BUMPER: A Bug-Fix Search Engine

Mathieu Nayrolles,
Software Behaviour Analysis (SBA) Research Lab
ECE, Concordia University
Montreal, Canada
m_nayrol@ece.concordia.ca

Abdelwahab Hamou-Lhadj
Software Behaviour Analysis (SBA) Research Lab
ECE, Concordia University
Montreal, Canada
abdelw@ece.concordia.ca

***Abstract*—In the last decade, the use of bug tracking tools have been adopted by both open-source and industry to take advantage of their various features. Bug tracking systems such as Bugzilla or Jira have grown to contain hundreds of thousands of bugs and provide crucial data to several active research fields such as bug reproduction, bug triaging, identification of duplicate bug, bug comprehension, and empirical studies. Easy access to the combined histories of all major open-source projects will save developers time and efforts when facing a bug or crash. The fixes to these bugs, however, are contained in software versioning and revision systems and not in the bug tracking system. Furthermore, each software project comes with its own bug report and source versioning system. As a result, developers tend to reinvent the wheel instead of leveraging known solutions to similar problems. In this paper, we present an online tool called BUMPER (BUg Metarepository for dEvelopers and Researchers) that aims to ease this process and allows natural language searches in bug reports, commit messages and source code all together at https://bumper-app.com/. Currently, BUMPER contains more than 380 projects, 100,000 resolved/fixed bugs and almost 70,000 changesets from Netbeans and the Apache Software foundations softwares. BUMPER is designed to be easily extensible to contain bugs and fixes from other bug and version control systems.**

I. Introduction

Debugging programs, during their creations or at maintenance time is challenging [1] and taxing – Studies have shown that the cost of software maintenance can reach up to 70% of the overall cost of the software development process.

When facing a yet unknown bug, one might want to leverage decades of open-source software history in order to find a solution. Indeed, chances are a similar bug or crash has already been fixed somewhere in an open-source project.

However, each open-source project host its data on a different data repository and using different bug tracking and source code management systems. Moreover, these systems have different interfaces to access data. The data is not represented in a uniform way either. This is further complicated by the fact that bug tracking tools and code versioning systems are not necessarily connected. The former follows the life of the bug itself and not its fixes, which are managed by the latter. As a general practice, developers create a link between the bug report system and the source versioning tool by either writing the bug ID in their commit message or add a link towards the changeset as a comment in the bug report system.

As a result, one would have to search the source code versioning system repository to find candidate solutions. Moreover, developers mainly use classical search engines that index specialized sites such as StackOverflow.

However, specialized sites are organized in the form of question-response where one explain his problem and gets answers from the community. While the answers are often accurate and precise, they do not leverage the history of open-source software that has been shown to provide useful insight to help with many maintenance activities such as bug fixing [2] [3] , bug reproduction [4] [5] [6], fault analysis [7], etc. For the present time, analyzing the bugs and their fixes from different sources require going back and forth between diverse tools, creating parsers, mapping data from a repository to another, etc. These tasks are not only time consuming but add no value to the analysis itself.

In this paper, we introduce BUMPER (BUg Metarepository for dEvelopers and Researchers), a web-based infrastructure that can be used by software developers and researchers to access data from diverse repositories using natural language query in a transparent manner, regardless of where the data was originally created and hosted.

The idea behind BUMPER is that it can connect to any bug tracking and source code versioning system and download the data into a single database. BUMPER uses a web-based interface that allows users to search the aggregated database by expressing queries through a single access point. This way, users can focus on the analysis itself and not on the way data is represented or where it is located.

To create an approach that will efficiently help the community at debugging time we first ask computer science students, professionals and practitioners to answer a short survey about how they face unknown bug, crash or exception.

Even thought their answers, presented in Section II were not surprising, we still used them to build an approach that fits the needs of the community and built a dataset of 840 open source projects belonging to five major open-source institutions with more than 100,000 resolved/fixed and with 60,000 changesets that were involved in fixing them from Eclipse, Netbeans, Gnome, Github and the Apache Software foundation's software.

BUMPER supports many features including: (1) the ability to search large data repositories very efficiently using both natural languages and a specialized query language, (2) the mapping between the bug reports and the code involved to fix it, (3) the ability to export the search results in Json, CSV and

XML formats.

Finally, we went back to the community and ask them to solve bugs and crashes using their habitual habits and using BUMPER. The results indicate that our approach reduces the amount of data-sources to be visited in order to find a suitable solution for the bug at hand and therefore, speeding up the debugging and maintenances processes.

