The Firefox Temporal Defect Dataset

Mayy Habayeb, Andriy Miranskyy, Syed Shariyar Murtaza, Leotis Buchanan, Ayse Bener
Data Science Lab, Ryerson University, Canada
Emails: {mayy.habayeb, avm, syed.shariyar}@ryerson.ca, LeotisBuchanan@exterbox.com, ayse.bener@ryerson.ca

***Abstract*—The bug tracking repositories of software projects capture initial defect (bug) reports and the history of interactions among developers, testers, and customers. Extracting and mining information from these repositories is time consuming and daunting. Researchers have focused mostly on analyzing the frequency of the occurrence of defects and their attributes (e.g., the number of comments and lines of code changed, count of developers). However, the counting process eliminates information about the temporal alignment of events leading to changes in the attributes count. Software quality teams could plan and prioritize their work more efficiently if they were aware of these temporal sequences and knew their frequency of occurrence.**

In this paper, we introduce a novel dataset mined from the Firefox bug repository (Bugzilla) which contains information about the temporal alignment of developer interactions. Our dataset covers eight years of data from the Firefox project on activities throughout the project's lifecycle. Some of these activities have not been reported in frequency-based or other temporal datasets. The dataset we mined from the Firefox project contains new activities, such as reporter experience, file exchange events, code-review process activities, and setting of milestones. We believe that this new dataset will improve analysis of bug reports and enable mining of temporal relationships so that practitioners can enhance their bug-fixing process.

I. Introduction

Bug repositories provide a wealth of valuable information for the software research community. Many researchers in various areas of software engineering have attracted several attributes from these repositories [1]–[7]. They used these attributes to build models for time estimation [1], [7], effort estimation [4], [6], etc. The attributes include the number of copied developers, comments exchanged, and developers involved [1], [2]. Researchers use the frequency of occurrence of these attributes within certain time periods.

We mined the Mozilla Firefox web browser, which is well known in the research community [2], [5].This popular open-source software has a large customer base, with more than 450 million users reported worldwide [8]. The Bugzilla defect tracking system stores Firefox defect data. Lamkanfi et al. [9] introduced Eclipse and Mozilla tracking dataset in 2013 after extracting various bug-report attributes from Bugzilla: summaries, short descriptions, severity, priority, version, assignments, people in copy, operating system, product/component, status, and resolution. We build on their effort[1] by adding attributes from the Firefox project focused on tracking key process attributes and their temporal relationships. We track events related to reporter experience, file exchange, code review, comments exchanged between developers, and the involvement of more than one person within a certain time frame.

[1]We have been in contact with the authors of [9] to understand and extend their work.

Consider this example of a chain of activities that we track. A junior developer fixes a bug and sends the code patch to a senior developer for review. The senior developer can approve the request, assign the bug to another developer, or write a comment on the "whiteboard". This sequence of events continues until the bug is resolved. These activities can be useful in variety of ways, including detecting common activities for which a final resolution frequently is reached and determining the time to fix a bug. Our main contribution is, through the Firefox Temporal Defect Dataset (FTDD), to provide researchers with a dataset that reflects the key temporal events which occur during the life cycle of a bug without mining the bug's history files. This dataset can be linked easily to the Mozilla defect tracking set via bug id, allowing researchers to work with the datasets simultaneously. We provide the data in MySQL and CSV formats at the URL: https://googledrive.com/host/0B5TJwohpS9LzcHc3NldtdjZQRFk.

The paper is organized as follows: Section II describes FTDD in detail and provides the rationale for extracting certain attributes, and Section III discusses the methodology and tools used for data extraction. Section IV highlights the key features of the dataset, while Section V examines its possible uses and relevance to the research community. Section VI discusses the challenges and limitations of this work, and finally, Section VII presents the conclusions.

