This zip file contains The Technical Debt Dataset [1] as a set of CSV files.
In the rest of the document, there is a brief explanation of each of the tables that make up the dataset.

PROJECTS

This table contains the basic information about the projects. It contains the project ID, the project name, as well as the links to the GitHub repository, the Jira issue tracker, and the SonarQube project key.

SONAR_ANALYSIS

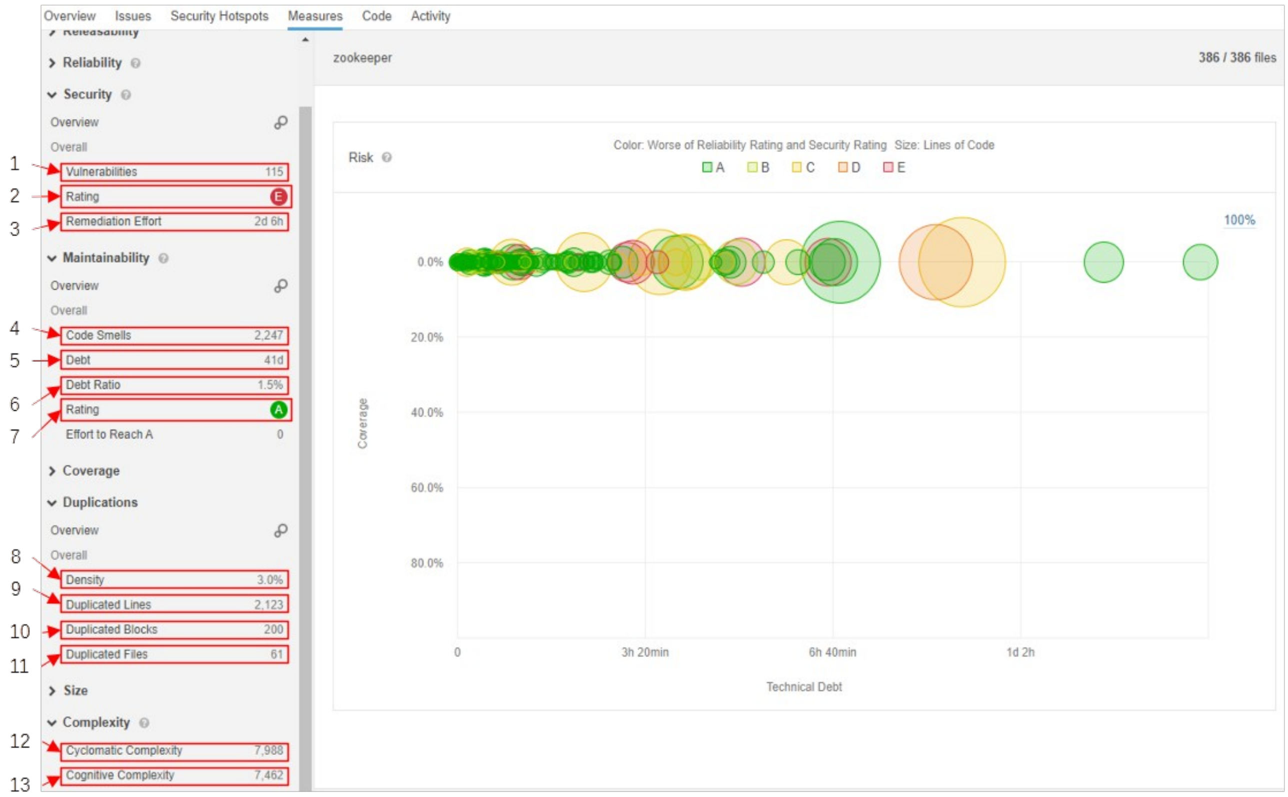
This table contains the analysis keys to join with the SONAR_MEASURES and SONAR_ISSUES tables as well as the analysis date and the revision (i.e. commit hash) that it belongs to. This table allows the integration of the different SonarQube tables with the Git information offered by tables like GIT_COMMITS.