The remaining of this paper is organized as follows. First we present the results of our first survey aiming to understand the behavior of developers when facing a bug in Section II. Then, we present background informations about bug tracking and code versions systems in Section III. BUMPER itself is presented in terms of its architecture, metadata and API in section IV. Our second survey, presenting the advantages of BUMPER compared to habitual ways of doing maintenance and bug fixing is the subject of Section V. Finally, we present related works and concluding remarks in Sections VII and VIII, respectively.

II. Tools and methods used to fix unknown bug

In this section we present the results of a two-rounds survey we conduct with students and developers in order to understand what are their habits when dealing with an yet unknown bug, exception or crash. In the first round, we ask participant to answer the following two question: (1) *When facing an unknown bug, crash or exception, where do you look for informations ?* and (2) *When facing an unknown bug, crash or exception, what are you searching for ?*. We then summarized their answers and took the most popular ones to build the following multi-choices survey:

1) When facing an unknown bug, crash or exception, where do you look for informations ?
   a) Online search engines (Google, Bing, ...).
   b) Online specialized websites (Stackoverflow, ...)
   c) Offline search (Books, Slides, ...).
   d) Official documentation of the project / programming languages / api (Offline and Online).
   e) Other.
2) When facing an unknown bug, crash or exception, what are you searching for ?
   a) How to do X.
   b) The exception yield by the crash / bug.
   c) Official documentation of the project / programming languages / api (Offline and Online).

We submitted this survey to a total of 150 peoples and got 89 responses (59%). Respondent to our surveys have been classified into four distinct groups: undergraduate (42) and graduate (25) students in computer sciences, electrical and computer engineering and software engineering. The two last groups are junior developers (15) (0-3 years of experience) and intermediate or senior developers (7) (3+ years of experience). We report the results of our surveys using descriptive statistics (percentage) and present the results for each of our four surveyed groups.

Figure 1 presents the results of our first question : *When facing an unknown bug, crash or exception, where do you look for informations ?* 73%, 65%, 75% and 58% of Undergraduate, Graduate, Junior developers and intermediate or senior developers, respectively, indicate that they use *Online search engines* such as Google or Bing to resolve yet unknown bug. In the meantime, 5%, 25%, 12% and 28% of Undergraduate, Graduate, Junior developers and intermediate or senior developers, respectively, indicate that they use *Online specialized websites* such as StackOverflow. Interestingly, some of the respondent that were surveyed offline and answered that they use *Online search engines* search engines told us that the best answer was often hosted on a *Online specialized websites* such as StackOverflow. *Offline* (Books, slides, ...) and *documentation* searches reach 22% and 0%, 5% and 2%, 5% and 8% and 8% and 12% for Undergraduate, Graduate, Junior developers and intermediate or senior developers, respectively.

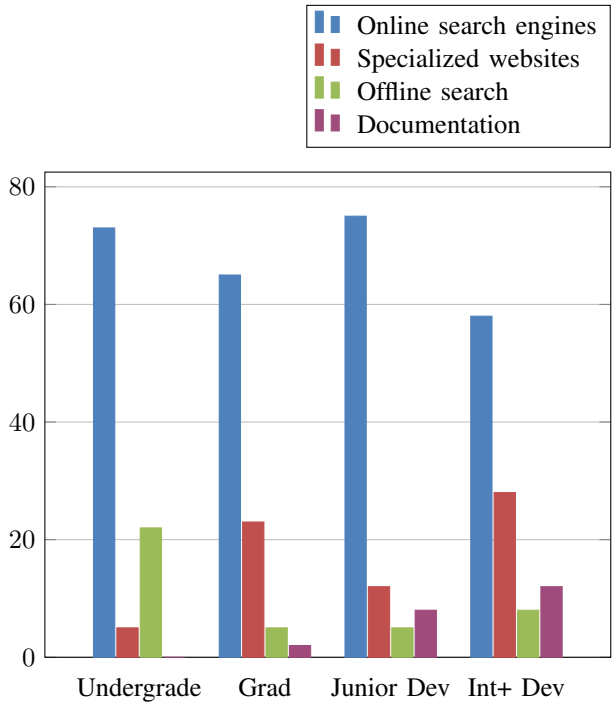

Fig. 1. Answers to *When facing an unknown bug, crash or exception, where do you look for informations ?* in percentage

To summarize, 84.7% of respondents declared that their first reflex when facing an yet unknown bug was to search for a suitable solution online. These 84.7% participant were asked to continue the survey with the second question. To keep our results as qualitative as possible we excluded participants that indicated searching for resources offline as our approach aimed to improve onlime behaviors.