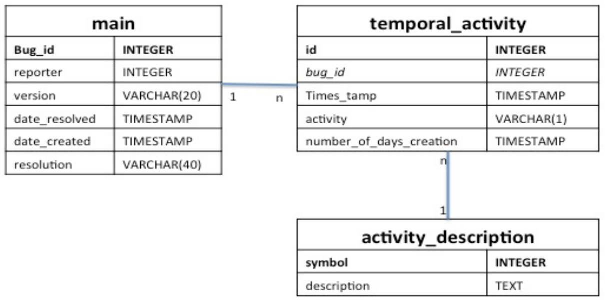
II. Description of the Dataset

The FTDD contains 86,444 bug reports covering an eight-year period from January 1, 2006, to December 31, 2014. Figure 1 presents the data schema (further explained in Section III). The dataset captures the key activities generated during the key internal processes over the lifetime of a bug until resolution is reached (resolved). Each activity has two attributes: 1) a symbol denoting a given activity; and 2) a date-time when the activity occurred.

Our criterion for selecting the key activities from the bug history logs was the level of involvement by developers and bug reporters. These activities are listed in Table I and are divided into three categories: involvement, communications, and bug condition.

A. Involvement

Involvement activities reflect the types of people involved in the lifecycle of the bug report. Based on this rationale,



main	
Bug_id	**INTEGER**
reporter	INTEGER
version	VARCHAR(20)
date_resolved	TIMESTAMP
date_created	TIMESTAMP
resolution	VARCHAR(40)

temporal_activity	
id	**INTEGER**
bug_id	*INTEGER*
Times_tamp	TIMESTAMP
activity	VARCHAR(1)
number_of_days_creation	TIMESTAMP

activity_description	
symbol	**INTEGER**
description	TEXT

Fig. 1. FTDD schema–primary keys are highlighted in bold and foreign keys are italicized

we focused on extracting certain key activities related to involvement, as described below.

Reporters: Several studies have indicated the importance of the reporter's experience. Hooimeijer et al. [2] called this variable "Reporter Reputation", while Zanetti et al. [10] confirmed this finding. We capture reporters experience by indicating their status as novice (N), intermediate (M) or experienced (E) at the time of report creation (for details, see the description of N, M, E in Table I).

Developers: Giger et al. [1] and Jeong et al. [5] used the developer assigned to a bug as a key feature in their predictive models. Once received, a bug report usually remains in an $unconfirmed$ state until an initial assessment (triage) effort confirms that it requires investigation, which changes its state to confirmed. Note that, if a bug is reported by a person with authority $can-confirm$, its starting state is immediately set to new (i.e., confirmed). Once confirmed, a bug is either assigned to a particular developer or made publicly available so that any developer can volunteer to fix it. We distinguish these two cases using the symbols A and R, respectively (see Table I for details). Once work starts, the bug's state is changed based on the Bugzilla lifecycle [11]; the remaining state values are $assigned$, $verified$, $reopen$, and $resolved$.

Experts involvement: Experts can be copied or referred to in order to approve development changes. We track these activities by mining and extracting information about notification activities and code reviews.

Carbon Copies (CC): Both Giger et al. [1] and Panjer et al. [7] used the CC count generated at various time intervals as a key feature in their prediction studies. Zanetti et al. [10] also employed CC activities to capture the social network interactions between developers and reporters. We denote any CC (i.e., notification activity) using the symbol C. If more than one C occurs in an hour, we label this case with the symbol D. Such a high frequency of notifications typically indicates important activities.

Code Review: Many researchers have extensively investigated code reviews. For example, using the Firefox dataset, Jeong et al. [5] suggested an algorithm for recommending suitable reviewers and attempted to predict levels of code acceptance. We extract information about code review requests and classify them as $normal$ and $super$ reviews denoted by V and S, respectively. Normal code reviews usually are assigned to the module owner, and super code reviews to a set of senior developers who oversee significant architectural refactoring and API changes (see [12] for details of the review process). These activities indicate that bug fixing has neared completion and that the bug owner awaits the reviewer's decision (accept or reject). We also capture responses to $normal$ and $super$ reviews, denoted by Y and H, respectively, which indicate that the waiting period is over. To the best of our knowledge, our dataset is the first to capture super review requests and responses.

