Trendy Bugs

Topic Trends in the Android Bug Reports

Lee Martie, Vijay Krishna Palepu, Hitesh Sajnani, Cristina Lopes
Department of Informatics
University of California, Irvine
Irvine, CA 92697 USA
{lmartie, vpalepu, hsajnani, lopes}@ics.uci.edu

***Abstract*—Studying vast volumes of bug and issue discussions can give an understanding of what the community has been most concerned about, however the magnitude of documents can overload the analyst. We present an approach to analyze the development of the Android open source project by observing trends in the bug discussions in the Android open source project public issue tracker. This informs us of the features or parts of the project that are more problematic at any given point of time. In turn, this can be used to aid resource allocation (such as time and man power) to parts or features. We support these ideas by presenting the results of issue topic distributions over time using statistical analysis of the bug descriptions and comments for the Android open source project. Furthermore, we show relationships between those time distributions and major development releases of the Android OS.**

***Keywords*-bug logs; Android; topics; statistical trend analysis**

I. Introduction

For the 2012 MSR Challenge, inspired by the related work in [2], we present the trends in the discussions that took place among Android developers in the Android open source project issue tracker. Studying these vast volumes of discussions from these knowledgeable people can give an understanding of what the community has been most concerned about. However, the vast volume of discussion documents (greater than 80K) overloads the analyst. We present discussion trends on these documents, giving us a high level perspective on problematic features of the project rather than lower level problematic parts of the project. These trends provide actionable knowledge for managing organizations when making scheduling and resource allocation decisions.

Section II describes the document data used to generate the topic trends. Section III reports on the method used to convert 20,169 bug reports into high level topic trends occurring in them. Section IV presents our results and we conclude in Section V.

II. Input Data

The Android bug XML logs [10], as provided by the MSR 2012 Mining Challenge are extracted from [1]. A sample Android bug log entry is as follows.

```
<bug>
    <bugid>bug number</bugid>
    <title>bug title</title>
    <status>bug status e.g. new, closed, etc.</status>
    <owner>developer who owns the bug</owner>
    <type>type of bug e.g. Defect, enhancement etc.</type>
    <priority>priority of the bug</priority>
    <component>
            component of the project the bug belongs to
    </component>
    <closedOn> when the bug was closed ("null"if not closed)
    </closedOn>
    <stars>how many people voted or starred the issue</stars>
    <reportedBy>email id of person reporting the bug</reportedBy>
    <openedDate>date the bug was files</openedDate>
    <description>description of the bug</description>
    <comment>
             <who>person who commented</who>
            <when>time when commented</when>
            <what>text of the comment</what>
    </comment>
</bug>
```

III. Tools and Methodology

To transform the 20,169 bug reports into topic trends we built a parser to transform the bug tags into Java objects with the Java DOM library. We then created our documents from these bug objects. For a given entry in the bug log, we treat the collective content from the bug title and description as one document. Additionally, the content of the *what* tag of each individual comment for a given bug is treated as one document. Each of these documents is annotated with their creation time-stamps, which is obtained from the *openendDate* tag in the case of a bug and the *when* tag in the case of a comment. This results in the creation of 20,169 documents using bug titles and descriptions, and 67,730 documents using comments, thus generating a corpus of 87,899 documents. The resulting corpus is a collection of time-stamped remarks about the problems in the Android open source project. We then removed default stop words used in the MALLET library [5] from these documents.

Table 1: Bug-topics with Top 4 Descriptive Words in Descending Order of Probabilities – Bug-topics with greater than 40% statistical variance in probabilities over time stamps are emboldened – Emboldened topics are labeled with descriptive names in curly brackets

