.

[11] E. Giger, M. Pinzger, and H. Gall, "Predicting the fix time of bugs," in *RSSE*. ACM, 2010, pp. 52–56.

[12] H. Zhang, L. Gong, and S. Versteeg, "Predicting bug-fixing time: an empirical study of commercial software projects," in *ICSE*. IEEE Press, 2013, pp. 1042–1051.

[13] A. Lamkanfi, S. Demeyer, E. Giger, and B. Goethals, "Predicting the severity of a reported bug," in *MSR*, 2010, pp. 1–10.

[14] Y. Tian, D. Lo, and C. Sun, "Drone: Predicting priority of reported bugs by multi-factor analysis," in *ICSM*. IEEE, 2013, pp. 200–209.

[15] R. K. Saha, M. Lease, S. Khurshid, and D. E. Perry, "Improving bug localization using structured information retrieval," in *ASE*, 2013, pp. 345–355.

[16] J. Zhou, H. Zhang, and D. Lo, "Where should the bugs be fixed? - more accurate information retrieval-based bug localization based on bug reports," in *ICSE*, 2012, pp. 14–24.

[17] R. K. Saha, S. Khurshid, and D. E. Perry, "An empirical study of long lived bugs," in *CSMR/WCRE*. IEEE, 2014, pp. 144–153.

[18] A. Lamkanfi, S. Demeyer, Q. D. Soetens, and T. Verdonck, "Comparing mining algorithms for predicting the severity of a reported bug," in *CSMR*. IEEE, 2011, pp. 249–258.

[19] Y. Tian, D. Lo, and C. Sun, "Information retrieval based nearest neighbor classification for fine-grained bug severity prediction," in *WCRE*. IEEE, 2012, pp. 215–224.

[20] G. Antoniol, K. Ayari, M. Di Penta, F. Khomh, and Y.-G. Guéhéneuc, "Is it a bug or an enhancement?: a text-based approach to classify change requests," in *CASCON*. ACM, 2008, p. 23.

[21] A. Lamkanfi, J. Pérez, and S. Demeyer, "The Eclipse and Mozilla defect tracking dataset: a genuine dataset for mining bug information," in *MSR*. IEEE Press, 2013, pp. 203–206.

[22] A. Murgia, J. Pérez, S. Demeyer, C. De Roover, C. Scholliers, and V. Jonckers, "Predicting bug-fixing time using bug change history," *BENEVOL 2013*, p. 20, 2013.

[23] N. Japkowicz, "Learning from imbalanced data sets: A comparison of various strategies," in *Proceeding of the AAAi Workshop on Learning from Imbalanced Data Sets*. AAAi, 2000, pp. 10–15.

[24] R. Rosenthal and R. L. Rosnow, *Essentials of behavioral research: Methods and data analysis*. McGraw-Hill New York, 1991, vol. 2.

[25] R. Shokripour, J. Anvik, Z. M. Kasirun, and S. Zamani, "Why so complicated? simple term filtering and weighting for location-based bug report assignment recommendation," in *MSR*, 2013, pp. 2–11.

[26] D. Huo, T. Ding, C. McMillan, and M. Gethers, "An empirical study of the effects of expert knowledge on bug reports," in *ICSME*. IEEE, 2014, pp. 1–10.

[27] T. Menzies and A. Marcus, "Automated severity assessment of software defect reports," in *ICSM*. IEEE, 2008, pp. 346–355.

[28] P. Bhattacharya, M. Iliofotou, I. Neamtiu, and M. Faloutsos, "Graph-based analysis and prediction for software evolution," in *ICSE*. IEEE Press, 2012, pp. 419–429.

[29] P. Hooimeijer and W. Weimer, "Modeling bug report quality," in *ASE*. ACM, 2007, pp. 34–43.

[30] C. Bird, A. Bachmann, E. Aune, J. Duffy, A. Bernstein, V. Filkov, and P. Devanbu, "Fair and balanced?: bias in bug-fix datasets," in *ESEC/FSE*. ACM, 2009, pp. 121–130.

[31] T. H. Nguyen, B. Adams, and A. E. Hassan, "A case study of bias in bug-fix datasets," in *WCRE*. IEEE, 2010, pp. 259–268.

[32] S. Kim, H. Zhang, R. Wu, and L. Gong, "Dealing with noise in defect prediction," in *ICSE*. IEEE, 2011, pp. 481–490.

[33] F. Rahman, D. Posnett, I. Herraiz, and P. Devanbu, "Sample size vs. bias in defect prediction," in *FSE*. ACM, 2013, pp. 147–157.

[34] P. S. Kochhar, Y. Tian, and D. Lo, "Potential biases in bug localization: Do they matter?" in *ASE*. ACM, 2014, pp. 803–814.