SONAR_MEASURES

SonarQube is one of the most common open-source static code analysis tools for static quality analysis. This table contains the different measures SonarQube analyses from the commits. As an example, in Figure 1 we can see some of the measures analysed which correspond to:

1. **vulnerabilities**
   Number of vulnerability issues.
2. **security_rating**
   A = 0 Vulnerabilities
   B = at least 1 Minor Vulnerability
   C = at least 1 Major Vulnerability
   D = at least 1 Critical Vulnerability
   E = at least 1 Blocker Vulnerability
3. **security_remediation_effort**
   Effort to fix all vulnerability issues. The measure is stored in minutes in the DB. An 8-hour day is assumed when values are shown in days.
4. **code_smells**
   Total count of Code Smell issues.
5. **sqale_index**
   Effort to fix all Code Smells. The measure is stored in minutes in the database. An 8-hour day is assumed when values are shown in days.
6. **sqale_debt_ratio**
   Ratio between the cost to develop the software and the cost to fix it. The Technical Debt Ratio formula is:
   `Remediation cost / Development cost`
7. **sqale_rating**
   Rating given to your project related to the value of your Technical Debt Ratio.
8. **duplicated_lines_density**
   = duplicated_lines / lines * 100
9. **duplicated_lines**
   Number of lines involved in duplications.
10. **duplicated_blocks**
    Number of duplicated blocks of lines.

11. **duplicated_files**
    Number of files involved in duplications.
12. **Complexity**
    It is the Cyclomatic Complexity calculated based on the number of paths through the code. Whenever the control flow of a function splits, the complexity counter gets incremented by one. Each function has a minimum complexity of 1. This calculation varies slightly by language because keywords and functionalities do.
13. **cognitive_complexity**
    How hard it is to understand the code's control flow.



Vulnerabilities	115
Rating	E
Remediation Effort	2d 6h
Code Smells	2,247
Debt	41d
Debt Ratio	1.5%
Rating	A
Effort to Reach A	0
Density	3.0%
Duplicated Lines	2,123
Duplicated Blocks	200
Duplicated Files	61
Cyclomatic Complexity	7,988
Cognitive Complexity	7,462

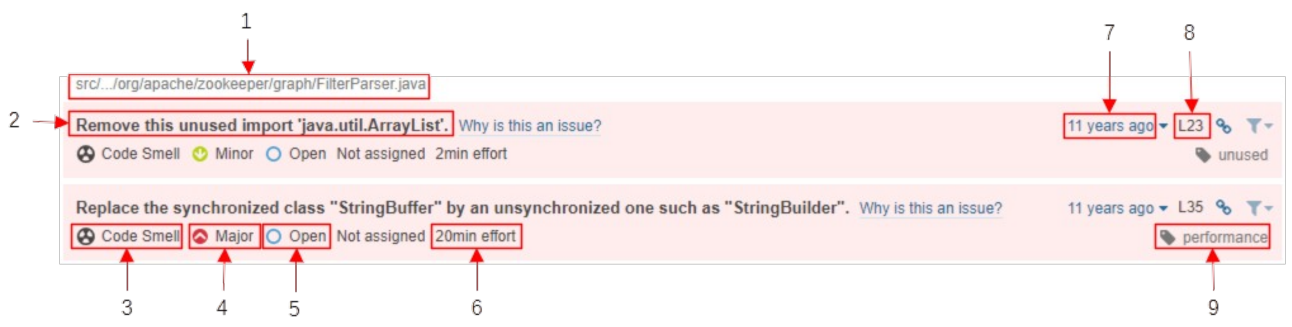
Figure 1: Example of some SonarQube measures.

For more information on the SonarQube measures visit: https://docs.sonarqube.org/latest/user-guide/metric-definitions/

SONAR_ISSUES

This table lists all of the SonarQube issues, as well as the anti-patterns and code smells detected in the analysed projects. In Figure 2 we can see some of the attributes of a SonarQube issue, which correspond to:

1. **COMPONENT**
   File containing the issue.
2. **RULE**
   SonarQube rule that has been triggered.
3. **TYPE**
   Type of the issue, which can be Bug, Vulnerability or Code Smell.
4. **SEVERITY**
   Severity of the issue, which can be Blocker, Critical, Major, Minor or Info.
5. **STATUS**
   After creation, issues flow through a lifecycle, taking one of the following statuses: Open, Confirmed, Resolved, Reopened or Closed.
6. **EFFORT**
   Estimated effort required to fix the issue.
7. **CREATION_DATE**
   Creation date of the issue.
8. **START_LINE**
   Start line of the issue in the file.
9. **TAGS**
   Custom tags of the issue.



