Data in brief 25 (2019) 104005

Contents lists available at ScienceDirect

Data in brief

journal homepage: www.elsevier.com/locate/dib

Data Article

The SEOSS 33 dataset — Requirements, bug reports, code history, and trace links for entire projects

Michael Rath [a], [b], [*], Patrick Mäder [b]

[a] *DLR Institute of Data Science, Jena, Germany*
[b] *Technische Universität Ilmenau, Ilmenau, Germany*

A R T I C L E I N F O

Article history:
Received 11 January 2019
Received in revised form 8 May 2019
Accepted 9 May 2019
Available online 24 May 2019

Keywords:
Mining software repositories
Data collection
Data mining
Requirements analysis
Traceability
Bug localization
Issue tracking
Changeset data

A B S T R A C T

This paper provides a systematically retrieved dataset consisting of 33 open-source software projects containing a large number of typed artifacts and trace links between them. The artifacts stem from the projects' issue tracking system and source version control system to enable their joint analysis. Enriched with additional metadata, such as time stamps, release versions, component information, and developer comments, the dataset is highly suitable for empirical research, e.g., in requirements and software traceability analysis, software evolution, bug and feature localization, and stakeholder collaboration. It can stimulate new research directions, facilitate the replication of existing studies, and act as benchmark for the comparison of competing approaches. The data is hosted on Harvard Dataverse using DOI 10.7910/DVN/PDDZ4Q accessible via https://bit.ly/2wukCHc.

* Corresponding author. DLR Institute of Data Science, Jena, Germany.
E-mail address: michael.rath@tu-ilmenau.de (M. Rath).

https://doi.org/10.1016/j.dib.2019.104005

Specifications Table

Subject area	*Computer Software*
More specific subject area	*Software Development Artifacts*
Type of data	*Tables, SQLite databases*
Data format	*Raw, processed and analyzed*
How data was acquired	*Automatic software repository mining and ex- traction*
Data format	*Raw, filtered, analyzed, etc.*
Experimental factors	*Two automated mining algorithms captured software artifacts from open source software.*
Experimental features	*A heuristic technique created trace links among the discovered artifacts.*
Data source location	*The original data are stored in respective project issue tracking systems and version control systems.*
Data accessibility	*Hosted on Harvard Dataverse* *DOI: 10.7910/DVN/PDDZ4Q* *URL: https://bit.ly/2wukCHc*
Related research article	*A subset of the data was used in Rath, M., Lo, D. and Mäder, P., Analyzing requirements and traceability information to improve bug localization. IEEE/ACM* 15th *International Conference on Mining Software Repositories (MSR), 2018, 442–453 [11].*