B. *Communications*

Communication between developers indicates active efforts and progress toward resolving the bug. We capture two communication activities: 1) file attachments exchanged among developers, reviewers, or reporters, denoted by F; and 2) comments exchanged between developers, indicating a discussion or exchange of information[2], denoted by W.

C. *Bug Condition*

We capture key events throughout the bug's lifetime which affect its overall status and possibly indicate a change in the expected sizing or resolution type. We capture three activities:

1) Change in severity: Severity usually is set by the reporter and is changed infrequently by the triage or development team. Severity was changed for only 7% of bugs in the FTDD. Increased or decreased severity influences the overall standing of the bug report. We denote changes in severity by Q.

2) Change in priority: The triage or development team sets the priority. Again, the priority was changed for only 7% of bugs in FTDD in order to more clearly indicate the assessment effort being carried out and the condition of the bug. Although most studies have found that this field has low predictive power and effect, it might produce different results when put in the context of a temporal sequence. We denote a change in priority with the symbol P.

3) Resolution reached: This final activity in the sequence of activities is denoted by Z.

III. Methodology

Our methodology took place in three steps.

Step 1: Data Retrieval. First, we extracted from the Mozilla Firefox Defect dataset [9] bug ids covering a six-year period (2006 – 2012). The process was automated, with a script converting original XML data into CSV format. Second, we used the Bugzilla advanced search engine to extract bug and reporter ids covering a two-year period (2013 – 2014). We used the same selection criteria as in the original Mozilla Firefox Defect dataset [9], excluding any enhancements and selecting only bugs which had reached a resolution. We then joined both sets of bug ids and directly extracted their history logs from the Firefox Bugzilla repository. Sample history log

[2]These have been used in previous studies but as frequencies at certain time snapshot [1], [2], [7].

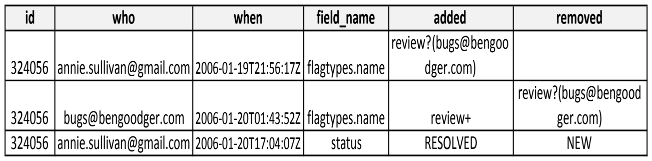
TABLE I
ACTIVITY SET

Activity	Symbol	*Description*
Reporting	N	Reporter has only reported one bug as of bug creation date.
	M	Reporter has reported more than one and less than ten bugs prior to bug creation date.
	E	Reporter has reported more than ten bug reports prior to bug creation date.
Assignment	A	Bug confirmed and assigned to a named developer.
	R	Bug confirmed and put to general mailbox for volunteers to work on.
Copy	C	A certain person has been copied on the bug report.
	D	More than one person has been copied within one hour on the bug report.
Review	V	Developer requested code review.
	Y	Response to code review.
	S	Developer requested super review.
	H	Response to super code review.
File Exchange	F	File exchanged between developers and reporters.
Comments exchange	W	Comment exchanged on whiteboard.
Milestone	L	A Milestone has been set for solution deployment.
Priority	P	Priority changed for bug report.
Severity	Q	Severity changed for bug report.
Resolution	Z	Bug reached status resolved.

is shown in Figure 2. The extraction was performed with a script using the repository's REST API.

Step 2: Mining & Extraction. We filtered and mined the retrieved history logs using another custom script which outputs the key activities (described in Section II) and their respective date-time of occurrence.