src/.../org/apache/zookeeper/graph/FilterParser.java

Remove this unused import 'java.util.ArrayList'. Why is this an issue? 11 years ago L23
Code Smell Minor Open Not assigned 2min effort unused

Replace the synchronized class "StringBuffer" by an unsynchronized one such as "StringBuilder". Why is this an issue? 11 years ago L35
Code Smell Major Open Not assigned 20min effort performance

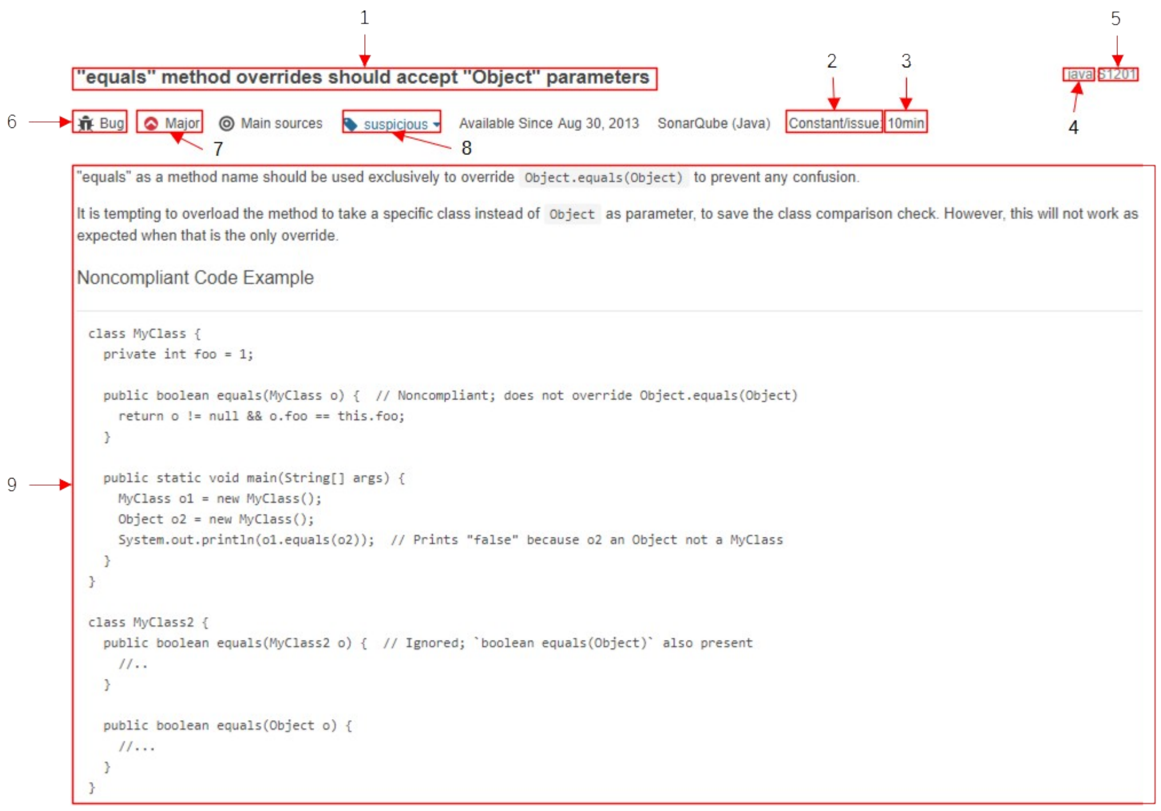
Figure 2: Example of SonarQube issues.

For more information on the SonarQube issues visit:
https://docs.sonarqube.org/latest/user-guide/issues/

SONAR_RULES

This table lists the rules monitored by SonarQube. In Figure 3 we can see some of the most relevant features of this table, which correspond to:

1. **NAME**
   Summary of the rule that serves as its name.
2. **DEF_REMEDIATION_FUNCTION**
   Remediation function used to estimate the effort to solve the issue.
3. **DEF_REMEDIATION_BASE_EFFORT**
   Estimated time to solve the issue.
4. **PLUGIN_NAME**
   Family of rules where the rule belongs to.
5. **PLUGIN_RULE_KEY**
   Rule identifier.
6. **TYPE**
   Type of the issue, which can be Bug, Vulnerability or Code Smell.
7. **SEVERITY**
   Severity of the issue, which can be Blocker, Critical, Major, Minor or Info.
8. **SYSTEM_TAGS**
   Tags used by SonarQube to better group the rules.
9. **DESCRIPTION**
   Description of the rule.