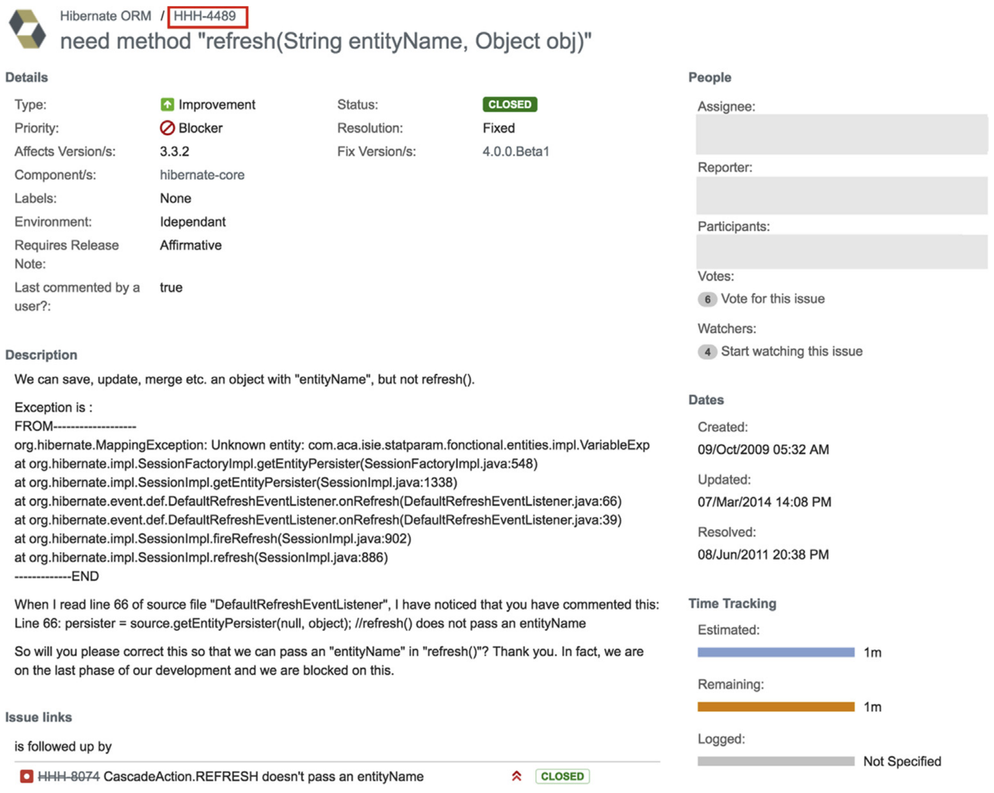
Value of the Data

- Empirical research in the software engineering domain lacks appropriate data. For example, available datasets used for studying software traceability, i. e. the relationships among different development artifacts, are highly restricted in terms of contained artifact types, traces and their overall size. Scientists studying software feature localization and bug localization, bug prediction, or software traceability often propose both: a novel technique and an accompanying dataset. This dataset overcomes the listed limitations and may establish a common baseline and enable direct comparison for benchmarking.
- Large amounts of data is required to perform solid scientific investigations. Especially novel approaches building upon machine learning (ML) and information retrieval (IR) techniques require datasets for training, validation, and testing. This dataset supports these approaches offering development data with $\approx$300.000 issue tracker artifacts, $\approx$ 350.000 version control (source code) artifacts and $\approx$200.000 trace links among them.
- This dataset spans 33 open-source projects from different domains including application servers, databases, web frameworks and widely used programming libraries. The complete timestamped history of each project in terms of typed artifacts (bug reports, feature requests, improvements, source code changes), manifold artifact properties (texts, stakeholder data), and trace links among these artifacts retrieved from issue trackers and source code repositories is included.
- On project level, the dataset combines issue tracking systems and version control systems using trace links. The joint analysis of both systems permits studies that span whole software development processes. This includes bug localization [7,11,15], bug prediction [9], feature localization [3], requirements traceability [2], and studying social aspects [8] of software development.
- The data is provided in the form of relational databases facilitating convenient access, as requested by researchers [6], and easy to embedded in existing approaches.

1. Data

The data stems from 33 open-source projects and their respective issue tracking system (ITS) and version control system (VCS). Fig. 1 shows an example of issue HHH-4489[1] of project Hibernate as represented in the ITS Atlassian Jira.[2] The VCS captures the evolution of a project's source code in form of changesets such as the example[3] from project Hibernate shown in Fig. 2. The retrieval process of all ITS and VCS artifacts is shown in Fig. 4. This process also reliably discovers traceability links among changesets of the VCS and referenced issue artifacts in the ITS. All issues, changesets and respective properties are part of the dataset on a project level and key figures are reported in Table 2. To unify

[1] https://hibernate.atlassian.net/browse/HHH-4489.
[2] https://www.atlassian.com/software/jira.
[3] https://bit.ly/2L4L5Sp.



Hibernate ORM / HHH-4489

need method "refresh(String entityName, Object obj)"

Details

Type:	Improvement	Status:	CLOSED
Priority:	Blocker	Resolution:	Fixed
Affects Version/s:	3.3.2	Fix Version/s:	4.0.0.Beta1
Component/s:	hibernate-core		
Labels:	None		
Environment:	Idependant		
Requires Release Note:	Affirmative		
Last commented by a user?:	true		

People

Assignee: [redacted]

Reporter: [redacted]

Participants: [redacted]

Votes: 6 Vote for this issue

Watchers: 4 Start watching this issue

Description

We can save, update, merge etc. an object with "entityName", but not refresh().

Exception is :
FROM-------------------
org.hibernate.MappingException: Unknown entity: com.aca.isie.statparam.fonctional.entities.impl.VariableExp
at org.hibernate.impl.SessionFactoryImpl.getEntityPersister(SessionFactoryImpl.java:548)
at org.hibernate.impl.SessionImpl.getEntityPersister(SessionImpl.java:1338)
at org.hibernate.event.def.DefaultRefreshEventListener.onRefresh(DefaultRefreshEventListener.java:66)
at org.hibernate.event.def.DefaultRefreshEventListener.onRefresh(DefaultRefreshEventListener.java:39)
at org.hibernate.impl.SessionImpl.fireRefresh(SessionImpl.java:902)
at org.hibernate.impl.SessionImpl.refresh(SessionImpl.java:886)
-------------END

When I read line 66 of source file "DefaultRefreshEventListener", I have noticed that you have commented this:
Line 66: persister = source.getEntityPersister(null, object); //refresh() does not pass an entityName

So will you please correct this so that we can pass an "entityName" in "refresh()"? Thank you. In fact, we are on the last phase of our development and we are blocked on this.

Dates

Created:
09/Oct/2009 05:32 AM

Updated:
07/Mar/2014 14:08 PM

Resolved:
08/Jun/2011 20:38 PM

Time Tracking

Estimated: 1m

Remaining: 1m

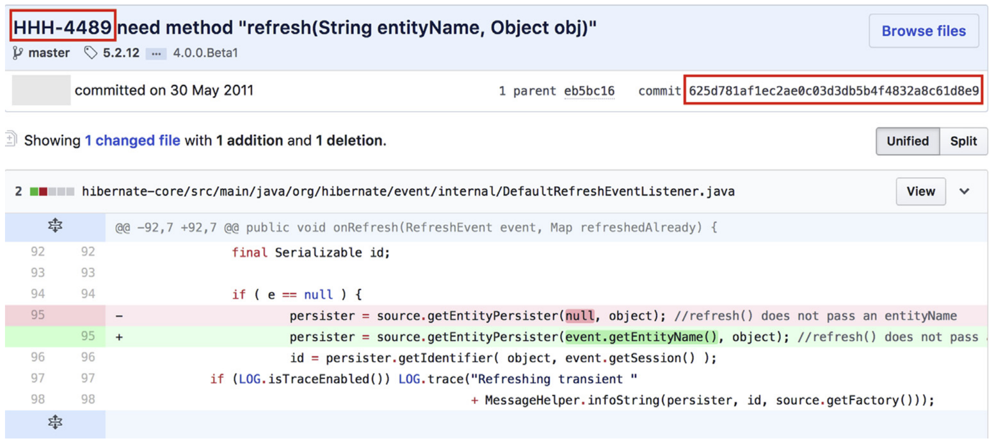
Logged: Not Specified

Issue links

is followed up by

~~HHH-8074~~ CascadeAction.REFRESH doesn't pass an entityName CLOSED

Fig. 1. Example of Jira issue of type *improvement* in project Hibernate. It shows all available data fields (title, description, status, etc) and the unique issue identifier HHH-4489 (marked red). Additionally, the *improvement* has a vertical trace link to issue HHH-8074. Individual-related data such as *reporter* is obfuscated for privacy reasons.