Step 3: Dataset Construction. The data extracted in Step 1 were used to build the schema's *main* table. The output of the script described Step 2 was stored in the schema's *temporal_activity* table. The reference table *activity_description* and descriptions of the activities and their symbols are given in Table I.



id	who	when	field_name	added	removed
324056	annie.sullivan@gmail.com	2006-01-19T21:56:17Z	flagtypes.name	review?(bugs@bengoodger.com)	
324056	bugs@bengoodger.com	2006-01-20T01:43:52Z	flagtypes.name	review+	review?(bugs@bengoodger.com)
324056	annie.sullivan@gmail.com	2006-01-20T17:04:07Z	status	RESOLVED	NEW

Fig. 2. Sample history log.

IV. DATASET ANALYSIS

This section presents a detailed description of the temporal activity sequences generated in our dataset. First, we summarize the reporter experience. In Figure 3, we plot the reporter experience versus the final *fixed* resolution to show that a reporter's experience increases the odds of a bug reaching a final *fixed*. For example, only 13.74% of bug reports reported by $novices$ reached a final resolution of *fixed*, whilst the rest are resolved and closed as one of the following : *duplicate, invalid ,worksforme, incomplete or wontfix*.

Second, we discuss the length of activity sequences and the duration of bug fixes, as shown in Table II. On average, each bug undergoes a sequence of ≈ 5 key activities. In Table II, we can also observe that the average number of days until the final resolution is achieved is 63, and that 50% of the defects (median) are closed in less than one day (0 days).

Third, we investigate the temporal relationship between activities by applying an n-gram pattern extraction algorithm, which extracts patterns of a given length from a sequence of activities by sliding a window one activity at a time. For example, if $DFRD$ represents a sequence of four activities, then three patterns of length 2 (namely, DF, FR, and RD) can be extracted from this sequence. Figure 4 illustrates the most frequent 2, 3, and 5 grams in our dataset.The frequent n-grams help us identifying common patterns in the dataset. From 3-grams, we observe that 959 bugs have the temporal activity sequence NCZ, which represents a full bug lifecycle: reported by novice N, carbon copy C issued and finally resolved by Z. Similarly, in 5-gram patterns, 143 bugs share the temporal pattern of $NCWCZ$. In all, 1,863 bugs across all three types of n-grams (2-grams, 3-grams, 5-grams) share one of the following sequences: NZ, NCZ, MCZ, NDZ, $NCWCZ$, $NCCCZ$, $NWCCZ$ for its full bug lifecycle. Of these bugs, 87% have a $novice$ reporter, which implies that bugs reported by more experienced reporters have longer sequences. In an interesting pattern, VVV occurs 112 times among the 3-grams, indicating that developers request code reviews but receive no response from reviewers.

Fourth, we examine activities frequency of occurrence by bug state (i.e., status detailed in Section II-A), which reflects a temporal stage in the bug lifecycle. Table III compares the frequencies and states of our key extracted activities. We note first that changing bug severity Q happens mostly in the initial states of *unconfirmed* (66.4%) or *new* (18.01%) which indicate an assessment stage. A similar observation can be made for changes in priority P. We also note that making the bug available for volunteers R happens most frequently when bug is in the *new* state (57.59%), consistent with the Bugzilla lifecycle model. We also see that approximately 50% of code reviews V, Y and super code reviews S, H happen in the *assigned* state which usually has a named developer. Analysis of milestone-setting activity L shows that 47.35% of the activity occurs during the final *resolved* state, likely as a wrap-up and closure activity; and 30.92% during the *new* state, indicating either upfront knowledge of the solution or planning efforts.

V. RELEVANCE OF THE DATASET

Based on the findings presented in Section IV, we believe that researchers in temporal analysis and pattern recognition can use our dataset to predict defect proneness or defect sizing and suggest potential defect fixes. The FTDD can easily be extended with additional attributes. For example, to add a text

TABLE III
DISTRIBUTION OF ACTIVITIES AT VARIOUS DEFECT STATES. ACTIVITY SYMBOLS ARE DESCRIBED IN TABLE I.