"equals" method overrides should accept "Object" parameters

"equals" as a method name should be used exclusively to override `Object.equals(Object)` to prevent any confusion.

It is tempting to overload the method to take a specific class instead of `Object` as parameter, to save the class comparison check. However, this will not work as expected when that is the only override.

Noncompliant Code Example

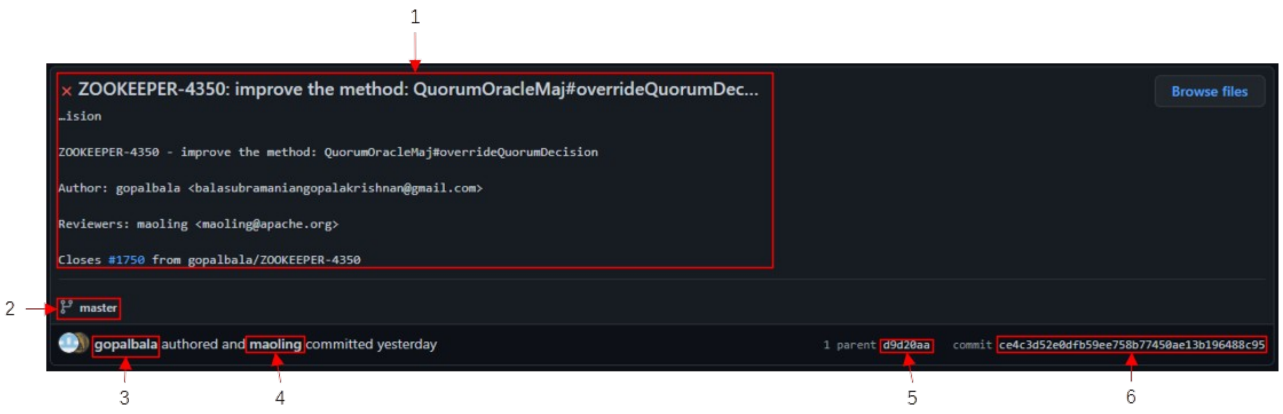
```
class MyClass {
  private int foo = 1;

  public boolean equals(MyClass o) {  // Noncompliant; does not override Object.equals(Object)
    return o != null && o.foo == this.foo;
  }

  public static void main(String[] args) {
    MyClass o1 = new MyClass();
    Object o2 = new MyClass();
    System.out.println(o1.equals(o2));  // Prints "false" because o2 an Object not a MyClass
  }
}

class MyClass2 {
  public boolean equals(MyClass2 o) {  // Ignored; `boolean equals(Object)` also present
    //..
  }

  public boolean equals(Object o) {
    //...
  }
}
```

Figure 3: Example of a SonarQube rule.

GIT_COMMITS

This table reports the commit information retrieved from the git log. As shown in Figure 4, the main attributes of the GIT_COMMITS table are:

1. **COMMIT_MESSAGE**
   Message summarizing the changes introduced with the commit.
2. **BRANCHES**
   Branches where this commit is present.
3. **AUTHOR**
   Developer who created the commit.
4. **COMMITTER**
   Developer who committed the commit.
5. **PARENTS**
   Commit hash of the previous commit.
6. **COMMIT_HASH**
   Commit hash used to identify the commit.