Figure 2 presents the results of our first question : *When facing an unknown bug, crash or exception, what are you searching for ?* 47%, 32%, 55% and 22% of Undergraduate, Graduate, junior developers and intermediate or senior developers, respectively, indicate that they use a *How to do X*

search query when searching solution to resolve yet unknown bug. Another 53%, 60%, 37 % and 67% of Undergraduate, Graduate, junior developers and intermediate or senior developers, respectively, copy past directly the exception or crash output at hand. Finally, 0%, 8%, 8% and 11% of Undergraduate, Graduate, junior developers and intermediate or senior developers, respectively search for solution in the online official documentation of the project they are working on.

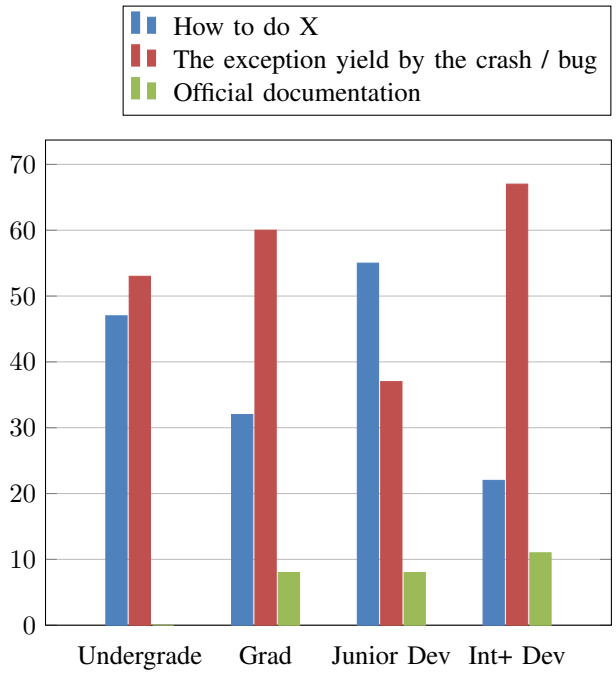

Fig. 2. Answers to *When facing an unknown bug, crash or exception, what are you searching for ?* in percentage

To summarize, 93.25 % of surveyed participant who search solution to an unknown bug or crash online use query composed of *How to do X* or directly copy past their error / exception. Both these information can be found in bug report and bug fixes as described in the next section.

III. BACKGROUND

A. Bug Tracking System

Open source bug tracking systems allow end-users to directly create BRs to report on system crashes. These systems are also used by development teams to manage the BRs, and keep track of the fixes. The lifecycle of a bug in both systems is as follows: After a bug is submitted by an end-user, it is set to the UNCONFIRMED state until it receives enough votes or that a user with the proper permissions modifies its status to EW. The bug is then assigned to a developer to fix it. When the bug is in the ASSIGNED state, developers start working on fixing the bug. A fixed bug moves to the RESOLVED state. Developers have five different possibilities to resolve a bug: FIXED, DUPLICATE, WONTFIX, WORKSFORME and INVALID. Finally, the bug is closed after it is resolved. A bug can be reopened (set to the REOPENED state) and then assigned again if the initial fix was not adequate (the fix did not resolve the problem). The elapsed time between the bug is marked as new one and the resolved status is known as the fixing time, we denote the fixing time by BFT. It usually in days. If the bug is reopened then the days between the time the bug is reopened and the time it is marked as RESOLVED/FIXED are cumulated. Bugs can be reopened many times. The relationship between a BR and the actual fix can be hard to establish and it has been a subject for various research studies (e.g., [8] [9]). While it is considered a good practice to link each BR with the source code repository by indicating the bug id on the commit message, more than half of the bugs in our dataset are not linked to a commit. We exclude these bugs in this study and only consider the ones that have a commit. This way, we can establish a link between the bug and its fixes. A commit contains the number of files changed in all the commits linked to the bug, the number of hunks (the number of lines in a changeset the commits linked to a bug), the number of lines modified in all the commits. We use this data to construct BUMPER metamodel.

B. Version Constrol System

Version control systems are used to maintain the versions of files such as source code and other software artifacts. Many software tools have been created to help practitioners manage the versions of their software artifacts. Each evolution of a software is a version (or revision) and each version (revision) is linked to the one before through modifications of software artifacts. These modifications consist of updating, adding or deleting software artifacts can be referred as diff, patch or commit. Each diff, patch or commit own the following characteristics:

- Number of Files: The number of software artifacts that have been modified, added or deleted.
- Number of Hunks: The number of consecutive block of modified, added or deleted lines in textual files.
- Hunks are useful to determine, in each file, how many different places the developer has modified.
- Number of Churns: The number of lines modified. However, the churn value for a line change should be at least two as the line had to be deleted first and then added back with the modification

IV. BUMPER COMPONENTS

Figure 3 shows BUMPERs main interface. BUMPER aggreagates data from various bug tracking and version control systems that can be efficiently accessible using a webbased interface. Currently, BUMPER supports 380 projects, more than 100,000 resolved/fixed and with 60,000 changesets from Netbeans and the Apache Software foundations softwares. It can readily be extended to support other systems. Using BUMPER, software engineers can query multiple repositories and save the data in various formats including Json, CSV, and XML for further analysis.

