Experience Report: How Do Bug Characteristics Differ Across Severity Classes: A Multi-platform Study

Bo Zhou Iulian Neamtiu Rajiv Gupta
Department of Computer Science and Engineering
University of California, Riverside
Email: {bzhou003,neamtiu,gupta}@cs.ucr.edu

***Abstract*—Bugs of different severities have so far been put into the same category, but their characteristics differ significantly. Moreover, the nature of issues with the same severity, e.g., high, differs markedly between desktops and smartphones. To understand these differences, we perform an empirical study on 72 Android and desktop projects. We first define three bug severity classes: high, medium, and low. Next, we study how severity changes and quantify differences between classes in terms of bug-fixing attributes. Then, we focus on topic differences: using LDA, we extract topics associated with each severity class, and study how these topics differ across classes, platforms, and over time. Our findings include: severity and priority affect bug fixing time; medium-severity contributors are more experienced than high-severity contributors; and there have been more concurrency and cloud-related high-severity bugs on desktop since 2009, while on Android concurrency bugs are prevalent.**

***Keywords*—Bug severity, cross-platform analysis, bug report, topic model.**

I. Introduction

Bug tracking systems such as Bugzilla, Trac, or Google Code are widely popular in large-scale collaborative software development. Such systems are instrumental as they provide a structured platform for interaction between bug reporters and bug fixers, and permit reporting, tracking the progress, collaborating on, and ultimately fixing or addressing the reported bugs (issues).

A key attribute of each bug report on these systems is *bug severity*—an indicator of the bug's potential impact on users. While different bug tracking systems use different severity scales, we found that bugs can be assigned into one of three severity classes: *high severity bugs* represent issues that are genuine show-stoppers, e.g., crashes, data corruption, privacy leaks; *medium severity bugs* refer to issues such as application logic issues or occasional crashes; and *low severity bugs* usually refer to nuisances or requests for improvement (Section II-C defines classes and provides details).

Severity is important for effort planning and resource allocation during the bug-fixing process; we illustrate this with several examples. First, while our intuition says that bugs with different severity levels need to be treated differently, for planning purposes we need know how bug severity influences bug characteristics, e.g., fix time or developer workload. Second, assigning the wrong severity to a bug will lead to resource mis-allocation and wasting time and effort; a project where bugs are routinely assigned the wrong severity level might have a flawed bug triaging process. Third, if high-severity bugs tend to have a common cause, e.g., concurrency, that suggests more time and effort needs to be allocated to preventing those specific issues (in this case concurrency). Hence understanding bug severity can make development and maintenance more efficient.

To this end, in this paper we perform a thorough study of severity in a large corpus of bugs on two platforms, desktop and Android. Most of the prior work on bug severity has focused on severity prediction [1]–[4]; there has been no research on how severity is assigned and how it changes, on how bugs of different severities differ in terms of characteristics (e.g., in fix time, developer activity, and developer profiles), and on the topics associated with different classes of severity. Therefore, to the best of our knowledge, we are the first to investigate differences in bug characteristics based on different severity classes.

Our study is based upon 72 open source projects (34 on desktop and 38 on Android) comprising of 441,162 fixed bug reports. The bug reports cover a 15-year period for desktop and a 6-year period for Android. In particular, we studied the bug-fix process features, bug nature and the reporter/fixer relationship to understand how bugs, as well as bug-fixing processes, differ between bug severity classes. Section II describes our methodology, including how we selected projects, the steps and metrics we used for extracting bug reports, process features, and topics.

The study has two thrusts, centered around nine research questions, RQ1–RQ9. First, a *quantitative* thrust (Section III) where we study how severity assigned to a bug might change; next, we compare the three severity classes in terms of attributes associated with bug reports and the bug-fixing process, how developer profiles differ between high, medium and low severity bugs, etc. Second, a *topic* thrust (Section IV) where we apply LDA (Latent Dirichlet Allocation [5]) to extract topics from bug reports and gain insights into the nature of bugs, how bug categories differ among bug severity classes, and how these categories change over time.

We now present some highlights of our findings:

- Severity changes are more frequent on Android, where some projects have change rates in excess of 20%, than on desktop.
- Bug-fix time is affected by not only severity but also priority. Interestingly, there are marked quantitative difference between the three severity classes on desktop, but

TABLE I: Projects examined, number of fixed bugs, severity classes percentage, severity level change percentage and time span.

Desktop						
Project	Fixed bugs	Severity(%)				Time span
		Hi	Med	Low	Change	
Mozilla Core	101,647	20	73	7	8	2/98-12/13
OpenOffice	48,067	14	73	13	1	10/00-12/13
Gnome Core	42,867	13	71	17	8	10/01-12/13
Eclipse platform	42,401	15	71	14	10	2/99-12/13
Eclipse JDT	22,775	11	71	18	10	10/01-12/13
Firefox	19,312	9	79	12	6	4/98-12/13
SeaMonkey	18,831	21	64	14	13	4/01-12/13
Konqueror	15,990	18	72	10	3	4/00-12/13
Eclipse CDT	10,168	12	74	14	10	1/02-12/13
WordPress	9,995	14	67	19	8	6/04-12/13
KMail	8,324	15	57	27	4	11/02-12/13
Linux Kernel	7,535	18	76	5	3	3/99-12/13
Thunder-bird	5,684	14	69	17	7	4/00-12/13
Amarok	5,400	20	58	22	6	11/03-12/13
Plasma Desktop	5,294	24	62	14	6	7/02-12/13
Mylyn	5,050	8	50	41	11	10/05-12/13
Spring	4,937	63	0	37	NA	8/00-12/13
Tomcat	4,826	21	55	24	9	11/03-12/13
MantisBT	4,141	26	0	74	2	2/01-12/13
Hadoop	4,077	82	0	18	NA	10/05-12/13
VLC	3,892	19	72	9	8	5/05-12/13
Kdevelop	3,572	24	57	19	5	8/99-12/13
Kate	3,326	15	61	24	5	1/00-12/13
Lucene	3,035	65	0	35	NA	4/02-12/13
Kopete	2,957	18	60	22	8	10/01-9/13
Hibernate	2,737	80	0	20	NA	10/00-12/13
Ant	2,612	14	47	39	9	4/03-12/13
Apache Cassandra	2,463	54	0	46	NA	8/04-12/13
digikam	2,400	20	56	25	5	3/02-12/13
Apache httpd	2,334	21	52	27	9	2/03-10/13
Dolphin	2,161	32	51	17	5	6/02-12/13
K3b	1,380	18	70	13	8	4/04-11/13
Apache Maven	1,332	85	0	15	NA	10/01-12/13
Portable OpenSSH	1,061	11	57	31	5	3/09-12/13
Total	*422,583*	*20*	*69*	*11*	*7*	

