The Unexplored Treasure Trove of Phabricator Code Reviews

Kudrjavets, Gunnar; Nagappan, Nachiappan ; Rastogi, Ayushi

The Unexplored Treasure Trove of Phabricator Code Reviews

Gunnar Kudrjavets
University of Groningen
Groningen, Netherlands
g.kudrjavets@rug.nl

Nachiappan Nagappan*
Microsoft Research
Redmond, WA, USA
nnagappan@acm.org

Ayushi Rastogi
University of Groningen
Groningen, Netherlands
a.rastogi@rug.nl

ABSTRACT

Phabricator is a modern code collaboration tool used by popular projects like FreeBSD and Mozilla. However, unlike the other well-known code review environments, such as Gerrit or GitHub, there is no readily accessible public code review dataset for Phabricator. This paper describes our experience mining code reviews from five different projects that use Phabricator (Blender, FreeBSD, KDE, LLVM, and Mozilla). We discuss the challenges associated with the data retrieval process and our solutions, resulting in a dataset with details regarding 317,476 Phabricator code reviews. Our dataset[1] is available in both JSON and MySQL database dump formats. The dataset enables analyses of the history of code reviews at a more granular level than other platforms. In addition, given that the projects we mined are publicly accessible via the Conduit API [18], our dataset can be used as a foundation to fetch additional details and insights.

CCS CONCEPTS

• **Software and its engineering** → *Open source model*; *Programming teams*; *Software version control*.

KEYWORDS

Code review, dataset, mining, Phabricator

ACM Reference Format:
Gunnar Kudrjavets, Nachiappan Nagappan, and Ayushi Rastogi. 2022. The Unexplored Treasure Trove of Phabricator Code Reviews. In *Proceedings of MSR '22: Proceedings of the 19th International Conference on Mining Software Repositories (MSR 2022)*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

A variety of code review datasets are published. Some of the most well-known include Code Review Open Platform (CROP) [16], Gerrit code review dataset [26], and GHTorrent [9]. Several popular open-source software projects (e.g., FreeBSD, LLVM, Mozilla) use a code collaboration tool called Phabricator [24] to conduct their code reviews. We have not found any published code review datasets for

Phabricator. The search of existing literature about mining popular code collaboration tools reveals a study documenting the mining of Gerrit data for Android [15] and GitHub [9]. We can locate only one thesis about mining projects using Phabricator [2]. This thesis describes the development of a data mining tool called Phabry [3]. Phabry, however, cannot be used to collect code changes associated with a code review.

The absence of a readily accessible dataset of code changes for Phabricator projects has deprived code review researchers of a rich information source. The benefit of Phabricator is the ability to formally distinguish between different events taking place during the code review. Each event and action taken during the life cycle of a code review is associated with an author's identity and an event's timestamp. Researchers can track when a code review was accepted, abandoned, taken over by someone else, when a reviewer resigned, when some attributes (e.g., title) were updated, etc.

Table 1: Different events during Phabricator code review.

abandon	create	reopen	subscribers
accept	draft	request-changes	summary
author	inline	request-review	testPlan
close	plan-changes	resign	title
commandeer	projects	reviewers	update
comment	reclaim	status	

Table 1 lists all possible events we noted during the code review life cycle of popular Phabricator projects. We are not aware of any other code review system that tracks events with this level of granularity. GitHub introduced the more formalized code review process, including functionality for actions such as formal acceptance of code changes, only in 2016 [8]. By not utilizing publicly available Phabricator data, researchers miss out on potentially valuable insights and opportunities to study influential and popular software projects with a multi-year development history.

Without a pre-existing accessible dataset, we set out to acquire the Phabricator data ourselves and convert the data to a format suitable for further analysis. Based on our experience, reliably mining data associated with hundreds of thousands of code reviews, even with a pre-existing tool, is an involved and time-consuming process requiring a nontrivial amount of manual labor. We describe the challenges encountered and our solutions in Section 3.4.

The primary motivation behind our paper is to publish a dataset that (a) does not require extra mining effort, (b) includes data about code changes in the code reviews (files changed; lines of code added, deleted, or updated), and (c) can be imported into a relational database system such as MySQL in addition to being published in a plain JSON format.