0 emulator adb android system	**25 {Code Review} – source android review https**	**51 {HTTP} – http thread google www**	**75 {Issues} – problem show issues order**
1 droid motorola android problem	26 iq em eve ed	**52 {Debugger} – debug info system lib**	**76 {Forum Support} – google android forum bugs**
2 {Issue Assignments} – issue engineer assigned work	**27 {Issues} – id google android issues**	53 eclipse android sdk windows	**77 {Eclipse} – java eclipse org internal**
3 permission android permissions app	28 problem issue solution fix	54 phone alarm volume mode	78 language support android keyboard
4 server ssl certificate client	29 view screen touch mode	55 test cts android tests	79 works work fine problem
5 android code api application	30 http android html developer	**56 {Graphics Library} – gl public void int**	80 es itq thatq thereq
6 vpn connect mtpd server	31 text keyboard type key	**57 {Issues} – issue merged duplicate resolved**	81 intent action android true
7 contacts contact phone sync	32 card sd sim memory	58 thread state event wait	**82 {CPU} – timed identity cpu pegged**
8 good great google idea	33 samsung galaxy problem android	59 file files download directory	83 time data make lot
9 nexus issue problem froyo	34 number phone numbers call	60 update problem issue version	84 browser page web android
10 {Email} – email client youq device	**35 {String} – java string public import**	61 gps location maps google	85 problem phone back issue
11 database sqlite android content	**36 {Issue Tracker} – apps bug tracker google**	62 error unable open stack	86 ere youq weq ell
12 dalvikvm ms bytes gc	37 java org apache harmony	63 time date zone timezone	87 message sms messages text
13 hardware bluetooth support android	38 android phone feature google	**64 {XML Schema} – android layout xml id**	88 honeycomb transformer tablet asus
14 email mail gmail exchange	39 call phone calls incoming	65 class method null string	**89 {Build} – build target lib mk**
15 app apps market android	40 bug issue report fixed	**66 {Kernal Code} – git kernel platform android**	**90 {Security} – ca ou cn certificate**
16 settings menu option select	41 proxy address apn ipv	**67 {Device} – android source report devices**	91 activity dialog called call
17 user feature option make	42 eq string uri html	68 mediaplayer audio media xx	92 phone battery time problem
18 music player mp video	43 memory mb size system	69 calendar event events google	93 build version android mobile
19 {Calendar} – calendarparser ad key verify	44 htc desire problem android	70 fix google issue phone	94 image gallery camera images
20 data reset settings phone	45 project file android xml	**71 {Runtime} – java android androidruntime os**	95 system err locale en
21 {Forum Support} – google mobile forum android	46 usb device driver phone	72 google issue comments people	96 protocol canvas draw pppd
22 {Fixed Issues} – fixed sdk release issue	**47 {Media Codecs} – return omxcodec cpp media**	73 font support characters android	97 camera preview bitmap xb
23 screen button home back	48 search list add find	74 wifi network connection connect	98 voice phone bluetooth dial
24 code test log problem	**49 {Runtime} – view android java androidruntime**		99 google account gmail password
	50 bluetooth phone car music		

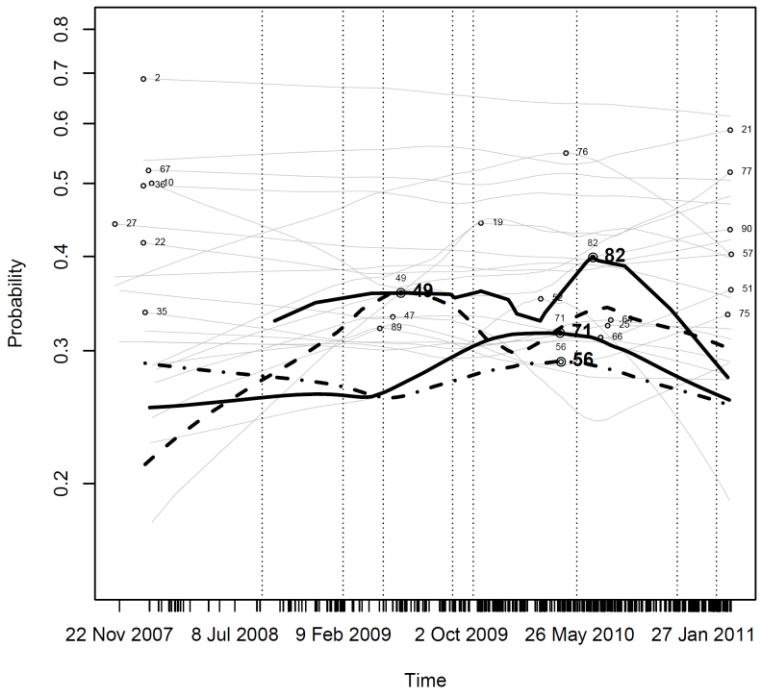

Figure 1: Topic Trends over Time

Once we created these documents, we fitted an LDA [8] model on them using the Gibbs sampling implementation in the MALLET library [5]. LDA is a probabilistic model that associates a probability distribution of topics to each document in a corpus. Each distribution assigns a probability of a topic as describing a document. Further, each topic in LDA is a probability distribution over words in the corpus. This distribution assigns a probability of a word as describing a topic. For an explanation of Gibbs sampling, refer [9]. We ran 3000 sampling iterations and generated 100 topics. Anecdotally, at 3000 iterations the same topics seemed consistently produced; however there could be gains in the quality of topics produced by increasing the number of iterations. We chose to generate 100 topics since it seemed to give us a fine grained level of topics that were mostly interpretable. Choosing the number of topics is often an art and can be adjusted for different results and applications. The important result from running Gibbs sampling is 87,899 topic distributions (one topic distribution for each document). In each topic distribution there are 100 probabilities (one probability for each topic). So we now have a distribution of topics for each document. We can use the probability of a topic in the distribution for a document as a measure for how prominent that topic is in that document.