Android						
Project	Fixed bugs	Severity(%)				Time span
		Hi	Med	Low	Change	
Android Platform	3,497	1	97	2	1	11/07-12/13
Firefox for Android	4,489	12	86	2	4	9/08-12/13
K-9 Mail	1,200	4	94	2	4	6/10-12/13
Chrome for Android	1,601	2	79	19	14	10/08-12/13
OsmAnd Maps	1,018	2	97	1	8	1/12-12/13
AnkiDroid Flashcards	746	41	48	10	50	7/09-12/13
CSipSimple	604	5	92	3	7	4/10-12/13
My Tracks	525	11	87	2	5	5/10-12/13
Cyanogen-Mod	432	1	99	0	1	9/10-1/13
Andro-minion	346	2	92	5	3	9/11-11/13
WordPress for Android	317	78	0	22	0	9/09-9/13
Sipdroid	300	0	100	0	0	4/09-4/13
AnySoft-Keyboard	229	41	59	0	32	5/09-5/12
libphone-number	219	4	95	1	4	10/10-12/13
ZXing	218	6	62	32	13	11/07-12/13
SL4A	204	0	100	0	0	5/09-5/12
WebSMS-Droid	197	44	52	4	46	10/09-12/13
OpenIntents	188	28	61	10	5	12/07-6/12
IMSDroid	183	1	99	0	1	6/10-3/13
Wikimedia Mobile	166	15	37	48	8	1/09-9/12
OSMdroid	166	1	96	3	4	2/09-12/13
WebKit	157	1	98	1	0	11/09-3/13
XBMC Remote	129	37	58	5	33	9/09-11/11
Mapsforge	127	23	73	4	NA	2/09-12/13
libgdx	126	0	100	0	0	5/10-12/13
WiFi Tether	125	2	96	2	3	11/09-7/13
Call Meter NG&3G	116	47	52	1	46	2/10-11/13
GAOSP	114	6	89	5	11	2/09-5/11
Open GPS Tracker	114	30	68	2	12	7/11-9/12
CM7 Atrix	103	0	95	5	5	3/11-5/12
Transdroid	103	23	73	4	22	4/09-10/13
MiniCM	101	0	100	0	4	4/10-5/12
Connectbot	87	3	94	2	6	4/08-6/12
Synodroid	86	29	63	8	20	4/10-1/13
Shuffle	77	9	87	4	9	10/08-7/12
Eyes-Free	69	7	91	1	9	6/09-12/13
Omnidroid	61	22	68	10	20	10/09-8/10
VLC for Android	39	17	81	3	8	5/12-12/13
Total	*18,579*	*10*	*85*	*5*	*8*	

the differences are more muted on Android.

- Fixing medium-severity bugs imposes a greater workload than fixing high-severity bugs.
- Medium-severity bug reporters/owners are more experienced than high-severity bug reporters/owners.
- There have been more concurrency and cloud-related high-severity bugs on desktop since 2009, while on Android concurrency bugs have been and continue to be prevalent.

Since our study reveals that bug-fixing process attributes and developer traits differ across severity levels, our work confirms the importance of prior work that has focused on accurately predicting bug severity [2]–[4].

II. Methodology

We now present an overview of the projects we examined, as well as the methodology we used to extract the bug features and topics.

A. Examined Projects

We chose 72 open source projects for our study, spread across two platforms: 34 projects on desktop and 38 projects on Android. We used several criteria to choose these projects and reduce confounding factors. First, the projects we selected have large user bases, e.g., on desktop we chose[1] Firefox, Eclipse, Apache, KDE, Linux kernel, WordPress, etc.; on Android, we chose Firefox for Android, Chrome for Android, Android platform, K-9 Mail, WordPress for Android, etc. Second, we chose projects that are popular, as indicated by the number of downloads and ratings on app marketplaces. For the Android projects, the mean number of downloads as indicated on Google Play, was 1 million, while the mean number of user ratings was 7,807. Third, we chose projects that have had a relatively long evolution history ("relatively long" because the Android platform emerged in 2007). Fourth, to reduce selection bias, we chose projects from a wide range of categories—browsers, media players, utilities, infrastructure.

Table I shows a summary of the projects we examined. For each platform, we show the project name, the number of fixed bugs, the percentage of bugs in each severity class (High, Medium, and Low), the percentage of bugs that had severity changes, and finally, the time span, i.e., the dates of the first and last bugs we considered.

B. Collecting Bug Reports

We now describe the process used to collect data. All 72 projects offer public access to their bug tracking systems. The projects used various bug trackers: desktop projects tend to use Bugzilla, Trac, or JIRA, while Android projects use mostly Google Code, though some use Bugzilla or Trac. We used Scrapy,[2] an open source web scraping tool, to crawl and extract bug report features from bug reports located in each bug tracking system.

For bug repositories based on Bugzilla, Trac, and JIRA, we only considered bugs with resolution `RESOLVED` or `FIXED`, and status `CLOSED`, as these are confirmed bugs. We did not

[1]Many of the desktop projects we chose have previously been used in empirical studies [6]–[11].

[2]http://scrapy.org

consider bugs with other statuses, e.g., UNCONFIRMED and other resolutions, e.g., WONTFIX, or INVALID. For Google Code-based bug repositories, we selected bug reports with bug type defect and bug status fixed, done, released, or verified.

C. Severity Classes

Since severity levels differ among bug tracking systems, we mapped severity from different trackers to a uniform 10-point scale, as follows: 1=Enhancement, 2=Trivial/Tweak, 5=Minor/Low/Small, 6=Normal/Medium, 8=Major/High, 9=Critical/Crash, 10=Blocker. Then we classified all bug reports into three classes, *High*, with severity level ≥ 8, *Medium*, with severity level=6, and *Low*, with severity level ≤ 5. This classification is based on previous research [1], [2]; we now describe each category.

High severity bugs represent issues that are genuine show-stoppers, e.g., crashes, data corruption, privacy leaks.

Medium severity bugs refer to issues such as application logic issues or occasional crashes.

Low severity bugs usually refer to either nuisances or requests for improvement.

D. Quantitative Analysis

To find quantitative differences in bug-fixing processes we performed an analysis on various features (attributes) of the bug-fixing process, e.g., fix time, comment length. We now provide definitions for these features.

FixTime: the time required to fix the bug, in days, computed from the day the bug was reported to the day the bug was closed. *BugTitle:* the text content of the bug report title. *BugDescription:* the text content of the bug summary/description. *DescriptionLength:* the number of words in the bug summary/description. *TotalComments:* the number of comments in the bug report. *CommentLength:* the number of words in all the comments attached to the bug report. *Priority* describes the importance and order in which a bug should be fixed compared to other bugs; it is set by the maintainers or developers who plan to work on the bug; there are 5 levels of priority, with P1 the highest and P5 the lowest.[3] *BugReporter:* the ID of the contributor who reported the bug. *BugOwner:* the ID of the contributor who eventually fixed the bug. *DevExperience:* the experience of developer X on project Y in year Z, defined as the difference, in days, between the date of the X's last contribution on project Y in year Z and X's first contribution on project Y ever.

E. Topic Analysis

For the second thrust of our paper, we used a topic analysis to understand the nature of the bugs by extracting topics from bug reports. We used the bug title, bug description and comments for topic extraction. We applied several standard text retrieval and processing techniques for making text corpora amenable to text analyses [12] before applying LDA: stemming, stop-word removal, non-alphabetic word removal, programming language keyword removal. We then used MALLET [13] for topic training. The parameter settings are presented in Section IV-A.

[3]We use the priority definition from Bugzilla: http://wiki.eclipse.org/WTP/Conventions_of_bug_priority_and_severity.

III. Quantitative Analysis

The first thrust of our study takes a quantitative approach to investigating the similarities and differences between bug-fixing processes on severity classes. Specifically, we are interested in how bug-fixing process attributes differ across severity classes on each platform; how the contributor sets (bug reporters and bug owners) differ across classes; and how the contributor profiles vary over time.