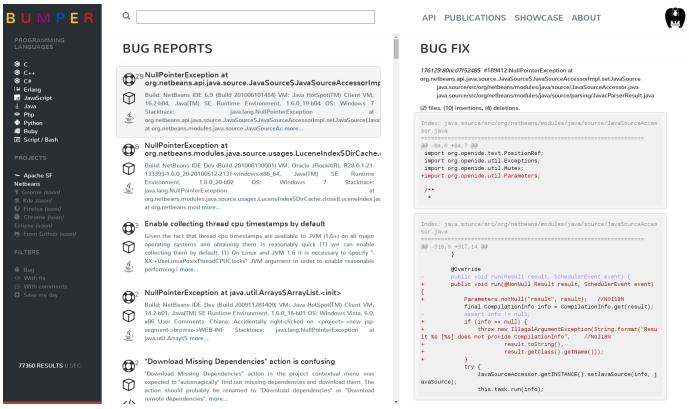

Fig. 3. BUMPER Snapshot.

A. BUMPER Architecture

Figure 4 shows the overall architecture of BUMPER. BUMPER relies on a highly scalable architecture composed of two Virtual Private Servers (VPS), hosted on a physical server. The first server, on the left, handles the web requests and runs three distinct components: Pound, Varnish, and NginX. Pound is a lightweight open source reverse proxy program and application firewall. It also serves to decode https to request to http. Translating an https request to http allows us to save the https decryption time required on each step. Pound also acts as a load-balancing service for the lower levels. The translated requests are then handled by Varnish. Varnish is an http accelerator designed for content-heavy and dynamic websites. It caches requests that come in and serves the Web requests from the cache is the cache is still valid. NginX (pronounced engine-x) is a web-server that was developed with a particular focus on high concurrency, high performances, and low memory usage. The second VPS also uses Pound for the same reasons and SolrCloud. SolrCloud is the scalable version of Apache Solr where the data can be separated into shards (e.g chunk of manageable size). Each shard can be hosted on a different server but still indexed in a central repository. Hence, we can guarantee a low query time while exponentially increasing the data. Finally, Lucene is the full text search engine powering Solr.

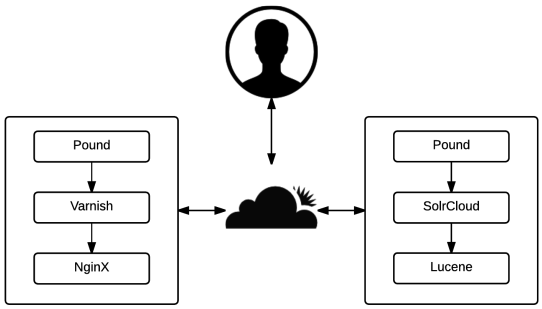

Fig. 4. BUMPER Architecture.

B. BUMPER Metadata

Figure 5 shows the core BUMPER metamodel, which captures the common dat elements used by bug tracking and control version systems. An issue (task) is characterized by a date, title, description, and a fixing time.

Issues are reported (created) by and assigned to users. Also, issues belong to a project that is in a repository and might be composed of subprojects. Users can modify an issue during life cycle events which impact the type, the resolution, the platform, the OS and the status. Issues are resolved (implemented) by changeset that are composed of hunks. Hunks contain the actual changes to a file at a given revision, which are versions of the file entity that belongs to a project. In addition, BUMPER has different indexes over the data in order to provide results in an efficient way. First of all, each feature of bug reports and bug fixes (number of files, number of hunks, etc.) have their own index. Furthermore, BUMPER has data indexes over the project name, the sub-project name, the programming language, etc. Finally, two major indexes are built upon the concatenations of all the textual fields of the bug report and all the textual field (source code included) of a bug fix. They are named report_t and fix_t, respectively. These indexes are used when querying bug reports or bug fixes using natural language. Indexes can also be found in relational database management system (RDBMS). Most modern RDBMS use binary trees to store indexes that keep data sorted and allow searches, insertions, and deletion in a logarithmic time. Nevertheless, an RDBMS can use only one index per table at the same time. This said, an RDBMS chooses only one index within the available ones–based on statistics–and uses it to complete the request. It is highly probable that thousands of records have to be scanned before the RDBMS finds the ones that match the query, especially if there is union or disjunction of records. Unlike a traditional RDBMS, BUMPER relies on Apache Lucene and uses compressed bitsets to store indexes. Bitsets are one of the simplest–and older–data structure that contain only 0 and 1. BUMPER supports binary operations like intersection, AND, OR and XOR that can be performed in a snap on even for thousands and thousands of records. As an example, if we wish to retrieve bug reports that contain the words null pointer exception and have a changeset containing a try/catch, a binary intersection will be performed between the two sets of documents, which is much faster than selecting bug reports that match null pointer exception first and then checking if they have a changeset containing a a try/catch as in the case of an RDBMS. This technique comes with a high overheadcomparing to an RDBMS–for indexes update, but, in practice, information retrievals are orders of magnitude faster. In our case, we want to provide a fast access to decades of open source history. We periodically update (at night) our indexes when sufficient amount of new data have been downloaded from the bug tracking and source versioning systems BUMPER supports