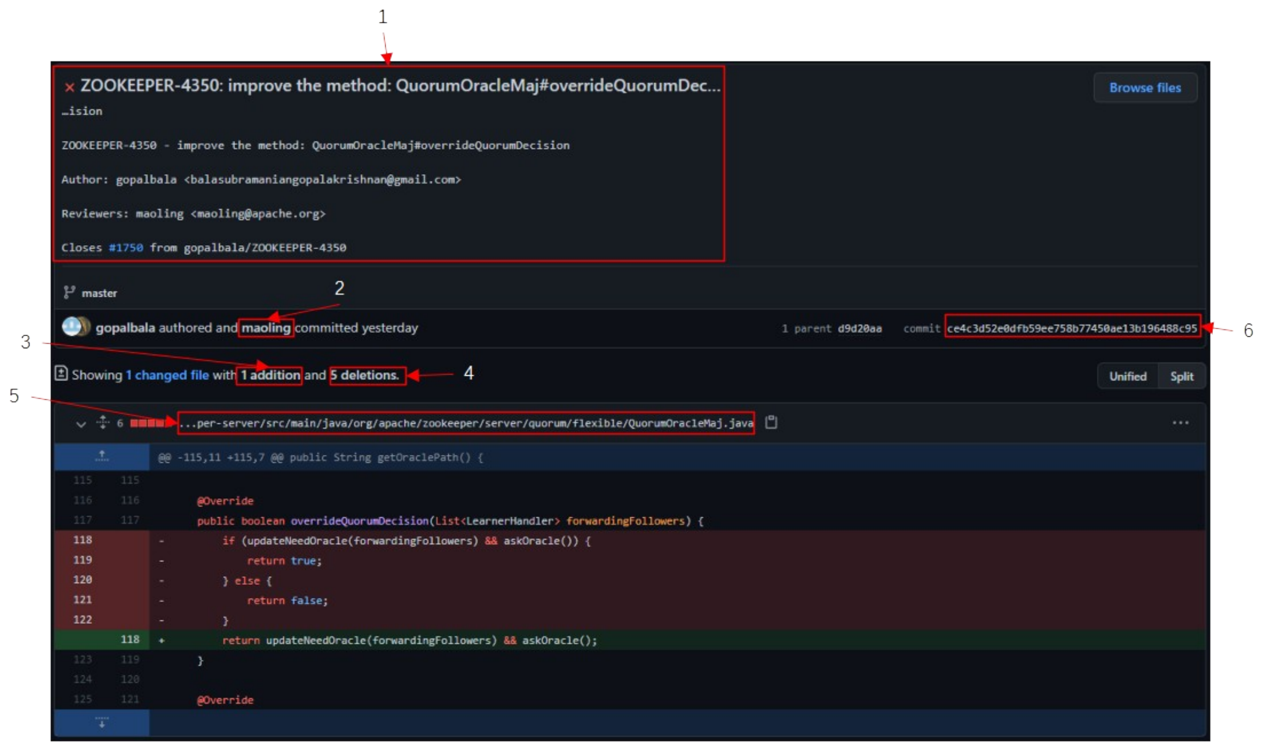
```
× ZOOKEEPER-4350: improve the method: QuorumOracleMaj#overrideQuorumDec…
…ision

ZOOKEEPER-4350 - improve the method: QuorumOracleMaj#overrideQuorumDecision

Author: gopalbala <balasubramaniangopalakrishnan@gmail.com>

Reviewers: maoling <maoling@apache.org>

Closes #1750 from gopalbala/ZOOKEEPER-4350

master

gopalbala authored and maoling committed yesterday          1 parent d9d20aa    commit ce4c3d52e0dfb59ee758b77450ae13b196488c95
```

Figure 4: Example of the main attributes of a git commit.

GIT_COMMIT_CHANGES

This table contains the changes performed in each commit. In Figure 5 we can see the main attributes, which are:

1. **NOTE**
   The description of what has been done in the commit.
2. **COMMITTER_ID**
   The ID of the developer who committed the commit.
3. **LINES_ADDED**
   Number of new lines added.
4. **LINES_REMOVED**
   Number of lines removed.
5. **FILE**
   The full path to the modified file.
6. **COMMIT_HASH**
   Hash used to identify the commit.



ZOOKEEPER-4350: improve the method: QuorumOracleMaj#overrideQuorumDec…

```
…ision

ZOOKEEPER-4350 - improve the method: QuorumOracleMaj#overrideQuorumDecision

Author: gopalbala <balasubramaniangopalakrishnan@gmail.com>

Reviewers: maoling <maoling@apache.org>

Closes #1750 from gopalbala/ZOOKEEPER-4350
```

master

gopalbala authored and maoling committed yesterday   1 parent d9d20aa   commit ce4c3d52e0dfb59ee758b77450ae13b196488c95

Showing 1 changed file with 1 addition and 5 deletions.