The quantitative thrust is centered around several specific research questions:

RQ1 *Does the severity level change and if so, how does it change?*

RQ2 *Are bugs in one severity class fixed faster than in other classes?*

RQ3 *Do bug reports in one severity class have longer descriptions than in other classes?*

RQ4 *Are bugs in one severity class more commented upon than in other classes?*

RQ5 *Do bugs in one severity class have larger comment sizes than other classes?*

RQ6 *How do severity classes differ in terms of bug owners, bug reporters, and owner or reporter workload?*

RQ7 *Are developers involved in handling one severity class more experienced than developers handling other classes?*

A. Severity Change

As mentioned in Section I, bug severity is not always fixed for the lifetime of a bug—after the initial assignment by the bug reporter, the severity can be changed, e.g., by the developers: it can be changed upwards, to a higher value, when it is discovered that the issue was more serious than initially thought, or it can be changed downwards, when the issue is deemed less important that thought initially. In either case, the result is that not enough resources (or too many resources, respectively) are allocated to a bug, which not only makes the process inefficient, but it affects users negatively, especially in the former case, as it prolongs bug-fixing. Hence severity changes should be avoided, or at least minimized. We now quantify the frequency and nature of severity changes.

We show the percentage of bugs that have at least one change in severity in the 3rd (desktop) and 7th (Android) columns of Table I, respectively. The overall change rates are less than 10% for both desktop and Android.[4] The low change rates in both platforms[5] indicate that bug reporters are accurate in assessing severity when a bug is reported. Still, for Android, the change rate is higher in some projects, e.g., AnkiDroid and WebSMS. In the next step, we will shed light on the nature of severity changes.

We found that, for those bug reports that did change severity, 89.40% of desktop bugs and 98.63% of Android bugs have

[4]For desktop, severity change is not available for projects hosted on JIRA; we marked these as 'NA' in the table.

[5]We investigated the reasons for high change rates in AnkiDroid (50%) and found that it was apparently due to a project-specific configuration—large numbers of bugs being filed with severity initially set to 'undecided'.

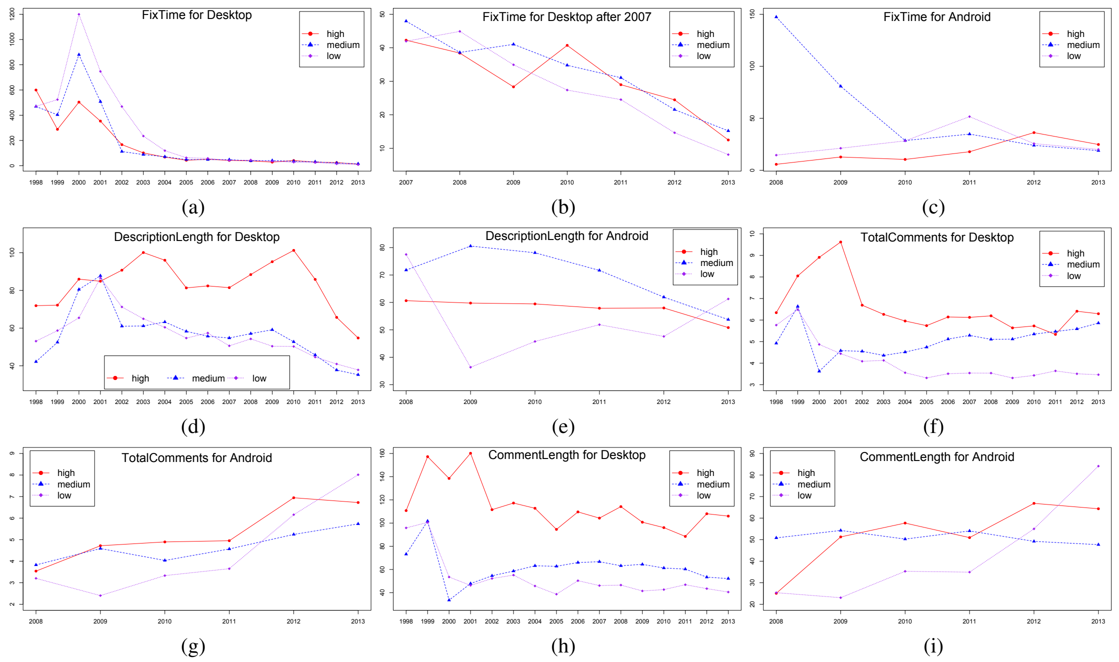


Fig. 1: Feature distribution per year; units are defined in Section II-D.

TABLE II: Top 5 severity change patterns.

Rank	Desktop	Android
1	Normal→Major (18.2%)	Medium→high (33.5%)
2	Normal→Critical (16.3%)	Medium→Low (13.2%)
3	Normal→Enhancement (13.1%)	Medium→Critical (12.7%)
4	Normal→Minor (8.3%)	Undecided→High (8.9%)
5	Normal→Blocker (3.9%)	Undecided→Normal (5.5%)

changed severity once; 8.6% of desktop bugs and 1.11% of Android bugs changed severity twice; finally, only 2% of desktop bugs and 0.26% of Android bugs have more than 3 severity changes. Repeated severity changes naturally lead to higher bug-fixing effort and longer fix times. For example, Firefox bug #250818 changed severity level *13 times*, the highest number of severity changes of all the bugs. It took developers 329 days to finally fix this bug. Next, we are going to show how severity changes.

We show the top-5 most common severity change patterns in Table II.[6] We found that 'Normal→Major' and 'Medium→High' are the most common severity changes for desktop and Android, respectively. We also found the common pattern 'Undecided→High' on Android platform. These patterns indicate that *bug reporters tend to underestimate bug severity more than they overestimate it*. For instance, Mozilla Core bug #777005 was assigned Normal severity initially, but the severity level increased from Normal to Critical, and eventually further increased to Blocker, which indicates the bug had to be fixed as soon as possible. The pattern 'Undecided →High' only exists on Android since issue tracking systems on desktop have Normal as default severity level.

***RQ1**: Less than 10% of bugs change severity on both desktop and Android; in those cases where severity does change, it tends to only change once. Normal→Medium and Major→High are the most common change patterns on desktop and Android, respectively.*

B. Bug-fix Process Attributes

We now proceed with the quantitative analysis of bug characteristics and bug-fixing process features. Rather than presenting aggregate values across the entire time span, we analyze the evolution of values on each platform, at yearly granularity, for two main reasons: (1) as feature values change over time, changes would not be visible when looked at in aggregate, and (2) we want to study the trends of severity classes. The evolution graphs, presented in Figures 1 through 5, will be discussed at length.

Data preprocessing. For each feature, e.g., FixTime, we compute the geometric mean[7] for feature values in each year. Moreover, to avoid undue influence by outliers, we have excluded the top 5% and bottom 5% when computing and plotting the statistical values.

Pairwise tests between classes. To test whether the differences in bug characteristics between classes are significant, we performed a non-parametric Mann-Whitney U test (aka Wilcoxon rank sum test) *for each year, for each platform* comparing the distributions of each attribute, e.g., FixTime for high vs medium severity, high vs. low severity, and medium vs. low severity. For brevity, when we discuss the results of the pairwise tests, we only provide summaries, e.g., the year, platform, and class pair for which differences are significant.