HHH-4489 need method "refresh(String entityName, Object obj)"

master 5.2.12 ... 4.0.0.Beta1

[redacted] committed on 30 May 2011 1 parent eb5bc16 commit 625d781af1ec2ae0c03d3db5b4f4832a8c61d8e9

Showing 1 changed file with 1 addition and 1 deletion.

2 hibernate-core/src/main/java/org/hibernate/event/internal/DefaultRefreshEventListener.java

```
@@ -92,7 +92,7 @@ public void onRefresh(RefreshEvent event, Map refreshedAlready) {
                        final Serializable id;

                        if ( e == null ) {
-                               persister = source.getEntityPersister(null, object); //refresh() does not pass an entityName
+                               persister = source.getEntityPersister(event.getEntityName(), object); //refresh() does not pass
                                id = persister.getIdentifier( object, event.getSession() );
                        if (LOG.isTraceEnabled()) LOG.trace("Refreshing transient "
                                                                + MessageHelper.infoString(persister, id, source.getFactory()));
```

Fig. 2. Example for a git changeset in project Hibernate. It shows the unique commit hash 625d781 … (marked red), a time stamp, the changed source code file, and the line-by-line modifications. The commit messages also contains a reference to an issue tracker artifact (HHH-4489, marked red). Individual-related data is obfuscated for privacy reasons.

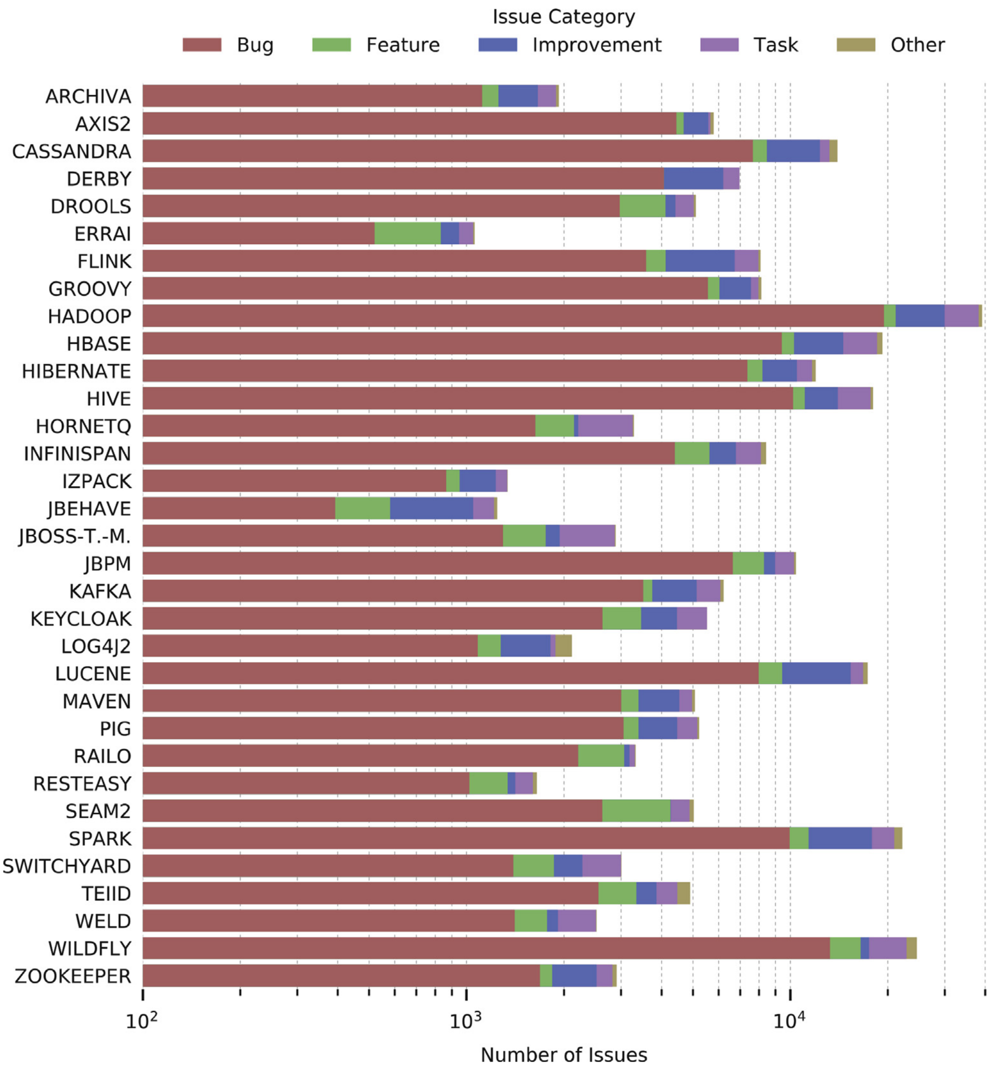


Fig. 3. Issue category distribution according to mapping defined in Table 1. Details are shown in Table 2.

project specific differences, the typed issues, e. g. *New Feature* or *Bug Report*, are mapped to five issue categories (see Table 1) resulting in the distribution shown in Fig. 3. An individual relational database consisting of nine tables (see Table 3) per project is used to store the projects data. Fig. 5 shows the used database schema.