Next, we replaced documents with their time-stamp to get topic distributions over time. Then we reordered each distribution in the chronological order to get topic distributions from Nov 2007 to March 2011. Thus, resulting in the 87,899 topic distributions, associated with time-stamps, being ordered chronologically. Since, it is possible that documents could have had the same time-stamp we could have multiple topic distributions for the same time-stamp. This means that a topic from each of these distributions may have several probabilities associated with one time-stamp. This issue is addressed via curve fitting as explained later in this section.

We then focused our attention to topics which showcased more than 40% statistical variance in their probabilities, from each distribution, across all time-stamps. We reasoned that such topics could indicate buggy features which would seemingly be resolved to the development community, thus reducing the discussions related to those bugs, only to resurface again, which would be indicated by a surge in the volume of discussion around the topic. This resulted in the 25 bold bug topics as shown in Table 1. Further, curly brackets in Table 1, contain a descriptive name, created manually, for the topics which they annotate.

Our motivation is to provide the analyst with a high level picture of problematic features over time, so we use curve fitting to give the analyst a shape that describes how a topic trends over time. We do this by first making a plot for each topic. Points are created by setting the x-value of a point to a time-stamp and the y-value of that point to the corresponding probability in the topic distribution. In this plot, the y-axis represents the probabilities and the x-axis represents chronologically ordered time-stamps. We then fit a curve to these points to see how the topic trends over time. Additionally, this curve gives each topic one y-value (probability) for each x-value (time-stamp) and addresses the issue of multiple probabilities for one time stamp, as mentioned above.

We use the LOWESS curve fitting algorithm using R, via the Java-R Interface [7], for curve fitting and thus generate our topic trends for the bug data. We lay these trend lines on top of each other for comparison.

Prior to curve fitting the plots of these 25 topics we reduce noisy data by removing all points with probabilities equal to or less than 0.1. We do this because we are only concerned with the behavior of topics when they are being talked about. The final result is shown in Figure 1. Line styles help visually distinguish between the curves. Further, we annotated this plot with vertical lines, at points of release dates of major versions of the Android OS, which is later used in the analysis.

IV. Analysis and Results

A. Bug Topics

The first interesting output that we obtained was a set of 100 bug topics, numbered 0 to 99 as listed in Table 1. This listing shows for each topic, 4 words that are ordered descendingly by their probability of describing the respective topic. Such a listing does not indicate any pattern or trend. Based on the words associated with the bug topics, one can speculate about the topics of discussion in relation with the bug reports. For instance topic 68, 63 and 14 show that there were bugs which resulted in a discussion: on the media player, dates and time zones, and gmail respectively.

B. Time Distribution of Bug Topics

After a careful study of the 25 topics, which are emboldened in Table 1, our attention turned to topics 49: *view android java androidruntime* and 71: *java android androidruntime os*. In one glance one can see the android runtime error was a problematic feature of the Android platform and as such it would be logical to schedule more time and man power to its development. In Figure 1, we noticed that the trend curve for topic 49 showed significant fluctuations. A similar fluctuating trend was also noticeable in the trend line of topic 82: *timed identity cpu pegged*. Initially, we could not discern any direct correlation between the two. However, upon a Google search of the words in topic 82, in that exact order, we were led to a Stackoverflow page [4] that raised a query regarding a “CPU maybe pegged” bug. The query was related to a graphics intensive application based on OpenGL. This motivated us to search for other topics that would be related to OpenGL or Graphics. This search led

us to discover topic 56: *gl public void int,* which exhibited a relation to a graphics library or *gl.* It is interesting to note that this topic too is one of the 25 topics that are emboldened in Table 1. The trend curves for all four topics have been plotted in the graph shown in Figure 1. Interestingly, all these topics show a declining trend after the release of Android 2.3 Gingerbread in December 2010 [3].

We know Android Gingerbread was shipped with a new concurrent Garbage Collector, which was meant to improve application speeds, specifically targeting graphic intensive applications [6]. Thus, by using these trends we can speculate that the new garbage collector may have resolved issues with the Android runtime and graphics applications that use heavy weight graphics libraries, since discussion about it quitted down. One immediate line of investigation then is to inquire the impact garbage collectors have on runtimes. Fixing memory leaks is one example. If the Android management knew the garbage collector was meant to address runtime problems they could see the payoff with the downward trend in the runtime topic after the release of Gingerbread with the new garbage collector. This goes to motivate the study of trends in issue tracking systems as a way to monitor problematic parts or features of projects.