FixTime: Figures 1a–1c show how the bug fixing time has changed over the years for each severity class on desktop and Android. Since the values after 2007 on desktop are much smaller than those pre-2007, we provide a zoom-in of

[6]We only used the 10-point scale in Table II since the 3-category scale would be too coarse-grained. The rest studies only use the 3-category scale.

[7]Since the distributions are skewed, arithmetic mean is not an appropriate measure [14], and we therefore used the geometric mean in our study.

TABLE III: Results of the generalized regression model.

Features	Desktop		Android	
	coefficient	p-value	coefficient	p-value
Severity	-95.850	$< 2e$-16	-20.892	$6.44e$-13
Priority	4.162	0.108	-14.891	0.00109

FixTime for years 2007 to 2013 in Figure 1b. We make several observations regarding FixTime.

We found that *in addition to severity, priority also affects FixTime*. However, not all bug reports have an associated priority, as not all bug trackers support it: for our datasets, 67.68% of desktop bugs have an associated priority while only 6.44 of Android bugs do (Google Code does not support priority). We first provide several examples of how priority and severity can influence FixTime; later we will show the results of a statistical analysis indicating that severity influences FixTime more than priority does:

- *High Severity & High Priority:* major functionality failure or crash in the basic workflow of software. These bugs usually took less time to fix since they have huge and deleterious effects on software usage. For instance, Mozilla Core bug #474866, which caused plugins to fail upon the second visit to a website, is a P1 blocker bug; it took developers just two days to fix it.
- *High Severity & Low Priority:* the application crashes or generates an error message, but the cause is a very specific situation, or a hard-to-reproduce circumstance. Usually developers need more time to fix these kind of bugs. For example, Mozilla Core bug 92322 had severity Blocker (highest) but priority P5 (lowest). The bug took 2 months to fix because it required adding functionality for an obscure platform (at the time).
- *Low Severity & High Priority:* this characterizes bugs such as a typo in the UI, that do not impact functionality, but can upset or confuse users, so developers try to fix these bugs quickly. For instance, Eclipse JDT bug #13141, whose severity is trivial, had priority P1 since it was a typo in the UI; it was fixed in one day.
- *Low Severity & Low Priority:* bugs in this class comprise nuisances such as misspellings or cosmetic problems in documents or configuration files. Developers would fix these bugs when the workload is low. For example, Apache httpd bug #43269, a typo in the server error message with trivial severity and P5 priority, took more than 3 years to be fixed.

To check the influence of severity and priority on FixTime, we built a linear regression model in which FixTime was the dependent variable, while severity and priority were independent variables. Table III shows the results. We found that severity is a better FixTime predictor than priority, with p-values much smaller than those of priority, but nevertheless the priority's p-values, 0.1 for desktop and 0.001 for Android suggest a relationship between priority and FixTime.

We now present the results of a per-year statistical analysis that shows FixTimes tend to differ significantly among severity classes and among priority classes. The "Severity" columns of Table IV provide the p-values of pairwise two-means tests of FixTimes between different severity classes. The results

TABLE IV: Significance values of whether FixTime differs between classes on Desktop and Android.

Year	Severity			Priority		
	Hi. vs. Med.	Hi. vs. Low	Med. vs. Low	Hi. vs. Med.	Hi. vs. Low	Med. vs. Low
Desktop						
1998	0.0189	0.2183	0.5654	< 0.01	< 0.01	0.7325
1999	< 0.01	< 0.01	< 0.01	< 0.01	0.0102	< 0.01
2000	< 0.01	< 0.01	< 0.01	0.3180	< 0.01	0.0346
2001	< 0.01	< 0.01	< 0.01	< 0.01	0.6562	< 0.01
2002	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
2003	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
2004	0.4082	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
2005	< 0.01	< 0.01	< 0.01	0.9191	< 0.01	< 0.01
2006	0.2646	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
2007	< 0.01	0.9771	0.0102	< 0.01	< 0.01	< 0.01
2008	0.9800	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
2009	< 0.01	< 0.01	< 0.01	< 0.01	0.7295	< 0.01
2010	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
2011	0.2227	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
2012	< 0.01	< 0.01	< 0.01	< 0.01	0.0267	< 0.01
2013	< 0.01	< 0.01	< 0.01	< 0.01	0.2575	0.2535
Android						
2008	< 0.01	0.0831	< 0.01	NA	NA	NA
2009	< 0.01	0.2286	< 0.01	< 0.01	NA	NA
2010	< 0.01	< 0.01	0.9944	< 0.01	NA	NA
2011	< 0.01	< 0.01	0.0105	0.0429	0.2958	0.0304
2012	< 0.01	0.0604	0.3382	0.0347	0.0711	0.9072
2013	< 0.01	0.1118	0.4905	0.4140	0.5035	0.8857

indicate that on desktop, with few exceptions (1998, 2007), FixTimes do differ significantly between classes. On Android, though, FixTimes differ less among severity classes.

The "Priority" columns of Table IV show the results of a similar analysis—how does FixTime differ between priority classes. To make comparisons with severity easier, we used a 3-point scale for priority.[8]

The table values indicate that differences in FixTime across priority classes again tend to be significant on desktop, but not on Android; the "NA" entries indicate that most of the Android bugs in that class were hosted on Google Code, which does not support priority.

For Android, FixTime differs significantly between high and medium severity bugs in all years, but does not differ significantly between high and low severity or between medium and low severity: this is explained by the fact that the Wilcoxon test is not transitive [15]. For high severity vs. low severity, only 2010 and 2011 have significant differences; for medium severity vs. low severity, the differences are only significant in 2008 and 2009.

Figures 1a–1c indicate how the population means vary for each class, year, and platform. We can now answer RQ2:

RQ2*: Fix time for desktop bugs is affected by severity and to a lesser extent by priority. FixTime does vary significantly across severity classes for desktop, but for Android the only significant difference is between high and medium severity.*

DescriptionLength: The number of words in the bug description reflects the level of detail in which bugs are described. A higher DescriptionLength value indicates a higher bug report quality [9], i.e., bug fixers can understand and find the correct fix strategy easier. As shown in Figure 1d, high-severity bugs on desktop have significantly higher description length values while medium and low-severity bugs have lower values (the DescriptionLength differences between medium and low-severity bugs is significant in only 10 out of 16 years). We found that the reason for high DescriptionLength for high-severity bugs is that reporters usually provide stack traces or error logs for the bugs.

[8]*High* means priority level ≤ 2; *Medium* means priority level=3; and *Low* means priority level ≥ 4.

However, as shown in Figure 1e, we found different trends on Android: medium severity bugs have the highest values of DescriptionLength, followed by high-severity bugs, while low-severity bugs usually have the smallest DescriptionLength. The difference between classes is small. The pairwise tests show that only during half of the studied time frame (2009–2011), the differences between severity classes are significant, for other years they are not. The reason for high-severity bugs not having the highest DescriptionLength is that unlike on desktop, for projects hosted on Google Code, reporters do not adhere to providing a stack trace or error log as strictly as they do on desktop.

***RQ3**: On desktop, DescriptionLength is significantly higher for high-severity bugs compared to medium and low-severity bugs. DescriptionLength differences are not always significant between medium and low-severity on desktop, or between classes on Android.*

TotalComments: Bugs that are controversial or difficult to fix have a higher number of comments. The number of comments can also reflect the amount of communication between application users and developers—the higher the number of people interested in a bug report, the more likely it is to be fixed [16]. According to Figure 1f, there are more comments in high-severity bug reports on desktop, while low-severity bug reports have the least number of comments. This indicates that high-severity bugs are more complicated, and harder to fix, while low-severity bugs are in the opposite situation.