The trace links allow to navigate from the ITS artifacts to VCS artifacts and vice-versa. They serve as a building block for manifold active research areas supporting tasks like defect prevention, change impact analysis, coverage analysis and building recommendation systems.

2. Experimental design, materials, and methods

This section introduces the *Software Engineering in Open Source Systems (SEOSS 33)* dataset and provides an overview and descriptive statistics of the contained projects, artifacts, and their origin.

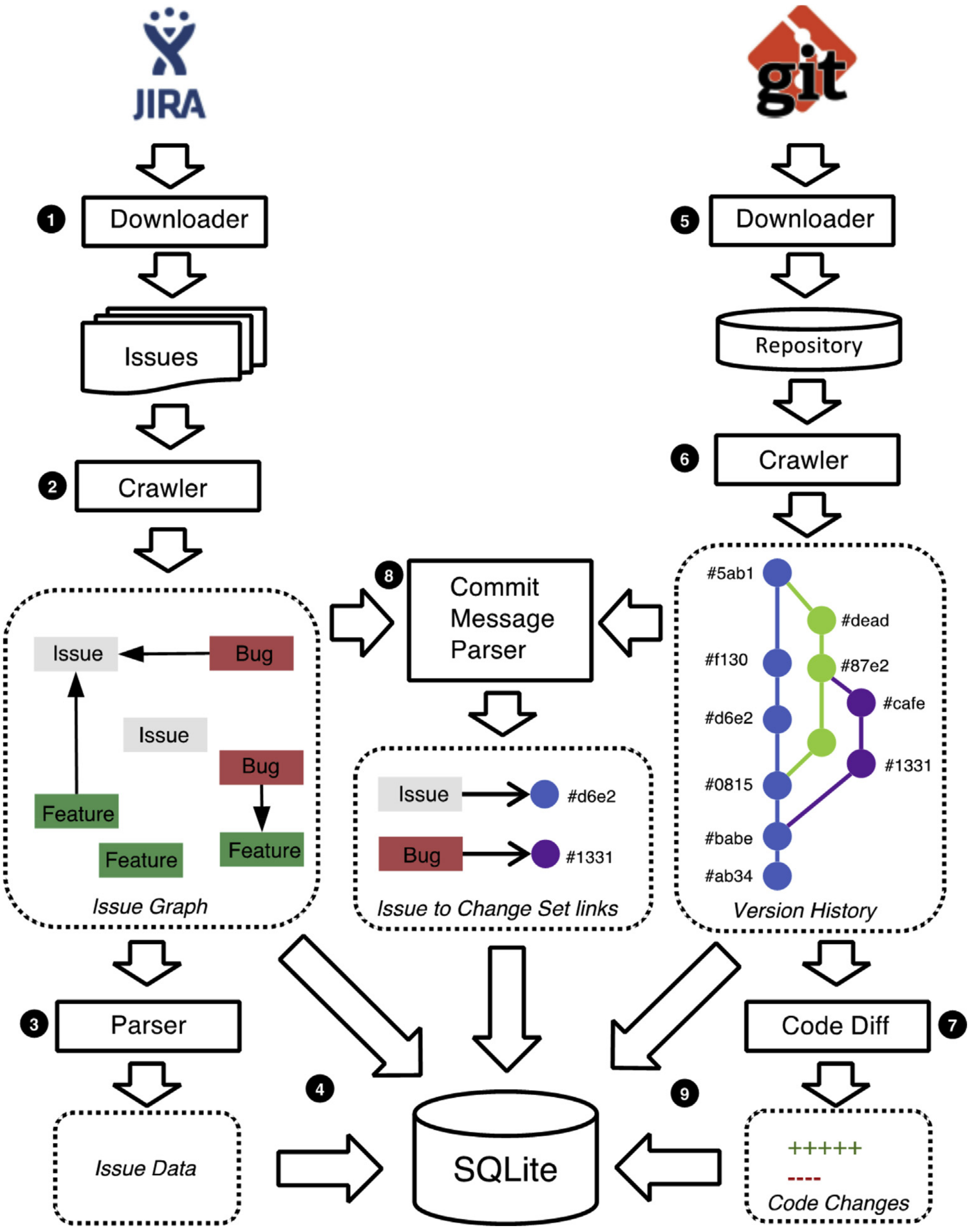


Fig. 4. The applied data collection process per project in the dataset. Infor-mation from each project's ITS and VCS is processed and stored into a combined database.

Table 1
Mapping from Jira issue types and their description to issue categories used in this paper. Both, the original issue type and the mapped typed, are contained in the dataset.

Issue Category	Mapped Jira issue types	Issue Category Description
Bug	Bug	A problem which impairs or prevents the functions of the product.
Feature	Feature, New Feature, Feature Request	A new feature of the product.
Improvement	Improvement, Enhancement	An enhancement to an existing feature.
Task	Task, Sub-task	A task that needs to be done.
Other	*various*: e. g. Brainstorming, Umbrella, Patch, Wish	*miscellaneous*

2.1. Project selection

We first settled on one combination of issue tracking system and version control system used by the projects to be potentially included in our dataset. We selected Atlassian Jira[4] and git[5] based on their popularity studied in various surveys,[6,7] and due to their tight integration. Focusing on one combination of ITS and VCS simplified the development of our data collection process. We also required that the projects in the dataset should be majorly written in one programming language easing analysis and tooling of scientist using the dataset. Considering programming language ranking polls,[8,9] and internet searches[10], we chose Java as main language. Given the tooling restrictions, we considered three major and publicly available hosting providers: the Apache Software Foundation, JBoss, and the Atlassian Cloud. We ran a pre-study, to get a quantitative overview of artifacts and traceability links across all Java projects hosted by these providers. We further narrowed the selection based on the following criteria:

1. A project shall capture a rich set of development artifacts, but at least requirements, bug reports, and source code.
2. A project shall continuously capture vertical and horizontal trace links among these artifacts.
3. A project shall be under active development for at least three years and shall have deployed stable releases.