C. Peaks and Trenches of Trend Curves of Bug Topics

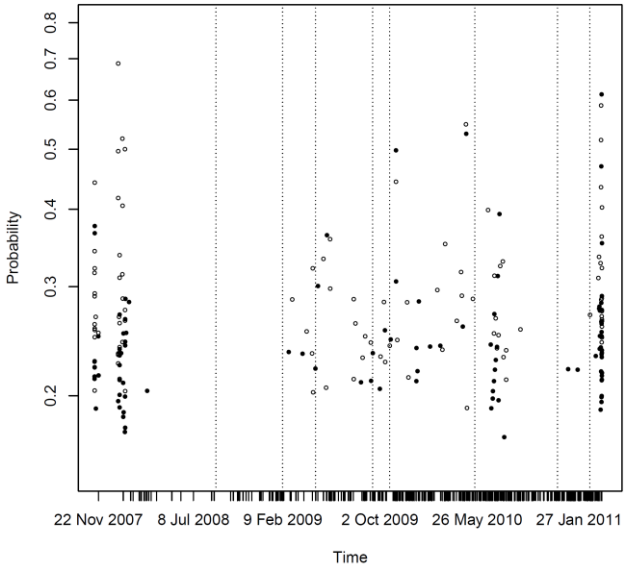

Figure 2: Peaks and Trenches of Trend Curves of Bug Topics

This analysis looks at plotting the peaks and trenches of trend lines, highest and lowest points respectively. We were motivated towards this particular result due to the findings of the previous analysis where we found shifts in trends for topics right before a major release.

These points are plotted as shown in Figure 2. The x and y axes are same as Figure 1. The open dots represent the peaks and the solid dots represent the trenches. Furthermore, we also marked the major release dates for Android starting from version 1.0 to version 4.0. These are represented as nine dotted vertical lines in the graph. The vertical clustering of the peaks and trenches indicate that peaks and trenches usually occur together at the same time. This clustering in the data prompts interesting questions like, is there a causal relation between what is being talked about in the issue tracker and the organization of the Android Team (e.g. scheduling)? As can be seen in Figure 2, the clustering in some instances actually occurs near the vertical lines representing major release dates.

V. CONCLUSION

In this paper we presented an approach to examine the topics of concern for the Android open source project. We modeled discussions in the project's public issue tracker. We showed trends for bug-topics with a statistical variance greater than 40% in their probabilities over all time-stamps. This allowed inspection of specific issues related to Runtime Errors, the Graphics Library and even discovery of a well-known bug called "CPU may be pegged". We note through the trends that the discussion of these issues declined with the release of Android Gingerbread, which introduced a new concurrent garbage collector. Thus, we were able to speculate using the trends that with the release of Gingerbread may have resolved many runtime and graphics related issues. We saw how trend peaks and trenches can cluster together at the same point in time, in vertical lines. Based on these results we believe that modeling discussions over time between developers can give us valuable insight into the state of the project. Future work includes applying formal analysis techniques to the trend lines presented.

REFERENCES

[1] Android Open Source Project - Public Issue Tracker. http://code.google.com/p/android/issues/list

[2] D. Hall, D. Jurafsky, and C. Manning, "Studying the history of ideas using topic models," in EMNLP '08: Proceedings of the Conference on Empirical Methods in Natural Language Processing, Honolulu, Hawaii, 2008, pp. 363–371.

[3] Android 2.3 Gingerbread - http://developer.android.com/sdk/android-2.3 .html

[4] "CPU may be pegged" Bug description on Stackoverflow - http://stackoverflow. com/questions/ 3112284/nexus-one-android-cpu-may-be-pegged-bug

[5] McCallum, Andrew Kachites. "MALLET: A Machine Learning for Language Toolkit." http://mallet.cs.umass.edu. 2002.

[6] Android 2.3 Platform and Updated SDK Tools, Android Developers Blog, http://android-developers.blogspot.com/2010/12/android-23-platform-and-updated-sdk.html.

[7] Java R Interface - http://www.rforge.net/JRI/

[8] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *J. Mach. Learn. Res.*, vol. 3, pp. 993–1022, Mar. 2003.

[9] G. Casella and E. I. George, "Explaining the Gibbs Sampler," *The American Statistician*, vol. 46, no. 3, pp. 167–174, 1992.

[10] E. Shihab, Y. Kamei, and P. Bhattacharya, "Mining Challenge 2012: The Android Platform," in *The 9th Working Conference on Mining Software Repositories*, 2012, p. to appear.