The pairwise tests show that all classes are different from each other except for a few years (medium vs. low in 1999 and 2001).

TotalComments evolution is similar on Android (Figure 1g), i.e., high-severity bugs have the highest value while low-severity bugs have the lowest value. The pairwise tests indicate that differences are significant between all class pairs, except for 2008 and 2009.

***RQ4**: On desktop and Android, high-severity bugs are more commented upon than the other classes, whereas low-severity bugs are less commented upon.*

CommentLength: This measure, shown in Figures 1h and 1i, bears some similarity with TotalComments, in that it reflects the complexity of the bug and activity of contributor community. We found similar results as for TotalComments on desktop. Pairwise tests indicate that high-severity bugs do differ from medium and low in all the cases while medium and low-severity bugs have significant differences only after 2004.

For Android, the trends are not so clear: the pairwise tests show that the difference between high and medium are not significant in 2009 to 2011. But differences between high and low, and medium and low, are significant in all years.

***RQ5**: High severity bugs on desktop have higher CommentLength than other classes. On Android, the differences between high and medium severity classes are not significant, but they both are significantly higher than for the low-severity class.*

TABLE V: Numbers of bug reporters and bug owners and the percentage of them who contribute to each severity class.

Year	**Reporters**				**Owners**			
	#	*(%)*			#	*(%)*		
		High	Med.	Low		High	Med.	Low
Desktop								
1998	164	33.5	76.2	17.1	64	43.8	78.1	29.7
1999	949	45.6	75.3	17.8	214	58.9	86.4	38.3
2000	3,270	35.0	76.1	15.7	449	42.5	80.0	35.0
2001	5,471	29.4	71.7	17.7	664	44.4	69.0	39.2
2002	7,324	35.5	55.6	18.5	995	41.2	60.5	33.9
2003	7,654	29.5	52.2	18.0	1,084	34.3	55.2	31.9
2004	8,678	28.5	52.5	21.0	1,273	36.4	56.2	32.8
2005	8,990	28.2	47.8	21.0	1,327	37.8	56.7	33.5
2006	7,988	30.7	51.2	21.8	1,408	39.9	58.9	33.7
2007	7,292	30.0	52.5	19.7	1,393	39.1	64.0	32.7
2008	8,474	30.9	55.5	20.4	1,546	39.7	65.3	32.5
2009	8,451	32.6	56.2	20.1	1,537	41.9	64.7	34.2
2010	7,799	34.0	56.6	18.0	1,475	45.5	65.5	32.4
2011	6,136	33.2	64.2	17.9	1,381	43.7	75.7	31.1
2012	5,132	32.1	67.9	17.1	1,352	47.4	76.0	29.6
2013	4,884	31.2	66.6	18.1	1,432	47.3	72.8	27.7
Android								
2008	429	4.4	97.7	2.6	41	36.6	95.1	17.1
2009	987	8.1	95.1	2.9	104	29.8	92.3	12.5
2010	1,875	12.6	87.0	7.3	163	33.1	88.3	21.5
2011	2,045	10.6	91.5	4.1	218	33.0	93.6	16.1
2012	1,998	11.6	89.6	6.4	340	26.8	87.9	20.3
2013	1,492	7.8	84.2	11.3	419	16.2	89.5	29.4

C. *Management of Bug-fixing*

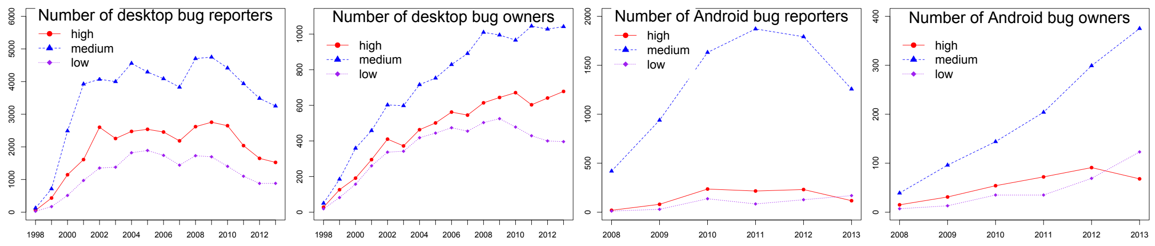
Resource allocation and management of the bug-fixing process have a significant impact on software development [11]; for example, software quality was found to be impacted by the relation between bug reporters and bug owners [6]. We defined the BugOwner and BugReporter roles in Section II-D and now set out to analyze the relationship between bug reporters and bug owners across the different severity classes.

1) Developer Changes: We examined the distributions and evolutions of bug reporters, as well as bug owners, for the three severity classes on the two platforms. Table V summarizes the results. Column 2 shows the total number of bug reporters in each year; columns 3–5 show the percentages of bug reporters who have reported high, medium, and low-severity bugs. Columns 6 through 9 show the numbers and percentages of bug owners.

We make several observations. First, the sums of percentages for high, medium, and low severity are larger than 100%, which indicates that there are reporters or owners who contribute to more than one severity class. Second, high-severity bug owners have larger contribution (percentage) values than those of corresponding bug reporters, because fixing high-severity bugs requires more resources.

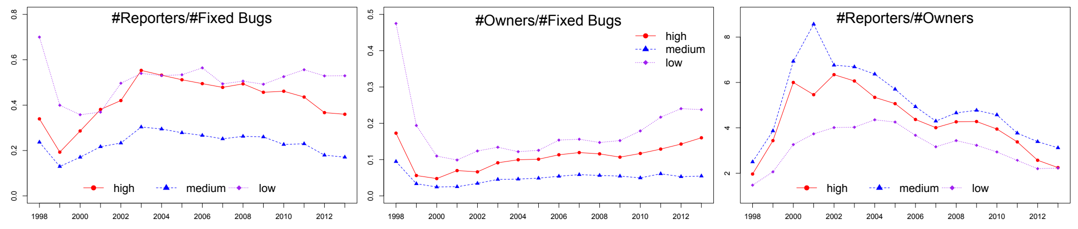
Figure 2 shows the trend of bug reporters and owners of each severity class. For desktop, according to Figures 2a and 2b, we found similar evolutions for the numbers of high, medium and low severity classes for both bug reporters and owners. For Android, Figures 2c and 2d indicate that there are many more medium severity reporters and owners than other severity classes. The differences in these trends between desktop and Android are due to the percentage of medium severity bugs on Android (Table I) being larger than the percentage of medium-severity bugs on desktop.

The number of fixed bugs differs across platforms, so to be able to compare reporter and owner activity between platforms, we use the number of bug reporters and bug owners in each year divided by the number of fixed bugs in that year.