After applying the information oriented selection strategy Maximum Variation Cases [5] to draw representative samples with varying project characteristics, we eventually selected 33 open-source projects. The detailed project list (File projects.csv) formatted as comma-separated values (CSV) is part of the dataset [12]. It provides the project name, a short description of the project, a link to its webpage, and the name of the database created by the data collection process.

2.2. Issue tracking systems (ITS)

All projects contained in the dataset use the web-based life-cycle management tool Atlassian Jira to manage requirements, defects, and traceability links. In Jira, the fundamental artifact is an uniquely identifiable *issue* capturing a set of properties like a *type*, a short textual *summary*, a long *description* and information about the stakeholder that created it (see Fig. 1). Issues are typed to adapt them to different phases of a development project. The predefined list of issue types contains *bug*, *improvement*, *new*

[4] https://www.atlassian.com/software/jira.
[5] https://git-scm.com.
[6] https://project-management.zone/ranking/category/issue.
[7] https://insights.stackoverflow.com/survey/2018#work-version-control.
[8] https://octoverse.github.com/.
[9] https://spectrum.ieee.org/computing/software/the-2017-top-programming-languages.
[10] https://www.tiobe.com/tiobe-index/.

Table 2
Key figures for the projects of the dataset. Issue links represent vertical and issue to change links horizontal trace links. The time period specifies the range between first and last artifact per project.

	Project	Time Period [month]	Issues	Comments	Change Sets	Issue Links	Issue to Change Set Links	Linked Change Sets [%]
1	Archiva	162	1929	5050	8006	510	2275	26.42
2	Axis2	164	5796	17,146	31,114	484	2609	18.93
3	Cassandra	106	13,965	96,007	23,592	3363	7263	37.03
4	Derby	160	6969	59,829	8156	3529	7263	83.17
5	Drools	146	5103	9763	11,117	633	5528	47.24
6	Erray	99	1060	1216	7645	40	638	8.11
7	Flink	43	8100	69,570	12,419	1895	5328	41.98
8	Groovy	173	8137	23,345	12,378	987	4901	37.91
9	Hadoop	150	39,086	453,765	27,776	23,626	29,309	97.13
10	Hbase	131	19,247	249,923	14,331	7694	13,379	90.06
11	Hibernate	172	11,971	39,339	8173	3110	6942	82.20
12	Hive	113	18,025	132,548	11,179	9110	11,058	96.34
13	Hornetq	158	3286	6274	11,121	732	2283	19.37
14	Infinispan	106	8422	17,382	10,654	2350	7499	68.55
15	Izpack	108	1337	964	5489	73	1804	30.73
16	Jbehave	162	1234	256	3282	4	1546	46.53
17	Jboss-T.-M.	145	2887	6217	2204	814	1615	71.64
18	JBPM	159	10,397	28,631	4919	2211	1133	22.06
19	Kafka	78	6219	32,960	4426	1699	2893	63.17
20	Keycloak	54	5523	7856	10,106	1408	5443	51.73
21	Log4j	117	2114	13,355	9792	488	3309	32.13
22	Lucene	197	17,329	158,194	28,995	4923	19,111	63.12
23	Maven	183	5073	17,298	10,315	1971	2596	24.10
24	Pig	123	5234	29,896	3134	1479	2980	92.85
25	Railo	101	3326	7428	3990	45	989	24.36
26	Restasy	119	1649	4565	3684	280	1345	34.77
27	Seam2	147	5031	12,574	11,309	953	3024	25.88
28	Spark	93	22,205	76,966	20,829	8070	14,306	61.72
29	Switchyard	86	3010	9513	2928	882	2428	80.29
30	Teeid	196	4899	16,808	7266	723	5858	72.38
31	Weld	115	2518	5341	8086	665	2555	30.45
32	Wildfly	203	24,566	59,719	36,711	11,033	22,388	57.71
33	Zookeeper	116	2907	31,251	1600	966	1423	87.12
	SUM		278,536	1,702,949	358,726	96,750	204,593	57.03

feature, and *task*. New issue types and properties can be created as needed for a project. The exact naming of issue types differs among the projects, e.g. some projects use *enhancement* instead of *improvement*, both expressing the same underlying concept. For generalization purposes, we apply a mapping between Jira issue types and a reduced set of five issue categories (see Table 1). However, the original issue type still is available in the dataset, along with the issue category. As depicted in Fig. 3, a majority of issues falls into these four categories, representing issue types present in every project. Examples for issue types categorized as Other are, e. g. “Umbrella” (used in Hbase, Spark), or “Patch” (used in Hibernate, Seam2). Similarly, Jira supports the creation and customization of traceability link types between issue types in addition to a set of predefined link types like *relates to* and *blocks*. For example, the improvement HHH-4489 (see Fig. 1) is linked to bug HHH-8074. Furthermore, comments can be added to issues providing additional details and facilitating collaboration with other developers. Hadoop provides the largest number of issue tracker artifacts in the dataset with ≈40,000 issues, ≈ 23,000 trace links among them and nearly half a million comments (see Table 2).