2 HISTORY AND OVERVIEW

Phabricator was initially developed as an internal code review tool for Facebook in 2011 [25]. As of this paper (November 2021), it is still the de facto code review environment for Facebook and is internally under active development. The public version of Phabricator is developed by a company called Phacility and distributed as open-source software [24].

When compared to other well-known code review environments, such as Gerrit or GitHub, Phabricator introduces some new code review related terminology. For example, the proposed code modifications in Gerrit are referred to as *change* (same as *pull requests* in the context of GitHub). A code review iteration in Gerrit is a version of the change and is called *patch set*. In Phabricator both the initial set of code modifications and its subsequent versions are called *differential revision*, which gets shortened to a *diff*. Committing and merging the accepted changes to the target branch is called *submitting* in Gerrit and *landing* in Phabricator.

Table 2: Descriptive data about Phabricator projects.

Name	Type of software	Year of first diff	Total reviews	Accessible reviews
Blender [1]	Graphics	2013	13,151	13,097 (99.59%)
FreeBSD [6]	OS	2013	32,884	32,725 (99.52%)
KDE [11]	Desktop	2015	29,953	29,874 (99.73%)
LLVM [12]	Compiler	2012	113,372	112,892 (99.58%)
Mozilla [14]	Browser	2017	130,567	128,888 (98.71%)

A wide range of Phabricator projects are publicly accessible. Table 2 lists the projects published by our dataset. We describe the type of project, when the first differential revision was published, the amount of available code review data as of November 2021, and the percentage of code reviews that are publicly accessible. The median age of a project is 8 years and the median number of unique contributors per project is 1,504. Out of 317,476 code reviews, only 258 (0.08%) do not have any associated data quantifying the code changes. The lack of data is caused by changes consisting of binary files or containing only renaming of files.

3 MINING DATA

3.1 Authentication and data access

The Phabricator user community maintains a list of organizations and projects that utilize the tool [21]. We find that resource, in addition to our knowledge from industry experience, to be the best available reference related to Phabricator's usage. Interaction with Phabricator is conducted via Conduit API [18]. Conduit API is a remote procedure call protocol where requests and responses are encoded in JSON (JSON-RPC). To mine Phabricator data, the client needs to have a Conduit API token for authentication purposes. Acquiring the token requires creating a user account for each Phabricator instance to be mined. Account creation can either require a manual approval from a member of the development team (FreeBSD), possession of the GitHub account (Mozilla, LLVM), or just filling out the required registration data (Blender, KDE).

The official API documentation contains only a limited number of examples about its usage. We find that practical experimentation with `curl` [4] or API console is the most efficient way to gain knowledge [22]. To mine the data, one can develop their own tool(s) (something authors of this paper initially did) or utilize existing API wrappers for different programming languages [17]. For our past code review related studies, we have utilized a version of Phabry with minor modifications to facilitate the debugging and adjustments necessary to mine different Phabricator instances [3]. We find that the thesis describing Phabry's development is a detailed and valuable reference about how to interact with Conduit API [2].

3.2 Parsing and interpretation

Data retrieved via Conduit API is returned in JSON format. Our initial instinct was to follow the approach taken in both Gerrit and GHTorent datasets and import the data into a database such as MySQL [9, 26]. Though the output from Conduit API is not documented, building a relational normalized database schema was a straightforward process. The downside of exposing the dataset as a database is the cost associated with maintaining the database instance, importing data, deciding what fields to index, etc. For our studies, we both parse and extract data from JSON directly and use SQL to mainly gather descriptive statistics. That approach proves to be performant even with dataset sizes between 2–3 GBs and up to 135,000 files. The dataset we expose contains both raw JSON files and MySQL database containing the same information.

3.3 Associating differential revisions with code

Each differential revision can evolve through multiple versions. Code changes between each version can differ. To understand the full evolution of the code review it is necessary to keep track of how the code review evolved over the time. However, most code review related studies limit themselves to only the initial or the final version of code changes. In addition, our intent is not to duplicate the data stored in the source control system. For our dataset, we keep track of number of files changed and lines added, deleted, and updated for the final version of the differential revision. We use `diffstat` to calculate the code churn statistics from the raw diff output [5].

There are multiple options for mapping the final code changes to differential revisions. The intuitive approach is to inspect the commit history of a source control system and match the commit content with a differential revision. Listing 1 displays a randomly picked FreeBSD commit using a Phabricator code review process.