State	*A*	*R*	*C*	*D*	*F*	*W*	*Q*	*L*	*P*	*S*	*H*	*V*	*Y*
unconfirmed	3.24%	5.19%	38.56%	31.91%	24.89%	46.14%	66.40%	1.71%	19.61%	2.93%	0.00%	1.83%	1.16%
new	53.39%	57.59%	29.78%	41.33%	24.99%	20.59%	18.01%	30.92%	54.06%	41.39%	46.94%	42.05%	38.13%
assigned	35.14%	8.02%	8.04%	4.82%	36.81%	16.64%	4.06%	15.57%	17.42%	50.55%	45.19%	50.41%	53.79%
resolved	6.25%	22.93%	17.39%	17.34%	8.06%	13.48%	9.69%	47.35%	6.44%	2.93%	4.96%	2.31%	2.94%
verified	0.97%	3.91%	4.96%	4.38%	0.72%	1.59%	1.16%	2.77%	0.90%	0.00%	0.29%	0.70%	0.61%
reopened	1.01%	2.36%	1.29%	0.22%	4.53%	1.55%	0.67%	1.69%	1.57%	2.20%	2.62%	2.70%	3.37%

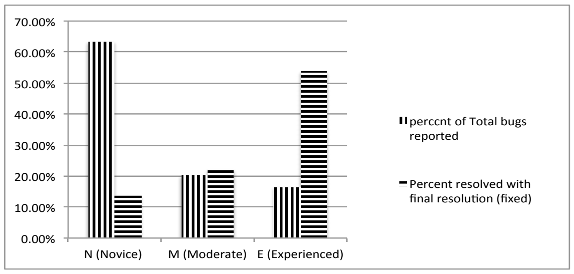


Fig. 3. Reporter Experience at time of reporting versus final resolution

TABLE II
SUMMARY STATISTICS FOR LENGTH OF ACTIVITY SEQUENCES & DURATION OF BUG FIX

	Min	1st Qu.	Median	Mean	3rd Qu.	Max
Length of activity seq.	2	3	4	5.404	6	193
Duration of bug fix (in days)	0	0	0	63.06	7	2290

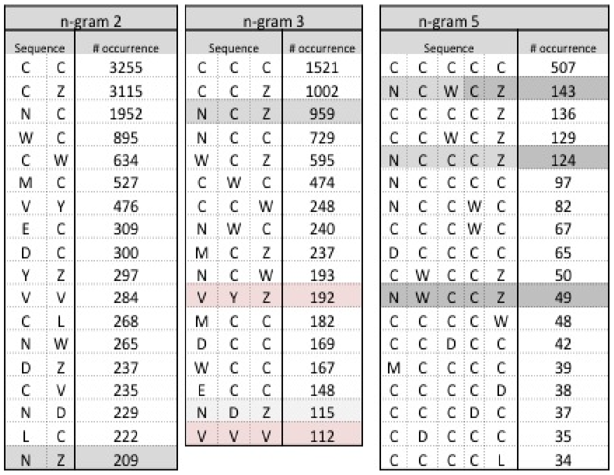
description field to a bug report (say, to perform text mining), a researcher can append the field to the FTDD by joining new data via the bug id. Data are provided in MySQL and CSV formats, allowing researchers to ingest the dataset with minimal effort and avoid costly extraction and transformation processes.

VI. CHALLENGES AND LIMITATIONS

When retrieving the histories of each bug report, we noted that some defects reported as resolved in the original dataset [9] were reopened after the dataset was published. We addressed this issue by incorporating these changes into our dataset. We depended on the bug repository data to retrieve and build a full picture of the bug lifecycle. However, the bug repository does not capture all software engineering activities and communications, an issue common in bug tracking repositories.