2.3. Version control system (VCS)

The projects in the dataset use git as VCS. Upon a change, git stores differences to the previous version of each altered file as atomic changeset termed *commit*. Each commit is uniquely identified by a hash value. Besides the atomic set of modified files, a commit also comprises the person performing the

Table 3
Descriptions of the available tables (see Fig. 5) per project's SQLite database in the dataset.

Table	Description
issue	This table contains all artifacts extracted from a project's issue tracker. Each issue includes a unique id, summary, a detailed description, information about type, status, names of people involved, and time stamps. The people names are available as full names (e.g. "John Doe") along with used login names (e. g. "jdoe") which allows to link them to user profiles used in source code version control system.
issue_link	This table represents vertical, typed trace links between issue tracker artifacts. The column name classifies the link (e.g. "Duplicate", "Reference") and column outward_label informs about the reading direction from source issue to target issue (e.g. "duplicates", "relates to"). The field is_containment is 1, when the target issue is a child of source issue.
issue_comment	User comments on the issue. Full names and login names are stored in the same way as for table issue.
issue_component	A mapping table from issue to a software component.
issue_component	The mapping table from issue to a version of the software, where the issue was (or will be) fixed.
issue_fix_version	The mapping table from issue to a version of the software, where the issue was (or will be) fixed.
change_set	This table contains the change sets mined from the projects' source code version control system uniquely identified by a commit hash value. Additional information like the author, time of change and a textual description is also stored.
change_set_link	A mapping table from issue to change sets, which describe horizontal traces.
code_change	All source code changes with file granularity. Every change is linked to a change set and described as number of added and deleted lines of code. The actual modified file con- tent can be easily retrieved from the VCS if required. The columns file_path and old_file_path capture file renaming or moving to a different directory in the source file tree.
meta	A meta table containing detailed information about processed Jira and git repositories for the project, stored as key value pairs. This includes URLs to the used issue tracking systems and git repositories along with respective HEAD revision numbers, and the time stamp when the mining was done.

change, a timestamp, and a commit message stating the purpose of the change (see Fig. 2). A commit message may contain information about, e.g., a newly implemented requirement, a changed or enhanced requirement, or a fixed defect. It is common practice that developers tag commits with the identifiers of issues they were working on when changing the code. In many projects, such as those hosted by the Apache Foundation, this procedure is demanded in the development guidelines, which, e.g., state that *"You need to make sure that the commit message contains at least [...] a reference to the Bugzilla or JIRA issue [...]"*.[11] For example, the changeset 625d781[12] of project Hibernate reads *"HHH-4489 need method "refresh(String entityName, Object obj)""* and mentions the issue identifier HHH-4489.

As shown in Table 2, the dataset contains a total of 350,000 commits, with project Wildfly contributing the largest share. Depending on the project, varying percentages of changesets are linked to issues. For example, in the ERRAI project only 8% of the changesets are linked to issues, whereas in the Hadoop project 97% are linked.

2.4. Data collection process

A sequential capturing process was applied to gather data per project (see Fig. 4). It improves the technique used to create the "IlmSeven dataset" [13], which aimed mainly to be a traceability dataset.

Analyzing a Project's ITS The Jira platform offers a RESTful web service, which allows to interact with the system programmatically. We implemented a collector utilizing the provided application programming interface (API). The collector automatically discovers all artifacts of a project and downloads them to local files stored in *JavaScript Object Notation (JSON)* format (**1**). The downloaded files are then analyzed to extract artifact data and traceability links among these artifacts (**2**). These results are parsed (**3**) and stored into the respective tables of the project's database (**4**).

[11] https://www.apache.org/dev/committers.html#applying-patches.
[12] https://bit.ly/2L4L5Sp.