Listing 1: Anonymized FreeBSD commit description.

```
commit mrmauqgsbpmdymqzchdtnmxmadimcakrzesmjeil
Author:     John Doe <john.doe@FreeBSD.org>
AuthorDate: 2971410770
Commit:     John Doe <john.doe@FreeBSD.org>
CommitDate: 2971410770

    foo: fix a memory corruption in bar.

    Differential Revision:  https://reviews.freebsd.org/D12345678
```

Based on our analysis, the presence of the string associating a commit with the specific differential revision is *optional* and depends on the project. In addition, we observe typographic errors in the URLs referencing differential revisions and using different

notations when referring to a code review. Therefore, we cannot reliably use the data from commit descriptions to determine what differential revision they are associated with. Another challenge with this approach is handling the presence of many-to-many relationships [7] between commits and differential revisions. A single commit can tag multiple differential revisions and a single differential revision can be referenced from multiple commits. Fetching the data about code changes directly from Phabricator results in a correct representation of final code changes.

3.4 Challenges

3.4.1 Networking. The server hosting Phabricator may apply rate limiting to the number of requests a Conduit API client can issue or the number of network connections the client can make overall. We find that it is necessary to have a retry mechanism in place to mitigate the presence of intermittent errors such as server returning a variety of HTTP error codes, connections timing out, etc. Depending on the specifics of a Phabricator instance, the server may also require a HTTP GET request for one project and PUT request for another (e.g., Blender).

3.4.2 Permissions. During our data mining process we found that there is a subset of differential revisions accessible only to authenticated users, i.e., they cannot be directly downloaded via `curl` without providing the required Conduit API token. We utilize the subsequent usage of getting the metadata about differential revision from `differential.query` [20] and using it to fetch the raw content by calling `differential.getrawdiff` [19].

However, there were some revisions which even an authenticated user could not access. Those differential revisions were a minor part of the overall dataset, accounting for a median of 0.42% of differential revisions per Phabricator instance.

3.4.3 API evolution. Phabricator is distributed as open-source software and each project is free to make any changes needed for their purposes. Depending on the Phabricator instance, the type of data returned by Conduit API may be different. Differences may manifest in the data fields present, action types that can be performed on a differential revision, and if certain fields are optional or mandatory. In some cases, even the data type of the field varies between different Phabricator instances.

4 DATABASE SCHEMA

4.1 Design decisions and data representation

One of the initial design decisions we faced was a choice of exposing the data in the database as close to its original representation in JSON versus using a third normal form [7]. Third normal form is used to reduce data duplication, amount of storage required, and increase the performance of database queries. For simplicity, we chose to match the JSON structure as much as possible unless normalization was needed to represent entries of variable count.

The full relational database schema is presented in Figure 1. Each table has a primary key called *Id*. The foreign key columns referencing parent tables are prefixed with *FK_* and end with the name of a referring table. In modeling the data, we chose to follow the Phabry output directory structure and the Phabricator design concepts. Two essential tables are *revisions* and *transactions*. Transactions in

the context of Phabricator are the history of edits associated with each revision [23]. Each revision belongs to a single Phabricator instance stored in the *instances* table. One revision can have many transactions associated with it. Each transaction belongs to only a single revision. Each revision can have many reviewers and subscribers related to it. A revision can belong to many projects. Each transaction can be associated with many comments, inline comments, a set of changed fields, or many commits. The timestamps (*dateCreated* and *dateModified*) are in Unix time (number of seconds since the Epoch) [10] and represented as integers.

Listing 2: Anonymized FreeBSD revision in JSON format.

```
{
  "id": 1234567890,
  "type": "DREV",
  "phid": "PHID-DREV-viqvtimavobvsqvgbugp",
  "fields": {
    "title": "Title description",
    "uri": "https://reviews.freebsd.org/D1234567890",
    "authorPHID": "PHID-USER-gnuefszwyfdzrescjihs",
    "status": {
      "value": "published",
      "name": "Closed",
      "closed": true,
      "color.ansi": "cyan"
    },
    "repositoryPHID": "PHID-REPO-tucbfqmbgohbfczzvcfg",
    "diffPHID": "PHID-DIFF-hmkchkgoiochmcmfmgor",
    "summary": "Summary of the code changes.",
    "testPlan": "",
    "isDraft": false,
    "holdAsDraft": false,
    "dateCreated": 3053866284,
    "dateModified": 3211780921,
    "policy": {
      "view": "public",
      "edit": "users"
    }
  },
...
```

For example, sample JSON content in Listing 2 represents a subset of a record in the *revisions* table.