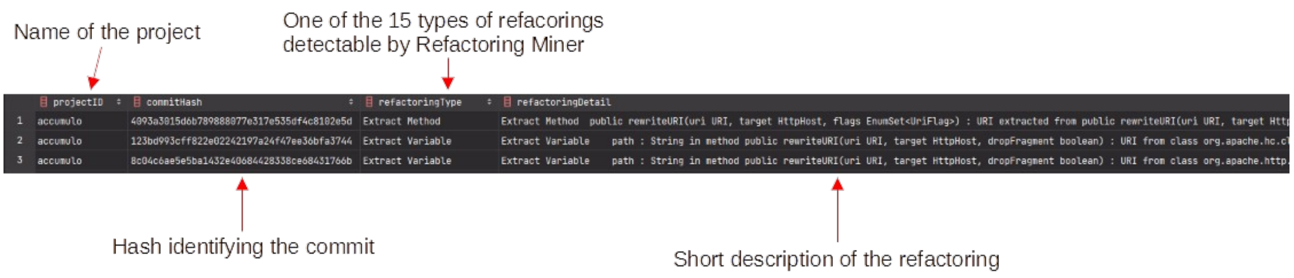
6 ...per-server/src/main/java/org/apache/zookeeper/server/quorum/flexible/QuorumOracleMaj.java

```
@@ -115,11 +115,7 @@ public String getOraclePath() {

      @Override
      public boolean overrideQuorumDecision(List<LearnerHandler> forwardingFollowers) {
-         if (updateNeedOracle(forwardingFollowers) && askOracle()) {
-             return true;
-         } else {
-             return false;
-         }
+         return updateNeedOracle(forwardingFollowers) && askOracle();
      }

      @Override
```

Figure 5: Example of the main attributes of a git commit.

REFACTORING_MINER

Refactoring Miner is an open-source tool that classifies the different refactorings in the history of Java projects. This table reports the list of refactoring activities applied in the studied repositories. Figure 6 shows a small extract of the table describing its attributes.

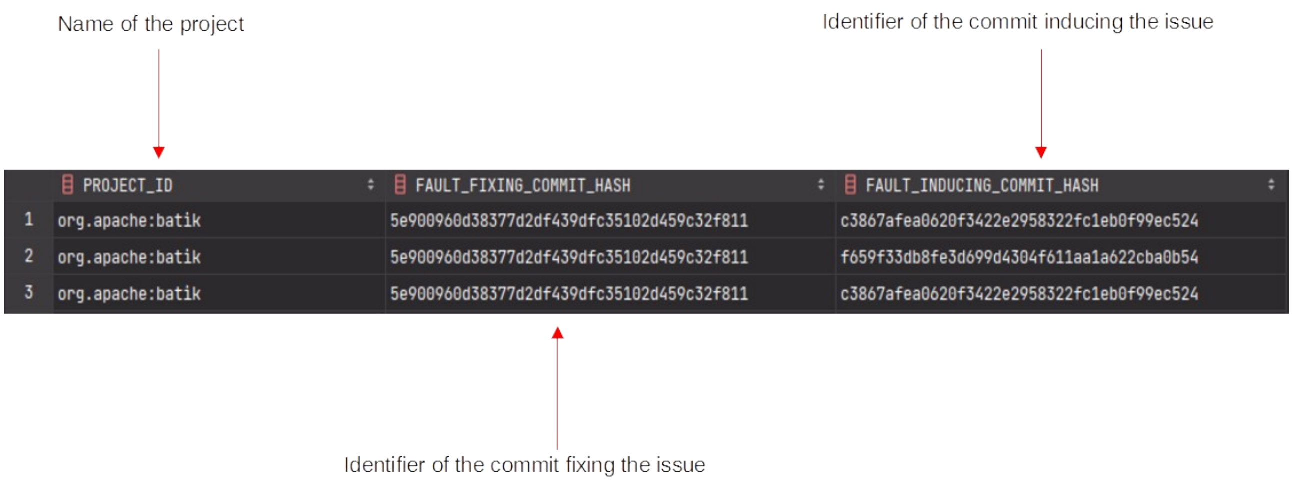

Name of the project
One of the 15 types of refacorings detectable by Refactoring Miner
Hash identifying the commit
Short description of the refactoring

	projectID	commitHash	refactoringType	refactoringDetail
1	accumulo	4093a3015d6b789888077e317e535df4c8102e5d	Extract Method	Extract Method public rewriteURI(uri URI, target HttpHost, flags EnumSet<UriFlag>) : URI extracted from public rewriteURI(uri URI, target Http
2	accumulo	123bd993cff822e02242197a24f47ee36bfa3744	Extract Variable	Extract Variable path : String in method public rewriteURI(uri URI, target HttpHost, dropFragment boolean) : URI from class org.apache.hc.c
3	accumulo	8c04c6ae5e5ba1432e40684428338ce68431766b	Extract Variable	Extract Variable path : String in method public rewriteURI(uri URI, target HttpHost, dropFragment boolean) : URI from class org.apache.http.