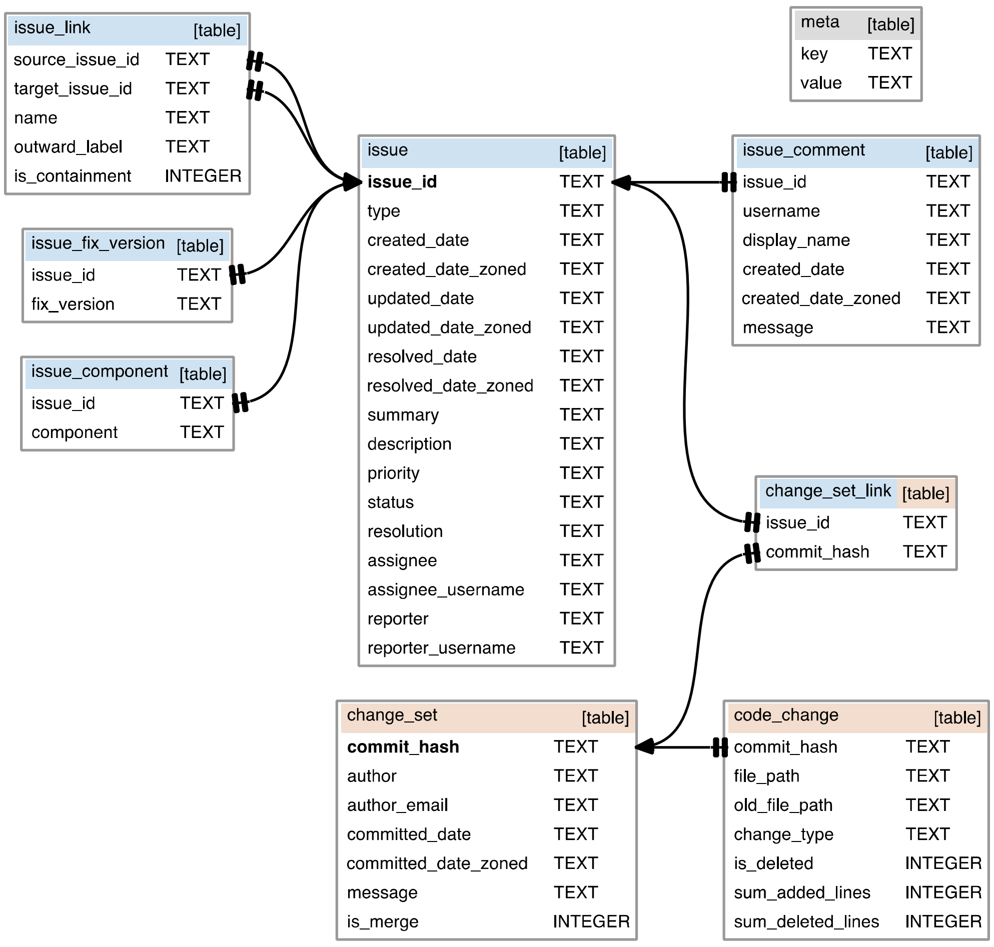


Fig. 5. Database schema used for each project in the dataset, primary keys in bold and arrows depict foreign key references among the tables. The tables highlighted in ▨ contain data retrieved from the project's ITS, while tables highlighted in ▨ contain data retrieved from the project's VCS. The table change_set_link contains the trace links connecting the ITS and VCS.

In Jira, each issue repository uses a common prefix for all contained artifacts, typically derived from the project name. Usually a project uses a single issue repository and thus all issues are prefixed in the same manner (e.g. projects GROOVY, MAVEN, and KAFKA). However, some projects in our set use multiple issue repositories. For example, the Hadoop project uses four repositories[13]: HADOOP, MAPREDUCE, YARN, and HDFS, where each repository holds issues just for a specific part of the software. We determined all issue repositories per project by visiting the projects websites. Relevant meta information about all issue repositories is stored in the meta table contained in each project's database (see Table 3).

Analyzing a Project's VCS A second collector was implemented to download the commit history held in each project's VCS (**5**). The retrieved version history is traversed in inverse chronological order, starting at the latest change (*HEAD* revision) back to the initial commit (**6**). During traversal, basic

[13] http://hadoop.apache.org/issue_tracking.html

information of every visited commit is captured, including the commit time stamp, the author, and the list of affected files along with detailed line-by-line change information (**7**). The content of the source files is not part of the dataset, because of space and efficiency reasons. However the number of added and deleted lines per file in each changeset is stored to quantify the overall size of the change. In case the modified file content is required, it can easily be retrieved using the unique commit hash and file path to query the publicly available VCS. The command git show −pretty = -U <commit_hash> <file_path> shows the modified file content in commit referenced by the supplied commit hash.

It is a common practice to tag commit messages with issue identifiers and thus to create links between source code and issues [14]. To retrieve these links, we scan individual commit messages (**8**) for Jira identifiers.[14] This technique reliably discovers traceability links among the changeset and the referenced issue artifacts in the ITS [1]. For example, this process extracts the issue identifier HHH-4489 in the commit message *"HHH-4489 need method "refresh(String entityName, Object obj)" "* (see Fig. 2). Thus, a vertical trace link is created from improvement HHH-4489 to changeset 625d781a. All captured information is stored in the project's database (**9**), compressed with bzip2. The command bzip2 -d <project_name>.sqlite3.bz2 decompresses a file. We selected SQLite[15] as relational database management system. In contrast to client-server databases, SQLite is easy to setup, is directly embeddable into programs and offers bindings to many programming languages and tools for machine learning and statistical analysis, which eases accessing and working with the data. Each database and thereby project data complies to the same database schema. Depending on the research area specialized storage formats might be beneficial, such as graph databases for traceability analysis [10]. For example, the authors in Ref. [4] outline multiple data transformations from different input representations to a graph database. A similar approach can be applied to this dataset by querying the main artifact tables issue and change_set as well as the tables issue_link and change_set_link representing the links between the artifacts (see Fig. 5).

2.5. *Post processing*

Similar to issue tracking, a project may use multiple version control repositories (e.g. project INFINISPAN[16]). The authors consulted the respective website to discover these repositories. Relevant meta information about all analyzed version control repositories is stored in the meta table per project database, e.g., the latest crawled commit hash (*HEAD* revision).

In order to facilitate usage of the retrieved data for different research topics, we avoided modifications of the raw data. This allows specific post processing steps depending on the actual uses case. For example, these steps may include filtering on artifact level (e. g., requirements, defects), on source code level (e. g. file names or paths) or on a specific temporal window. To simplify time based filtering, all time stamps are provided in universal coordinated time (UTC) and in their original time zone (respective *_zoned columns). All captured text fields are stored as is and may contain markup symbols[17] used for structuring.

2.6. *Database schema description*

Each database and thereby project data complies to the same schema containing of eight related tables and an additional meta-data table (see Fig. 5). The content of all nine tables is described in Table 3. Generally, date and time values are stored with ISO 8601 encoding[18] as text, e.g. 2015-10-21T07:28:00Z. Numbers and booleans are stored as integer and the remaining information are stored as column type text.