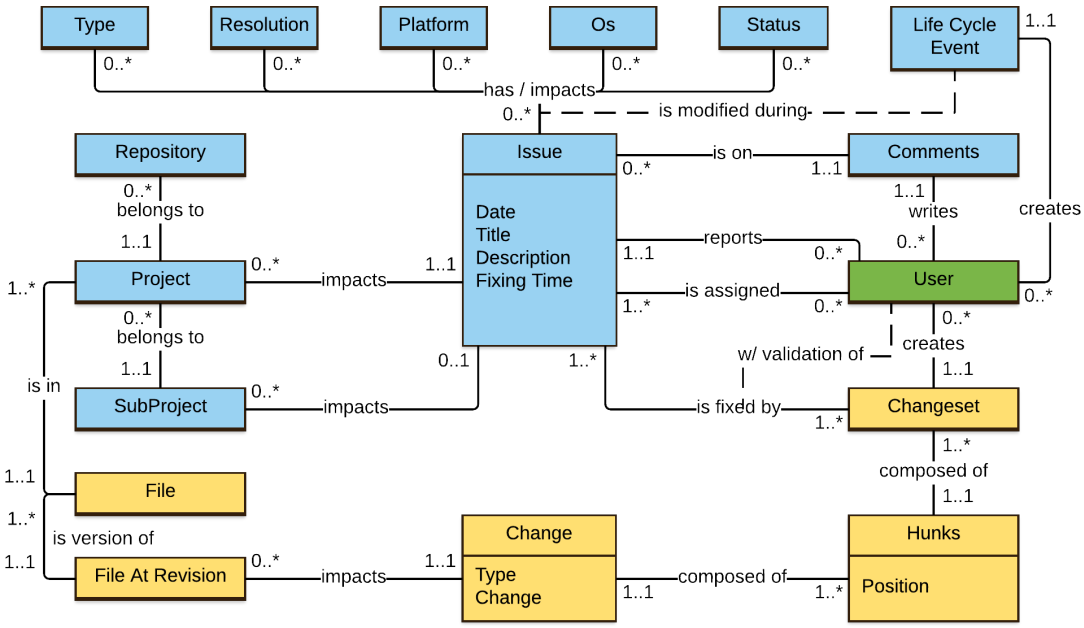

Fig. 5. BUMPER Metamodel

C. Bumper Query Language and API

BUMPER supports two query modes: basic and advanced. The basic query insert users' inputs (YOUR TERMS) in the following query:

$$(type:"BUG"\ AND\ report_t:"YOUR\ TERMS"\ AND\ -churns:0) \quad (1)$$

The first part of the query matches all bug reports (type:"BUG") that contain YOUR TERMS in the report_t index (report_t:"YOUR TERMS"). Finally, it excludes bug reports that do not have a fix (-churns:0). The result of this subquery will be merged with the following:

$$OR\ (\{!parent\ which="type:BUG"\}type:"CHANGESET"\ AND\ fix_t:"YOUR\ TERMS" \quad (2)$$

This query selects all bugs' changeset–using a parent-child relationship ($\{!parent which = "type : BUG"\}$ $type : "CHANGESET"$) – and that contain YOUR TERMS in the fix_t index (fix_t:"YOUR TERMS"). Finally, the results of the two previous queries will be added to the following subquery:

$$OR\ (\{!parent\ which="type:BUG"\}\{!parent\ which="type:CHANGESET"\}\ type:"HUNKS"\ AND\ fix_t:"YOURTERMS") \quad (3)$$