VII. SUMMARY

In this paper, we present a Firefox Temporal Defect Dataset (FTDD) and the rationale for selecting the key activities which it captures. We also describe the FTDD schema and data sources. We present an exploratory analysis on the dataset that we mined. We believe the analysis provides insights to researchers and practitioners to use FTDD in diagnosing and fixing problems in bug-fixing processes.



n-gram 2

Sequence		# occurrence
C	C	3255
C	Z	3115
N	C	1952
W	C	895
C	W	634
M	C	527
V	Y	476
E	C	309
D	C	300
Y	Z	297
V	V	284
C	L	268
N	W	265
D	Z	237
C	V	235
N	D	229
L	C	222
N	Z	209

n-gram 3

Sequence			# occurrence
C	C	C	1521
C	C	Z	1002
N	C	Z	959
N	C	C	729
W	C	Z	595
C	W	C	474
C	C	W	248
N	W	C	240
M	C	Z	237
N	C	W	193
V	Y	Z	192
M	C	C	182
D	C	C	169
W	C	C	167
E	C	C	148
N	D	Z	115
V	V	V	112

n-gram 5

Sequence					# occurrence
C	C	C	C	C	507
N	C	W	C	Z	143
C	C	C	C	Z	136
C	C	W	C	Z	129
N	C	C	C	Z	124
N	C	C	C	C	97
N	C	C	W	C	82
C	C	C	W	C	67
D	C	C	C	C	65
C	W	C	C	Z	50
N	W	C	C	Z	49
C	C	C	C	W	48
C	C	D	C	C	42
M	C	C	C	C	39
C	C	C	C	D	38
C	C	C	D	C	37
C	D	C	C	C	35
C	C	C	C	L	34

Fig. 4. Top 2-gram, 3-gram, 5-gram of FTDD activities.

ACKNOWLEDGMENT

This research is supported in part by NSERC DG no. 402003-2012.

REFERENCES

[1] E. Giger, M. Pinzger, and H. Gall, "Predicting the fix time of bugs," in *Proc. of the 2nd Int. Wkshp. on Rec. Sys. for Softw. Eng.*, 2010, pp. 52–56.

[2] P. Hooimeijer and W. Weimer, "Modeling bug report quality," in *Proc. of the 22nd Int. Conf. on Automated Softw. Eng.*, 2007, pp. 34–43.

[3] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in *Proc. of the 28th Int. Conf. on Softw. Eng.*, 2006, pp. 361–370.

[4] N. Jalbert and W. Weimer, "Automated duplicate detection for bug tracking systems," in *Proc. of Int. Conf. on Dependable Systems and Networks With FTCS and DCC*, 2008, pp. 52–61.

[5] G. Jeong, S. Kim, T. Zimmermann, and K. Yi, "Improving code review by predicting reviewers and acceptance of patches," *Research on Software Analysis for Error-free Computing Center Tech-Memo (ROSAEC MEMO 2009-006)*, pp. 1–18, 2009.

[6] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller, "How long will it take to fix this bug?" in *Proc. of the 4th Int. Workshop on Mining Softw. Repos.*, 2007, pp. 1–8.

[7] L. D. Panjer, "Predicting eclipse bug lifetimes," in *Proc. of the 4th Int. Workshop on Mining Softw. Repos.*, 2007, pp. 29–32.

[8] "Mozilla," accessed: 2015-02-14. [Online]. Available: http://blog.mozilla.org/press/ataglance/

[9] A. Lamkanfi, J. Pérez, and S. Demeyer, "The eclipse and mozilla defect tracking dataset: a genuine dataset for mining bug information," in *Proc. of the 10th Working Conf. on Mining Softw. Repos.*, 2013, pp. 203–206.

[10] M. S. Zanetti, I. Scholtes, C. J. Tessone, and F. Schweitzer, "Categorizing bugs with social networks," in *Proc. of the 2013 Int. Conf. on Softw. Eng.*, 2013, pp. 1032–1041.

[11] "Life cycle of a bug," accessed: 2015-02-14. [Online]. Available: https://www.bugzilla.org/docs/2.18/html/lifecycle.html

[12] "Super-review policy," accessed: 2015-02-14. [Online]. Available: https://www.mozilla.org/hacking/reviewers.html