[14] Regular expression pattern for Jira identifiers: https://bit.ly/2L2iZHg.
[15] https://www.sqlite.org.
[16] https://github.com/infinispan.
[17] https://bit.ly/2Fh8Vd8.
[18] https://www.iso.org/iso-8601-date-and-time-format.html.

2.7. Data model design

The chosen data model supports multiple use cases and thus is applicable in different research areas.

A common approach in *bug localization* is to use information retrieval (IR) techniques. Thus a bug report is treated as text document to query the projects source code repository, which forms a list of documents in IR terminology. The result of the query is a ranked list of the most relevant matches among all source files (documents). Bug reports are stored the table issue identified by column type = Bug. The projects source code can be locally cloned using the URL stored in the meta table. In bug localization, the algorithm performance can be evaluated by comparing the computed ranked list with the actual set of files modified to fix the bug. This set is built by collecting all file_paths in table code_change for each bug report with resolution = Fixed using the traversal issue → change_set_link → change_set → code_change.

The features (type = Feature) in table issue and their relations (table issue_link) to other issues define the starting point to study *change inpact analysis*. The source code, either on file level or even line level (requires detailed analysis of all code_changes), implementing the feature can be identified using a similar traversal as described in bug localization. Using this information, *recommender systems* can by trained to propose possible code locations that need to be considered in case a feature is altered or a new one is linked to an existing one. One attempt to create a training and testing is to apply temporal filtering, i. e. before and after a given point in time, using the stored timestamps on all involved artifacts.

The core building block for manifold research areas is the navigation between ITS and VCS artifacts as briefly outlined by the two examples. One main difference among the applications is the selection of artifacts [3,15], such as issue types, stakeholder data and texts found in issue descriptions, comments and commit messages for studying social aspects [8] or project evolution based on time information [2].

Acknowledgments

Our work is funded by the BMBF grants: 01IS16003B, DFG grant: MA 5030/3–1, the EU EFRE/TAB grant: 2015FE9033, and DLR grant: D/943/67258261.

Transparency document

Transparency document associated with this article can be found in the online version at https://doi.org/10.1016/j.dib.2019.104005.

References

[1] Adrian Bachmann, Abraham Bernstein, Software process data quality and characteristics: a historical view on open and closed source projects, in: Proceedings of the Joint International and Annual ERCIM Workshops on Principles of Software Evolution (IWPSE) and Software Evolution (Evol) Workshops, IWPSE-Evol '09. ACM, 2009.
[2] Jane Cleland-Huang, Orlena C.Z. Gotel, Jane Huffman Hayes, Patrick Mäder, Andrea Zisman, Software Traceability: Trends and Future Directions, ACM Press, 2014.
[3] Bogdan Dit, Meghan Revelle, Malcom Gethers, Poshyvanyk Denys, Feature location in source code: a taxonomy and survey, J. Softw.: Evol. Process 25 (1) (2013) 53–95.
[4] Randa Elamin, Rasha Osman, Implementing traceability repositories as graph databases for software quality improvement, in: 2018 IEEE International Conference on Software Quality, Reliability and Security, QRS, 2018.
[5] Bent Flyvbjerg, Five Misunderstandings about Case-Study Research, Qual. Inq. (2006).
[6] Ahmed E. Hassan, The road ahead for mining software repositories, in: Frontiers of Software Maintenance, 2008. FoSM 2008. IEEE, 2008.
[7] Andrea De Lucia, Massimiliano Di Penta, Rocco Oliveto, Improving source code lexicon via traceability and information retrieval, IEEE Trans. Softw. Eng. 37 (2) (2011).
[8] Marco Ortu, Giuseppe Destefanis, Bram Adams, Alessandro Murgia, Michele Marchesi, Roberto Tonelli, The JIRA repository dataset: understanding social aspects of software development, in: Ayse Bener, Leandro L. Minku, Burak Turhan (Eds.), Proceedings of the 11th International Conference on Predictive Models and Data Analytics in Software Engineering, PROMISE 2015, ACM, 2015.
[9] Thomas J. Ostrand, Elaine J. Weyuker, Robert M. Bell, Predicting the Location and Number of Faults in Large Software Systems, IEEE Trans. Software Eng. (2005).

[10] Michael Rath, David Akehurst, Christoph Borowski, Patrick Mäder, Are graph query languages applicable for requirements traceability analysis?, in: 22nd International Conference on Requirements Engineering Foundation for Software Quality (REFSQ 2017), 2017.

[11] Michael Rath, David Lo, Patrick Mäder, Analyzing requirements and traceability information to improve bug localization, in: Proceedings of the 15th International Conference on Mining Software Repositories, MSR 2018, 2018.

[12] Michael Rath, Patrick Mäder, The SEOSS Dataset — Requirements, Bug Reports, Code History, and Trace Links for Entire Projects, 2019. https://bit.ly/2V5Zh1S.

[13] Michael Rath, Patrick Rempel, Patrick Mäder, The IlmSeven dataset, in: 25th IEEE International Requirements Engineering Conference, RE 2017, IEEE Computer Society, 2017.

[14] Michael Rath, Jacob Rendall, L. Jin, C. Guo, Jane Cleland-Huang, Patrick Mäder, Traceability in the wild: automatically augmenting incomplete trace links, in: Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, 2018.

[15] Ripon K. Saha, Matthew Lease, Sarfraz Khurshid, Dewayne E. Perry, Improving bug localization using structured information retrieval, in: 2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013, 2013.