The query selects all the hunks that are child of changsets and grand-child of bug report ($\{!parent\ which = "type : BUG"\}\{!parent\ which = "type : CHANGESET"\}$ $type : "HUNKS"$) and that contain YOUR TERMS in the fix_t index. The intent of this composed query is search efficiently for YOUR TERMS in bug reports, commit messages and source code all together. The advanced query mode allows users to write their own query using the indexes they want and the unions or disjunctions they need. As an example, using the advanced query mode, one could write the following:

$$(type:"BUG"\ AND\ report_t:"Exception" \\ AND\ (project:"Axis2"\ OR\ project:"ide") \\ AND\ (reporter:"Rich"\ OR\ resolution:"fixed") \\ AND\ (severity:"Major"\ OR\ fixing_time:[10\ TO\ *]) \\ AND\ -churns:0) \quad (4)$$

This query finds all bug reports that contain Exception in the report_t index (first line) and belong to the Axis2 or the ide project (line 2) and have been reported by someone named Rich or have been fixed as a resolution (third line) and that have a Major severity or a fixing time greater than 10 days (fourth line) and have a fix (fifth line). More examples of the APIs usage can be found at https://bumper-app.com/api.

D. Bumper Data Repository

Figure 6 illustrates the process of collecting data from various repositories and use them in BUMPER. We show the process using the repositories currently supported by BUMPER.

First, we extract the raw data from bug report management systems of five different open source institutions used

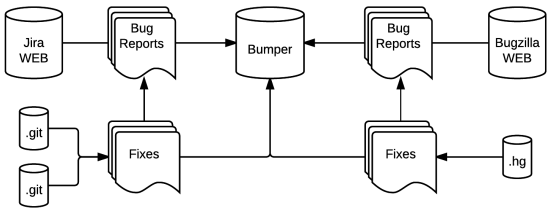

Fig. 6. Overview of the bumper database construction.

TABLE I
RESOLVED/FIXED BUG (R/F BR), CHANGSETS (CS), AND PROJECTS BY DATASET

Dataset	R/F BR	CS	Files	Projects
Gnome				10
Netbeans	53,258	122,632	30,595	39
Apache	49,449	106,366	38,111	349
Eclipse				192
Github				250
Total				840

in this study: Gnome, Eclipse, Netbeans, Apache Software Foundation and Github. Each institution is composed of many open source projects: 10, 192, 39, 349 and 250, respectively. Summing up to 840 different open source projects.

The extracted data is consolidated in one database where we associate each bug report with its fix. The fixes are mined from different types of source versioning system. Indeed, Gnome, Eclipse and Apache Software Foundation projects are based on Git (or have git based mirrors) while Netbeans uses Mercurial.

Gnome is a free desktop environment mainly developed in c and c++ by volunteers and paid contributors both.

Eclipse and Netbeans are integrated development environments (IDEs) for developing with many programming languages including Java, PHP, and C/C++.

The Apache Software Foundation (ASF) is a non-profit public charity established in 1999, that provides services and support for many like-minded software project communities of individuals who choose to join the ASF.

Finally, Github is a free web-based Git repository hosting service that allow anyone to create, version and maintain a team development project while using Git as source code management and Github issues as Bug management system.

The characteristics of the five dataset used in this paper are presented in Table I. Cumulatively, these datasets span from 2001 to 2014.

In summary, our consolidated dataset contains 102,707 bugs, 229,153 changesets, 68,809 files that have been modified to fix the bugs, 462,848 comments, and 388 distinct software tools. We also collected 221 million lines of code impacted by the changesets, identified 3,284 sub-projects, and 17,984 unique contributors to these bug report systems.

We choose these five datasets because they exposed a great diversity in programming languages, teams, localization, utility and maturity. Moreover, the used different tools, i.e. Bugzilla, JIRA, Git and Mercurial, and therefore, BUMPER is ready to host any other datasets that used any composition of these tools.

BUMPER users can follow the same process to add other repositories from various bug tracking and control version systems such as Eclipse, Mozilla Foundation datasets. The BUMPER architecture can then be used to organize and access multiple repositories in a unified manner.

V. Experiments

In order to assess the efficiency of BUMPER during the debugging / maintenance processes we conduct in which we present participants of the surveys presented in Section II with a bug to fix. Then, we ask them to find a suitable solution of the presented bug online as would have in a classical debugging / maintenance session and repeat the operation using BUMPER. Finally, participants were asked to report how long it took to find a suitable solution with both methods and to rate their experience using BUMPER.

In the remaining of this section we present the three different bugs we randomly submitted to our participant in Section V-A. Then, we describe the time to find a solution reported by our participants and present some of their comments in Sections V-B and V-C, respectively.