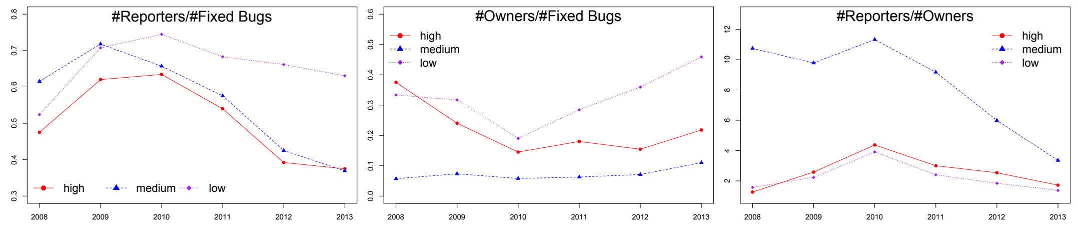
(a) (b) (c) (d)

Fig. 2: Number of bug reporters and owners for each platform.



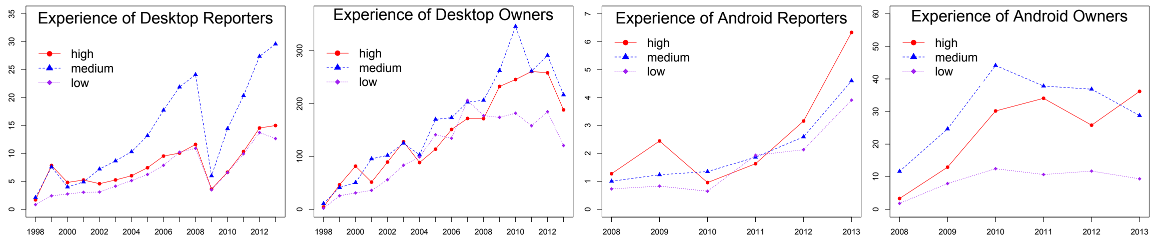
(a) (b) (c)

Fig. 3: Developer trends on desktop.



(a) (b) (c)

Fig. 4: Developer trends on Android.



(a) (b) (c) (d)

Fig. 5: DevExperience for each severity class and each platform, in days.

Figures 3a, 3b, 4a and 4b show the results; we will explain them one by one.

For desktop, the ratio of reporter or owner to fixed bugs have similar trends (Figures 3a, 3b): low-severity bugs have large values as they are easier to find and fix. They are followed by high-severity bugs since, although hard to fix, the importance of high-severity bugs plays a significant role and instills urgency. For Android, the ratios of bug owners to fixed bugs (Figure 4b) have similar trends as on desktop, since the bug-fixing process is similar on both platforms. On the contrary, the ratio of bug reporters to fixed bugs (Figure 4a) on Android and desktop are different. On Android, high severity has the lowest value for this ratio while the medium severity class has the lowest value on desktop. The reason is that the percentage of high-severity bugs on Android is less than that of desktop. Also the severity classification on Android is not as strict as on desktop (most projects hosted on Google Code only have 3 severity levels); and finally, the difference between high and medium is smaller on Android.

Furthermore, the ratio of owners to fixed bugs can reflect the inverse of workload and effort associated with bug-fixing (high ratio value = low workload). We find that for both desktop and Android (Figures 3b and 4b), low-severity bugs have the lowest workload (the highest curve) since they are easiest to fix. Medium-severity bugs require most resources due to their large quantity.

The ratio of reporters to owners (Figures 3c and 4c) reveals that medium-severity bugs have the highest ratio— this is unsurprising since most reports are at this severity level. Low severity has the smallest ratio as these bugs are the easiest to report and fix.

***RQ6**: Medium-severity bugs have the most reporters, most* *owners, and highest workload. Low-severity bugs are at the opposite end of the spectrum; high-severity bugs are in-between.*

2) Developer Experience: To analyze differences in contributors' level of experience, we use the DevExperience metric defined in Section II-D.[9] Figure 5 shows the evolution of DevExperience, in days, for bug reporters and bug owners of each severity class on desktop and Android. For bug reporters and owners, the experience on all severity classes increases with time.[10]

For desktop, as Figures 5a and 5b show, bug reporters and bug owners of medium-severity bugs are more experienced than the other two severity classes due to there being more medium-severity bugs, hence contributors can gain more experience. Low-severity bug reporters and owners have the lowest levels of experience since the number of low-severity bugs is small and these bugs are easiest to find and fix.

For Android, bug owners have similar trends as on desktop, medium-severity bug owners have more experience than high and low-severity owners. On the other hand, high-severity bug reporters on Android are more experienced than medium severity than low severity, but the difference in experience levels between severity classes is smaller than on desktop. Upon further investigation, we found that the portion of new bug reporters on Android is much larger than that of desktop, which we attribute to the lower "barrier to entry": it is easier to report a bug in an Android project than in a desktop project since most apps include a streamlined bug reporting functionality (send the crash report and/or Android `logcat` to developers by just pressing a button).

Also as Figure 4a shows, the high-severity class has the lowest ratio of reporters-to-fixed bugs, which indicates there are fewer new bug reporters for high severity. As a result, high-severity bug reporters are more experienced than others on Android.

RQ7*: Medium-severity class developers are more experienced (have been active for longer) than high-severity class developers than low-severity class developers.*

IV. Topic Analysis

So far we have focused on a *quantitative analysis* of bugs and bug fixing. We now turn to a *topic analysis* that investigates the nature of the bugs in each severity class. More concretely, we are interested in what kinds of bugs comprise each severity class on different platforms and how the nature of bugs changes as projects evolve.

We use topic analysis for this purpose: we extract topics (sets of related keywords) via LDA from the terms (keywords) used in bug title, descriptions and comments, as described in Section II-E. We extract the topics used each year in each severity class of each platform, and then compare the topics to figure out how topics change over time in each class and how topics differ across severity classes.

For the topic analysis, to investigate the difference between severity classes on bug nature, we designed the following research questions:

RQ8 *Do bug topics differ between severity classes?*
RQ9 *How do topics evolve over time?*

A. Topic Extraction

The number of bug reports varies across projects, as seen in Table I. Moreover, some projects are related in that they depend on a common set of libraries, for instance SeaMonkey, Firefox and Thunderbird use functionality from libraries in Mozilla Core for handling Web content. It is possible that a bug in Mozilla Core cascades and actually manifests as a crash or issue in SeaMonkey, Firefox, or Thunderbird, which leads to three separate bugs being filed in the latter three projects. For example, Mozilla Core bug #269568 cascaded into another two bugs in Firefox and Thunderbird. A similar issue may appear in projects from the KDE suite, e.g., Konqueror, Kdevelop, or Kate might crash (and have a bug filed) due to a bug in shared KDE libraries.

Hence we extract topics using a *proportional* strategy where we sample bug reports to reduce possible over-representation due to large projects and shared dependences.

More concretely, for high/medium/low severity classes on desktop, we extracted topics from 500/1,000/400 "independent" bug reports for each severity class, respectively. The independent bug report sets were constructed as follows: since we have 10 projects from KDE, we sampled 100 medium severity bugs from each KDE-related project. We followed a similar process for Mozilla, Eclipse and Apache. Android projects had smaller number of bug reports, so for Android we sampled 50/100/50 bug reports from high/medium/low severity classes, respectively.

With the proportional sets at hand, we followed the LDA preprocessing steps described in Section II-E; since there are only two bug reports in 1998 for desktop and one for Android in 2007, we have omitted those years. For desktop, the preprocessing of high, medium, and low severity sets resulted in 755,642 words (31,463 distinct), 758,303 words (36,445 distinct) and 352,802 words (19,684 distinct), respectively. For Android, the preprocessing of high, medium, and low severity sets resulted in 116,010 words (7,230 distinct), 289,422 words (13,905 distinct) and 31,123 words (3,825 distinct), respectively. Next, we used MALLET [13] for LDA computation. We ran for 10,000 sampling iterations, the first 1,000 of which were used for parameter optimization. We modeled bug reports with K = 100 topics for high and medium severity classes on desktop, 60 for low severity class on desktop, 50 for high and medium classes on Android and 40 for low severity bugs on Android; we choose K based on the number of distinct words for each platform; in Section V we discuss caveats on choosing K.