5 CONCLUSIONS AND FUTURE WORK

We envision several applications for our Phabricator code review dataset. So far, we have used it to investigate (a) the relationship of code changes to acceptance time, and (b) the presence of non-productive time during the code review process. We are also exploring the dataset for deeper insights into factors impacting code review acceptance.

The current dataset also opens avenues for new research opportunities. For example, (a) utilizing the formal association between code reviews and bugs (tracked by Mozilla project [13]), (b) evolution of differential revisions by analyzing their subsequent versions, and (c) investigating events that occur during code review life cycle that other code collaboration tools do not track.

In this version of the dataset, we chose not to include all the details about each version of the differential revision, such as statistics about code changes per file, file names, etc. Our intention here is to avoid duplication of data stored in a source control system. If such fine-grained data appears to be relevant, the current dataset can be augmented for deeper insights.
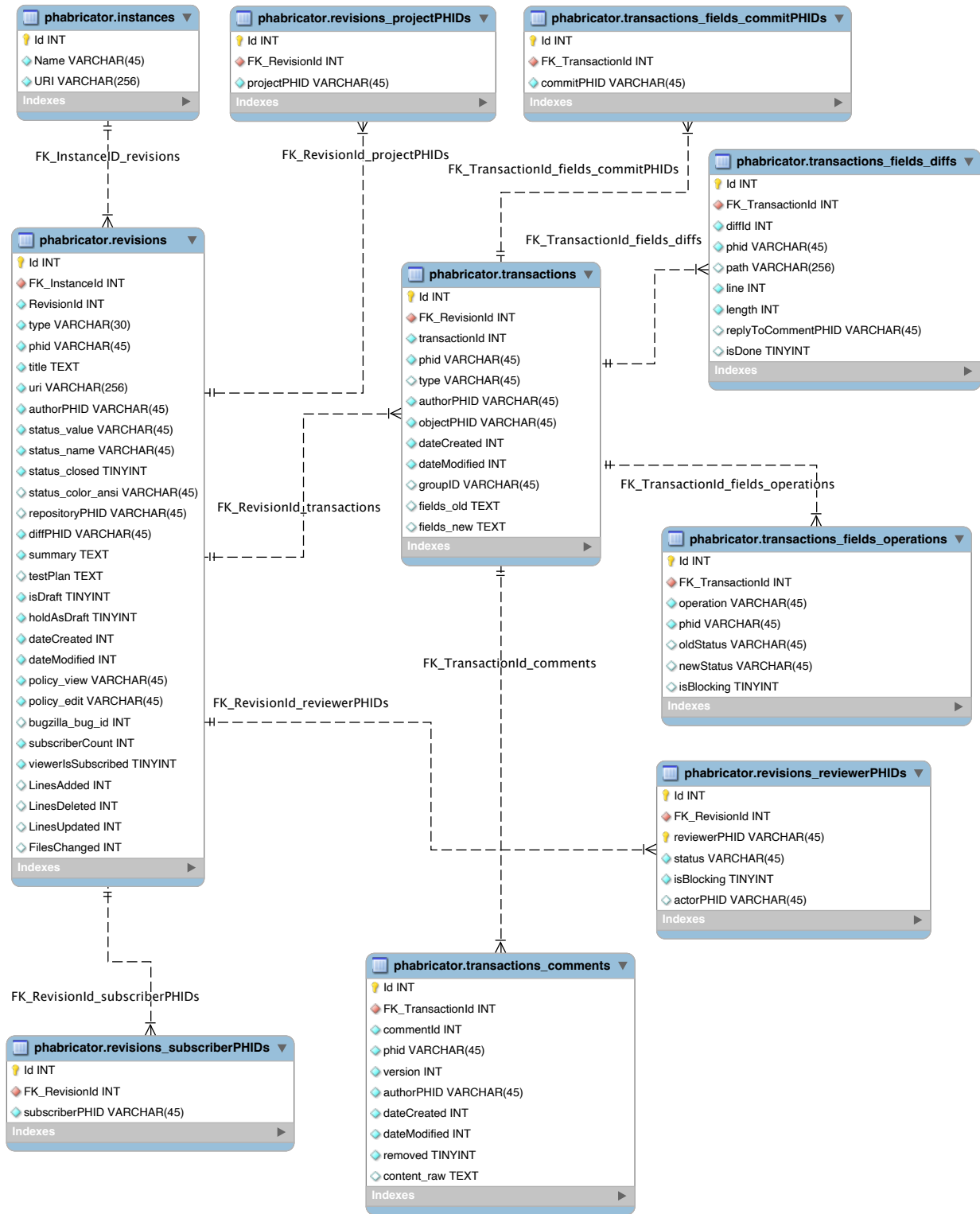
phabricator.instances
- Id INT
- Name VARCHAR(45)
- URI VARCHAR(256)
- Indexes