A. Bugs to fix

B. Time to find a solution

C. Users comments

VI. Research Oportunities

In this section we present several research opportunities that BUMPER supports in addition to enhance the debugging / maintenance processes.

A. ???

B. ???

C. Studying bug-fix relationship

Studying software repositories to gain insight into the quality of the code is a common practice. However, this task requires time and skills in order to download and link all the pieces of informations needed for adequate mining. BUMPER provides a straightforward interface to export bugs and their fixes into CSV, XML and JSON. As an example, if one wants to extract the features of a bug report that get fixed quickly (this type of analysis is performed in [10]), the following query can be used:

$$\begin{aligned}(type:"BUG"\ AND\ fixing_time:[0 TO 10]\ churns:0)\\ OR\ (type:"BUG"\ AND\ fixing\_time:[200\ TO\ *]\\ churns:0)\&facet=true\&facet.field=project\end{aligned} \quad (5)$$

The query returns bugs that have been fixed in less than 10 days and bugs that have been fixed in more than 200 days. Moreover, the $facet = true facet.field = project$ part of the query provides a count of each category (e.g., less than 10 days and more than 200 days) by projects. Finally, researchers

can press the XML, CSV or JSON button, shown in Figure 1. to export their data in the desired format. In short, BUMPER offers a framework that can be used by software practitioners and researchers to analyze (efficiently) bugs and their fixes without having to go from one repository to another, worry about the way data is represented and saved, or create tools for parsing and retrieving various data attributes. We hope that the community contributes by adding more repositories to BUMPER. This way, BUMPER can become a unified environment that can facilitate bug analysis and mining tasks.

VII. Related Works

In last decade, studying the software history in order to improve software maintenance tasks has become popular among researchers and practitioners. Researchers have specified what makes a good bug report [10], studied who should and how long it will take to fix a given bug [12][13]. In addition, researchers have investigated how to resolve the discrepancies between the source code revision system and the bug tracking system [8] [9] [11] [12] and how to extract and generate fix patterns [13] [14]. Panoply of approaches and tools built upon this data have also been assessed [15], [16]. In parallel tools and algorithms to predict or prevent bug insertions that rely on pre-defined bug pattern, theorem proving, model checking or past history have been created [17] [18] [19] [20].

However, to the best of our knowledge, no attempt has been made towards building an unified and online datasets, from multiple sources, where all the information related to a bug, or a fix can be easily accessed by researchers and engineers in order to leverage decades of historical information. Providing a unified, online, and easily accessible datasets has been shown to be effective for learning new APIs [21] [22] [23] and BUMPER aims to achieve a similar goal, which is empowering developers by giving access to multiple data repositories that they can use to find how similar bugs have been fixed throughout the history of hundreds of wellknown software projects.

VIII. Conclusion

In this paper, we presented an online tool named BUMPER (BUg Metarepository for dEvelopers and Researchers) accessible at https://bumper-app.com. BUMPER allows natural language searches in bugs reports, commit messages and source code all together while supporting complex queries and contain 380 projects, more than 100,000 resolved/fixed and with 60,000 changesets that were involved in fixing them from Netbeans and The Apache Software foundations software.

The speed of BUMPER allows developers to use it as a way to leverage decades of history scattered over hundreds of software project in order to find existing solutions to their problems. As future work, we want to improve BUMPER by adding more projects such as Eclipse, Mozilla Foundation datasets. Also, we intend to add other features to our bugs report such as the amount of reopening or the number of time a bug has been duplicated.

Acknowledgment

The authors would like to thank Ecole de Technologie Superieur Montreal's, Concordia University's, UQAM's and eXia's computer science students and Ubisoft's, Ericsson's, Morgan Stanley's, OVH's, 60fps' and Borealys' developers for participating in our surveys and helping us understanding their day to day needs.

References

[1] R. S. Pressman, *Software Engineering: A Practitioner's Approach*. Palgrave Macmillan, 2005. [Online]. Available: https://books.google.com/books?hl=en&lr=&id=bL7QZHtWvaUC&pgis=1

[2] C. Weiß, T. Zimmermann, and A. Zeller, "How Long will it Take to Fix This Bug ?" in *Fourth International Workshop on Mining Software Repositories (MSR'07)*, no. 2, 2007, p. 1.

[3] R. K. Saha, S. Khurshid, and D. E. Perry, "An empirical study of long lived bugs," in *2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*. IEEE, feb 2014, pp. 144–153. [Online]. Available: http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=6747164

[4] N. Chen, "Star: stack trace based automatic crash reproduction," Ph.D. dissertation, 2013.