B. Bug Nature and Evolution

We now set out to answer RQ8 and RQ9.

[9]There are few developers made a lot contribution over a short period while some made only few contributions over a long period. Since the situation are extremely rare, they will not affect the overall result.

[10]The dip in DevExperience for bug reporters on desktop in 2009 is caused by a large turnover that year; the fast rise afterwards is due to a surge in popularity of several projects (mainly Gnome Core, Amarok and Plasma).

TABLE VI: Top words and topic weights for desktop.

Label	Most representative words	Weights
High		
build	build gener log option link config select resolv duplic depend level	12%
compile	server sourc compil output local project info path search util tag expect tool	11%
crash	crash load relat size caus broken good current instanc paramet trace stop properli valid trigger affect unabl assum condition	9%
GUI	swt widget ui editor dialog jface gnome content enabl handler mod send kei	6%
concurrency	thread lwp pthread event wait process mutex cond thread oper qeventdispatch qeventloop	5%
Medium		
application logic	window call page displai log connect start add data output check support current appli enabl apach	17%
install	instal select item control option move linux directori core debug icon start correct server thing info save statu place edit account	15%
crash	warn good crash layout limit buffer affect lock confirm miss screenshot trigger quick	8%
widget	widget descript action select plugin workbench dialog swt progress jdt wizard max	6%
communi-cation	ssh debug local kei ssl protocol sshd authent messag password cgi login client channel launch exec	3%
Low		
config	configur support sourc instal size exist url inform util cach document info load custom src path move	22%
feature request	add option suggest gener find updat good local applic miss content address point data correct bit save duplic	20%
I/O	output space mous index block invalid separ net oper workaround stop foo wait timeout delet keyboard	9%
install	instal makefil icon dialog home build edit helper select normal page leav progress site bugzilla verifi	8%
database	db queri method menu php filter map sql jdbc execut databas count gecko init	8%

TABLE VII: Top words and topic weights for Android.

Label	Most representative words	Weights
High		
concurrency	handler init handl dispatch looper event htc samsung galaxi loop sm mobil enabl post option launch miss	22%
runtime error	runtim error fail code press screen happen doesn exit queri finish notic process edit didn wait lock delet trace	16%
runtime crash	crash thread patch doesn repli state code updat stack good resourc beta hit verifi unknown window	13%
Medium		
runtime crash	runtim doesn result fail item remov wrong happen input action file app system data	27%
runtime error	output error messag correct forc due requir complet specif debug occur count	19%
phone call	call devic task sip galaxi samsung servic hardwar motorola	12%
Low		
feature request	suggest client find guess support messag phone output account system log option applic check format displai send screen result user market latest	23%
application logic	android app file error correct wrong bit page librari nofollow map correctli didn fail full data launch logcat screenshot	21%
GUI	report menu screen button gener ad user link browser load press item url mode action context keyboard previou widget	12%

How Do Bug Topics Differ Across Severity Classes: Tables VI and VII show the highest-weight topics extracted from the proportional data set. We found that for both desktop and Android, bug topics with highest weight differ across severity classes.

On desktop, build- and compilation-related bugs are the most common bug type in the high-severity class; for medium severity, application logic bugs (failure to meet requirements) are the most popular, followed by installation errors; for low severity, configuration bugs and feature requests are most prevalent. It was interesting to see that bug severity is somewhat more developer-centric than user-centric: note how build/compile errors which mostly affect developers or highly-advanced users appear in the high-severity class, whereas install errors, which mostly affect end-users, are in the medium-severity class.

TABLE VIII: Top-3 bug topics per year for high-severity class on desktop and Android.

Year	Top 3 topics (topic weight)		
Desktop			
1999	make (58%)	install (12%)	layout (10%)
2000	widget (27%)	layout (24%)	install (16%)
2001	layout (38%)	install (10%)	compile (7%)
2002	GUI (17%)	compile (14%)	build (10%)
2003	compile (20%)	GUI (13%)	build (10%)
2004	crash (19%)	build (12%)	compile (9%)
2005	plugin (17%)	build (10%)	compile (10%)
2006	GUI (14%)	build (10%)	compile (8%)
2007	build (14%)	GUI (11%)	compile (9%)
2008	debug (16%)	compile (11%)	widget (9%)
2009	concurrency (33%)	GUI (11%)	compile (11%)
2010	concurrency (15%)	compile (11%)	debug (7%)
2011	concurrency (16%)	compile (12%)	plugin (11%)
2012	cloud (14%)	compile (12%)	build (11%)
2013	concurrency (22%)	build (11%)	cloud (11%)
Android			
2008	email (39%)	runtime error (24%)	concurrency (11%)
2009	connection (23%)	runtime error (22%)	concurrency (14%)
2010	concurrency (18%)	database (17%)	call (13%)
2011	map (27%)	concurrency (26%)	database (12%)
2012	concurrency (21%)	runtime crash (17%)	runtime error (17%)
2013	browser (18%)	runtime crash (16%)	concurrency (15%)

For Android, in the high-severity class, bugs associated with the Android concurrency model (event/handler) are the most prevalent, followed by runtime errors (the app displays an error) and then runtime crashes (the app silently crashes without an error or restarts); in the medium-severity class, runtime crashes and runtime errors are also popular, followed by problems due to phone calls. In the low-severity class, feature requests are the most popular, followed by problems due to application logic and GUI.

While some commonalities exist between desktop and Android (runtime errors, crashes, and feature requests are well-represented in the high-weight topics on both platforms), there are also some notable differences: build/compile/install errors do not pose a problem on Android, which might suggest that the Android build process is less error-prone. Second, owing to the smartphone platform, phone call issues appear more frequently among high-weight topics on Android. Third, concurrency is still posing problems: it appears as a high-weight topic on both desktop on Android.

RQ8*: Topics differ across severity classes and across platforms, e.g., for high-severity bugs, build/install/compile-related issues are the most prevalent on desktop, while concurrency and runtime error/crash-related bugs are the most prevalent on Android.*

How Do Bug Topics Evolve: To study macro-trends in how the nature of bugs changes over time, we analyzed topic evolution in each severity class on each platform. We found that high-severity bugs on desktop are the only class where topics change substantially over time; in the other classes, high-frequency topics tend to be stable across years. We limit our discussion to high-severity topics; Table VIII shows the top-3 topics and their corresponding weight for each year.

We make several observations. On desktop, concurrency started to be a big issue in 2009, and has remained so. This is unsurprising, since multi-core computers have started to become the norm around that time, and multi-threading programming has been seeing increased adoption. Furthermore, we found that cloud computing-related bugs have started to appear in 2012, again understandably as cloud computing has been getting more traction.

For Android, concurrency bugs rank high every year, suggesting that developers are still grappling to use the Android's event-based concurrency model correctly.