phabricator.revisions_projectPHIDs
- Id INT
- FK_RevisionId INT
- projectPHID VARCHAR(45)
- Indexes

phabricator.transactions_fields_commitPHIDs
- Id INT
- FK_TransactionId INT
- commitPHID VARCHAR(45)
- Indexes

FK_Instance|D_revisions

FK_RevisionId_projectPHIDs

FK_TransactionId_fields_commitPHIDs

phabricator.transactions_fields_diffs
- Id INT
- FK_TransactionId INT
- diffId INT
- phid VARCHAR(45)
- path VARCHAR(256)
- line INT
- length INT
- replyToCommentPHID VARCHAR(45)
- isDone TINYINT
- Indexes

FK_TransactionId_fields_diffs

phabricator.revisions
- Id INT
- FK_InstanceId INT
- RevisionId INT
- type VARCHAR(30)
- phid VARCHAR(45)
- title TEXT
- uri VARCHAR(256)
- authorPHID VARCHAR(45)
- status_value VARCHAR(45)
- status_name VARCHAR(45)
- status_closed TINYINT
- status_color_ansi VARCHAR(45)
- repositoryPHID VARCHAR(45)
- diffPHID VARCHAR(45)
- summary TEXT
- testPlan TEXT
- isDraft TINYINT
- holdAsDraft TINYINT
- dateCreated INT
- dateModified INT
- policy_view VARCHAR(45)
- policy_edit VARCHAR(45)
- bugzilla_bug_id INT
- subscriberCount INT
- viewerIsSubscribed TINYINT
- LinesAdded INT
- LinesDeleted INT
- LinesUpdated INT
- FilesChanged INT
- Indexes

phabricator.transactions
- Id INT
- FK_RevisionId INT
- transactionId INT
- phid VARCHAR(45)
- type VARCHAR(45)
- authorPHID VARCHAR(45)
- objectPHID VARCHAR(45)
- dateCreated INT
- dateModified INT
- groupID VARCHAR(45)
- fields_old TEXT
- fields_new TEXT
- Indexes

FK_RevisionId_transactions

FK_TransactionId_fields_operations

phabricator.transactions_fields_operations
- Id INT
- FK_TransactionId INT
- operation VARCHAR(45)
- phid VARCHAR(45)
- oldStatus VARCHAR(45)
- newStatus VARCHAR(45)
- isBlocking TINYINT
- Indexes

FK_TransactionId_comments

FK_RevisionId_reviewerPHIDs

phabricator.revisions_reviewerPHIDs
- Id INT
- FK_RevisionId INT
- reviewerPHID VARCHAR(45)
- status VARCHAR(45)
- isBlocking TINYINT
- actorPHID VARCHAR(45)
- Indexes

FK_RevisionId_subscriberPHIDs

phabricator.revisions_subscriberPHIDs
- Id INT
- FK_RevisionId INT
- subscriberPHID VARCHAR(45)
- Indexes

phabricator.transactions_comments
- Id INT
- FK_TransactionId INT
- commentId INT
- phid VARCHAR(45)
- version INT
- authorPHID VARCHAR(45)
- dateCreated INT
- dateModified INT
- removed TINYINT
- content_raw TEXT
- Indexes

Figure 1: Database schema describing Phabricator differential revisions, transactions, and associated entities.