[5] S. Artzi, S. Kim, and M. D. Ernst, "Recrash: Making software failures reproducible by preserving object states," in *Proceedings of the 22nd European conference on Object-Oriented Programming*, 2008, pp. 542–565.

[6] W. Jin and A. Orso, "BugRedux: Reproducing field failures for in-house debugging," in *2012 34th International Conference on Software Engineering (ICSE)*. Ieee, jun 2012, pp. 474–484. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6227168

[7] S. Nessa, M. Abedin, W. E. Wong, L. Khan, and Y. Qi, "Software Fault Localization Using N -gram Analysis," in *WASA*, 2008, pp. 548–559.

[8] R. Wu, H. Zhang, S. Kim, and S. Cheung, "Relink: recovering links between bugs and changes," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering.*, 2011, pp. 15–25. [Online]. Available: http://dl.acm.org/citation.cfm?id=2025120

[9] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo, "Recovering traceability links between code and documentation," *IEEE Transactions on Software Engineering*, vol. 28, no. 10, pp. 970–983, oct 2002. [Online]. Available: http://dl.acm.org/citation.cfm?id=630830.631291

[10] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, "What makes a good bug report?" in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering - SIGSOFT '08/FSE-16*. New York, New York, USA: ACM Press, 2008, p. 308. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1453101.1453146

[11] A. Bachmann, C. Bird, F. Rahman, P. Devanbu, and A. Bernstein, "The missing links," in *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering - FSE '10*. New York, New York, USA: ACM Press, nov 2010, p. 97. [Online]. Available: http://dl.acm.org/citation.cfm?id=1882291.1882308

[12] C. Bird, A. Bachmann, F. Rahman, and A. Bernstein, "LINKSTER," in *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering - FSE '10*. New York, New York, USA: ACM Press, nov 2010, p. 369. [Online]. Available: http://dl.acm.org/citation.cfm?id=1882291.1882352

[13] D. Kim, J. Nam, J. Song, and S. Kim, "Automatic patch generation learned from human-written patches," in *2013 35th International Conference on Software Engineering (ICSE)*, vol. 1, no. c. Ieee, may 2013, pp. 802–811. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6606626

[14] K. Pan, S. Kim, and E. J. Whitehead, "Toward an understanding of bug fix patterns," *Empirical Software Engineering*, vol. 14, no. 3, pp. 286–315, aug 2008. [Online]. Available: http://link.springer.com/10.1007/s10664-008-9077-5

[15] C. Bird, A. Bachmann, E. Aune, J. Duffy, A. Bernstein, V. Filkov, and P. Devanbu, "Fair and balanced?" in *Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering on European software engineering conference and foundations of software engineering symposium - E.* New York, New York, USA: ACM Press, aug 2009, p. 121. [Online]. Available: http://dl.acm.org/citation.cfm?id=1595696.1595716

[16] A. Bachmann and A. Bernstein, "Software process data quality and characteristics," in *Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops - IWPSE-Evol '09*. New York, New York, USA: ACM Press, aug 2009, p. 119. [Online]. Available: http://dl.acm.org/citation.cfm?id=1595808.1595830

[17] A. Dangel, "PMD," 2000.

[18] D. Hovemeyer, "FindBugs," 2007.

[19] S. Kim, T. Zimmermann, E. J. Whitehead Jr., and A. Zeller, "Predicting Faults from Cached History," in *29th International Conference on Software Engineering (ICSE'07)*. IEEE, may 2007, pp. 489–498. [Online]. Available: http://dl.acm.org/citation.cfm?id=1248820.1248881

[20] S. Kim, K. Pan, and E. E. J. Whitehead, "Memories of bug fixes," in *Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering - SIGSOFT '06/FSE-14*. New York, New York, USA: ACM Press, nov 2006, p. 35. [Online]. Available: http://dl.acm.org/citation.cfm?id=1181775.1181781

[21] J. E. Montandon, H. Borges, D. Felix, and M. T. Valente, "Documenting APIs with examples: Lessons learned with the APIMiner platform," in *2013 20th Working Conference on Reverse Engineering (WCRE)*. IEEE, oct 2013, pp. 401–408. [Online]. Available: http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=6671315

[22] M. M. Rahman, S. Yeasmin, and C. K. Roy, "An IDE-based context-aware meta search engine," in *2013 20th Working Conference on Reverse Engineering (WCRE)*. IEEE, oct 2013, pp. 467–471. [Online]. Available: http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=6671324

[23] M. Kim, D. Cai, and S. Kim, "An empirical investigation into the role of API-level refactorings during software evolution," *Proceeding of the 33rd international conference on Software engineering - ICSE '11*, p. 151, 2011. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1985793.1985815