RQ9*: Bug topics tend to be stable in low- and medium-severity classes. In the high-severity class, on desktop, there have been more concurrency and cloud-related bugs since 2009 and 2012, respectively; on Android, concurrency bugs have been and continue to be prevalent.*

V. Threats to Validity

We now discuss possible threats to the validity of our study.

Selection bias. We only chose open source applications for our study, so the findings might not generalize to closed-source projects. Our chosen projects use one of five trackers (Bugzilla, Trac, JIRA, MantisBT and Google Code); we did not choose projects hosted on GitHub since severity levels are not available on GitHub, hence our findings might not generalize to GitHub-hosted projects.

Furthermore, we did not control for source code size—differences in source code size might influence features such as FixTime.

Severity distribution on Android. According to Table I, many Android projects have skewed severity distributions (for 17 out of 38 Android projects, more than 90% of the bugs have medium severity) We believe this to be due to medium being the default value for the severity field on Google Code and most Android reporters or developers on Android not considering the severity level as important as on desktop.

Priority on Google Code and JIRA. We could not quantify the effect of priority on those projects hosted on Google Code and JIRA, as Google Code and JIRA do not have a priority field.

Data processing. For the topic number parameter K, finding an optimal value is an open research question. If K is too small, different topics are clustered together, if K is too large, related topics will appear as disjoint. In our case, we manually read the topics, evaluated whether the topics are distinct enough, and chose an appropriate K to yield disjoint yet self-contained topics.

Google Code does not have support for marking bugs as reopened (they show up as new bugs), whereas the other trackers do have support for it. About 5% of bugs have been reopened on desktop, and the FixTime for reopened bugs is usually high [17]. This can result in FixTime values being lower for Google Code-based projects than they would be if bug reopening tracking was supported.

VI. Related Work

Empirical software engineering studies have focused on studying bug characteristics, but a study comparing bug reports/bug-fixing processes/nature of bugs between the severity classes has been missing.

Cross-platform studies. In our previous study [**?**], we compared bugs and bug-fixing features between desktop, Android and iOS. This work uses a subset (desktop and Android) of those datasets. However, the thrust of this work is understanding differences (bug characteristics, bug topics) between severity classes rather than between platforms.

Bug severity studies. All the existing bug severity studies are focused on predicting severity levels from a newly-filed bug report. Menzies et al. [3] proposed a classification algorithm named RIPPER and applied it to bug reports in NASA to output fine-grained severity levels.

Lamkanfi et al. [2] apply various classification algorithms to compare their performance on severity predicting. They grouped bug reports into two classes, severe and non-severe.

Tian et al. [4] have applied the Nearest Neighbor algorithm to predict the severity of bug reports in a fine-grained way.

All these works are using information retrieval techniques to predict severity level of bug reports, but they did not consider the question whether there are differences in bug characteristics across severity classes. Our findings validate the importance and necessity of these previous works, and show that severity is an important factor not only for bug reporters but also for project managers.

Topic modeling. Topic models have been used widely in software engineering research. Prior efforts have used topic model for bug localization [18], source code evolution [12], duplicate bug detection [19], [20] and bug triaging [21].

Our work applies a similar process with previous work [12], but we use topic modeling technique for a different purpose: finding differences in bug topics across severity classes, and how bug topics evolve over time.

VII. Conclusions

We have presented the results of a study on desktop and Android projects that shows bugs of different severity have to be treated differently, as they differ in terms of characteristics and topics. We have defined three severity classes, high, medium and low. We have shown that across classes, bugs differ quantitatively e.g., in terms of bug fixing time, bug description length, bug reporters/owners. A topic analysis of bug topics and bug topic evolution has revealed that the topics of high-severity bugs on desktop have shifted over time from GUI and compilation toward concurrency and cloud, whereas on Android concurrency is a perennial topic. Our approach can guide severity assignment, e.g., compile/make bugs should have higher severity, and configuration errors should have lower severity.

VIII. Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. CCF-1149632. Research was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-13-2-0045 (ARL Cyber Security CRA). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

REFERENCES

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

[2] A. Lamkanfi, S. Demeyer, Q. Soetens, and T. Verdonck, "Comparing mining algorithms for predicting the severity of a reported bug," in *CSMR'11*, 2011, pp. 249–258.

[3] T. Menzies and A. Marcus, "Automated severity assessment of software defect reports," in *ICSM'08*, 2008, pp. 346–355.

[4] Y. Tian, D. Lo, and C. Sun, "Information retrieval based nearest neighbor classification for fine-grained bug severity prediction," in *WCRE'12*, 2012, pp. 215–224.

[5] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *JMLR*, vol. 3, pp. 993–1022, 2003.

[6] A. Bachmann and A. Bernstein, "When process data quality affects the number of bugs: Correlations in software engineering datasets," in *MSR'10*, 2010, pp. 62–71.

[7] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, "What makes a good bug report?" in *FSE'08*, 2008, pp. 308–318.

[8] E. Giger, M. Pinzger, and H. Gall, "Predicting the fix time of bugs," in *RSSE'10*, 2010, pp. 52–56.

[9] P. Hooimeijer and W. Weimer, "Modeling bug report quality," in *ASE'07*, 2007, pp. 34–43.

[10] A. Lamkanfi and S. Demeyer, "Filtering bug reports for fix-time analysis," in *CSMR'12*, 2012, pp. 379–384.

[11] J. Xuan, H. Jiang, Z. Ren, and W. Zou, "Developer prioritization in bug repositories," in *ICSE'12*, 2012, pp. 25–35.

[12] S. W. Thomas, B. Adams, A. E. Hassan, and D. Blostein, "Modeling the evolution of topics in source code histories," in *MSR'11*, 2011, pp. 173–182.

[13] A. K. McCallum, "MALLET: A Machine Learning for Language Toolkit," 2002. [Online]. Available: http://mallet.cs.umass.edu

[14] E. Limpert, W. A. Stahel, and M. Abbt, "Log-normal distributions across the sciences: Keys and clues," *BioScience*, vol. 51, no. 5, pp. 341–352, 2009.

[15] D. L. Gillen and S. S. Emerson, "Nontransitivity in a class of weighted logrank statistics under nonproportional hazards," *Statistics & Probability Letters*, vol. 77, no. 2, pp. 123 – 130, 2007.

[16] P. J. Guo, T. Zimmermann, N. Nagappan, and B. Murphy, "Characterizing and predicting which bugs get fixed: An empirical study of microsoft windows," in *ICSE'10*, 2010, pp. 495–504.

[17] 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," *Emp. Soft. Eng.*, vol. 18, no. 5, pp. 1005–1042, 2013.

[18] A. T. Nguyen, T. T. Nguyen, J. Al-Kofahi, H. V. Nguyen, and T. Nguyen, "A topic-based approach for narrowing the search space of buggy files from a bug report," in *ASE'11*, 2011, pp. 263–272.

[19] A. T. Nguyen, T. T. Nguyen, T. N. Nguyen, D. Lo, and C. Sun, "Duplicate bug report detection with a combination of information retrieval and topic modeling," in *ASE'12*, 2012, pp. 70–79.

[20] P. Runeson, M. Alexandersson, and O. Nyholm, "Detection of duplicate defect reports using natural language processing," in *ICSE'07*, 2007, pp. 499–510.

[21] X. Xie, W. Zhang, Y. Yang, and Q. Wang, "Dretom: Developer recommendation based on topic models for bug resolution," in *PROMISE'12*, 2012, pp. 19–28.