REFERENCES

[1] Blender. 2021. Blender's Phabricator instance. Retrieved November 30, 2021 from https://developer.blender.org/

[2] Dumitru Cotet. 2019. *Crawling Code Review Data From Phabricator*. Master's thesis. Friedrich-Alexander University Erlangen-Nürnberg. Retrieved November 30, 2021 from https://oss.cs.fau.de/2019/08/07/final-thesis-crawling-code-review-data-from-phabricator/

[3] Dumitru Cotet. 2021. Phabry. Retrieved November 30, 2021 from https://github.com/dimonco/Phabry

[4] The cURL project. 2021. Curl. Retrieved November 30, 2021 from https://curl.se/

[5] Thomas E. Dickey. 2021. *diffstat manual*. Retrieved November 30, 2021 from https://invisible-island.net/diffstat/diffstat.html

[6] FreeBSD. 2021. FreeBSD's Phabricator instance. Retrieved November 30, 2021 from https://reviews.freebsd.org/

[7] Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. 2009. *Database Systems: the Complete Book* (2nd ed ed.). Pearson Prentice Hall, Upper Saddle River, N.J. OCLC: ocn191926993.

[8] GitHub. 2016. A Whole New Github Universe: Announcing New Tools, Forums, and Features. Retrieved November 30, 2021 from https://github.blog/2016-09-14-a-whole-new-github-universe-announcing-new-tools-forums-and-features/

[9] Georgios Gousios. 2013. The GHTorrent Dataset and Tool Suite. In *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR '13)*. IEEE Press, San Francisco, CA, USA, 233–236.

[10] IEEE and The Open Group. 2021. The Open Group Base Specifications Issue 7, 2018 edition. Retrieved November 30, 2021 from https://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap04.html#tag_04_16

[11] KDE. 2021. KDE's Phabricator instance. Retrieved November 30, 2021 from https://phabricator.kde.org/

[12] LLVM. 2021. LLVM's Phabricator instance. Retrieved November 30, 2021 from https://reviews.llvm.org/

[13] Mozilla. 2021. Bugzilla Main Page. Retrieved November 30, 2021 from https://bugzilla.mozilla.org/home

[14] Mozilla. 2021. Mozilla's Phabricator instance. Retrieved November 30, 2021 from https://phabricator.services.mozilla.com/

[15] Murtuza Mukadam, Christian Bird, and Peter C. Rigby. 2013. Gerrit Software Code Review Data from Android. In *Proceedings of the 10th Working Conference on Mining Software Repositories* (San Francisco, CA, USA) *(MSR '13)*. IEEE Press, 45–48.

[16] Matheus Paixao, Jens Krinke, Donggyun Han, and Mark Harman. 2018. CROP: Linking Code Reviews to Source Code Changes. In *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*. 46–49.

[17] Phabricator. 2021. Community Resources. Retrieved November 30, 2021 from https://secure.phabricator.com/w/community_resources/

[18] Phabricator. 2021. Conduit API Overview. Retrieved November 30, 2021 from https://secure.phabricator.com/book/phabricator/article/conduit/

[19] Phabricator. 2021. differential.getrawdiff. Retrieved November 30, 2021 from https://secure.phabricator.com/conduit/method/differential.getrawdiff/

[20] Phabricator. 2021. differential.query. Retrieved November 30, 2021 from https://secure.phabricator.com/conduit/method/differential.query/

[21] Phabricator. 2021. Organizations Using Phabricator. Retrieved November 30, 2021 from https://secure.phabricator.com/w/usage/companies/

[22] Phabricator. 2021. Query: Modern Methods. Retrieved November 30, 2021 from https://secure.phabricator.com/conduit/

[23] Phabricator. 2021. transaction.search. Retrieved November 30, 2021 from https://secure.phabricator.com/conduit/method/transaction.search/

[24] Phacility. 2021. Phacility - Home. Retrieved November 30, 2021 from https://www.phacility.com/

[25] Alexia Tsotsis. 2011. Meet Phabricator, The Witty Code Review Tool Built Inside Facebook. Retrieved November 30, 2021 from https://social.techcrunch.com/2011/08/07/oh-what-noble-scribe-hath-penned-these-words/

[26] Xin Yang, Raula Gaikovina Kula, Norihiro Yoshida, and Hajimu Iida. 2016. Mining the Modern Code Review Repositories: A Dataset of People, Process and Product. In *Proceedings of the 13th International Conference on Mining Software Repositories* (Austin, Texas) *(MSR '16)*. Association for Computing Machinery, New York, NY, USA, 460–463. https://doi.org/10.1145/2901739.2903504