Figure 6: Extract of the table REFACTORING_MINER.

SZZ_FAULT_INDUCING_COMMITS

The SZZ algorithm tries to identify the fault-inducing commits from a project's version history. The algorithm is based on Git's blame/annotate feature and assumes that the fault-inducing commit of a fault is known. Usually, this is done by combining data from an issue tracker and form Git's log command. This table reports the results from the execution of the SZZ algorithm, which labels the fault-inducing and -fixing commits. Figure 7 shows a small extract of the table describing its attributes.

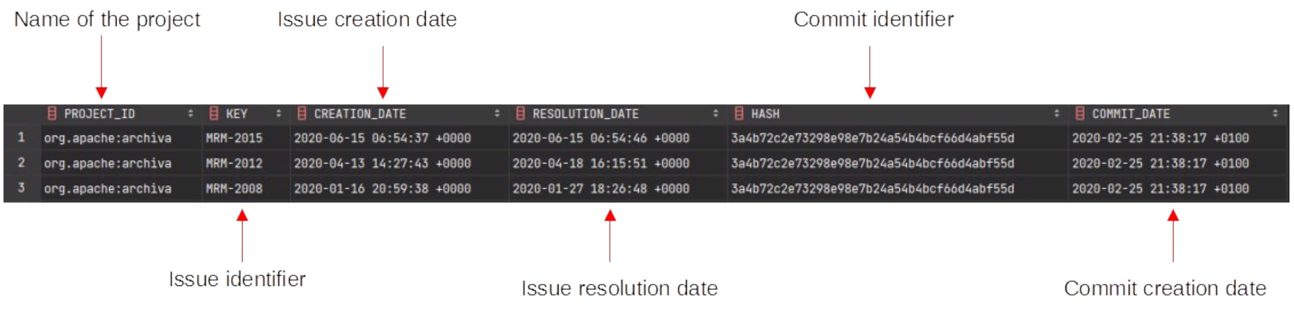

Name of the project
Identifier of the commit inducing the issue
Identifier of the commit fixing the issue

	PROJECT_ID	FAULT_FIXING_COMMIT_HASH	FAULT_INDUCING_COMMIT_HASH
1	org.apache:batik	5e900960d38377d2df439dfc35102d459c32f811	c3867afea0620f3422e2958322fc1eb0f99ec524
2	org.apache:batik	5e900960d38377d2df439dfc35102d459c32f811	f659f33db8fe3d699d4304f611aa1a622cba0b54
3	org.apache:batik	5e900960d38377d2df439dfc35102d459c32f811	c3867afea0620f3422e2958322fc1eb0f99ec524

Figure 7: Extract of the table SZZ_FAULT_INDUCING_COMMITS.

JIRA_ISSUES

Different organizations use Jira to track different kinds of issues, which can represent anything from a software bug to a project task, or a leave request form. This table contains the Jira issues for the analysed projects. Figure 8 shows a small extract of the table describing its attributes.



	PROJECT_ID	KEY	CREATION_DATE	RESOLUTION_DATE	HASH	COMMIT_DATE
1	org.apache:archiva	MRM-2015	2020-06-15 06:54:37 +0000	2020-06-15 06:54:46 +0000	3a4b72c2e73298e98e7b24a54b4bcf66d4abf55d	2020-02-25 21:38:17 +0100
2	org.apache:archiva	MRM-2012	2020-04-13 14:27:43 +0000	2020-04-18 16:15:51 +0000	3a4b72c2e73298e98e7b24a54b4bcf66d4abf55d	2020-02-25 21:38:17 +0100
3	org.apache:archiva	MRM-2008	2020-01-16 20:59:38 +0000	2020-01-27 18:26:48 +0000	3a4b72c2e73298e98e7b24a54b4bcf66d4abf55d	2020-02-25 21:38:17 +0100

Figure 8: Extract of the table JIRA_ISSUES.

Dataset schema

Figure 9 presents the Entity Relationship schema of the dataset. Regarding the fields in the tables, the fields needed to distinguish a row (i.e. primary key) have a key icon, the fields referencing a primary key of another table (i.e. foreign key) have a small arrow above their icon, and "..." means that the table has more fields than presented in the figure.

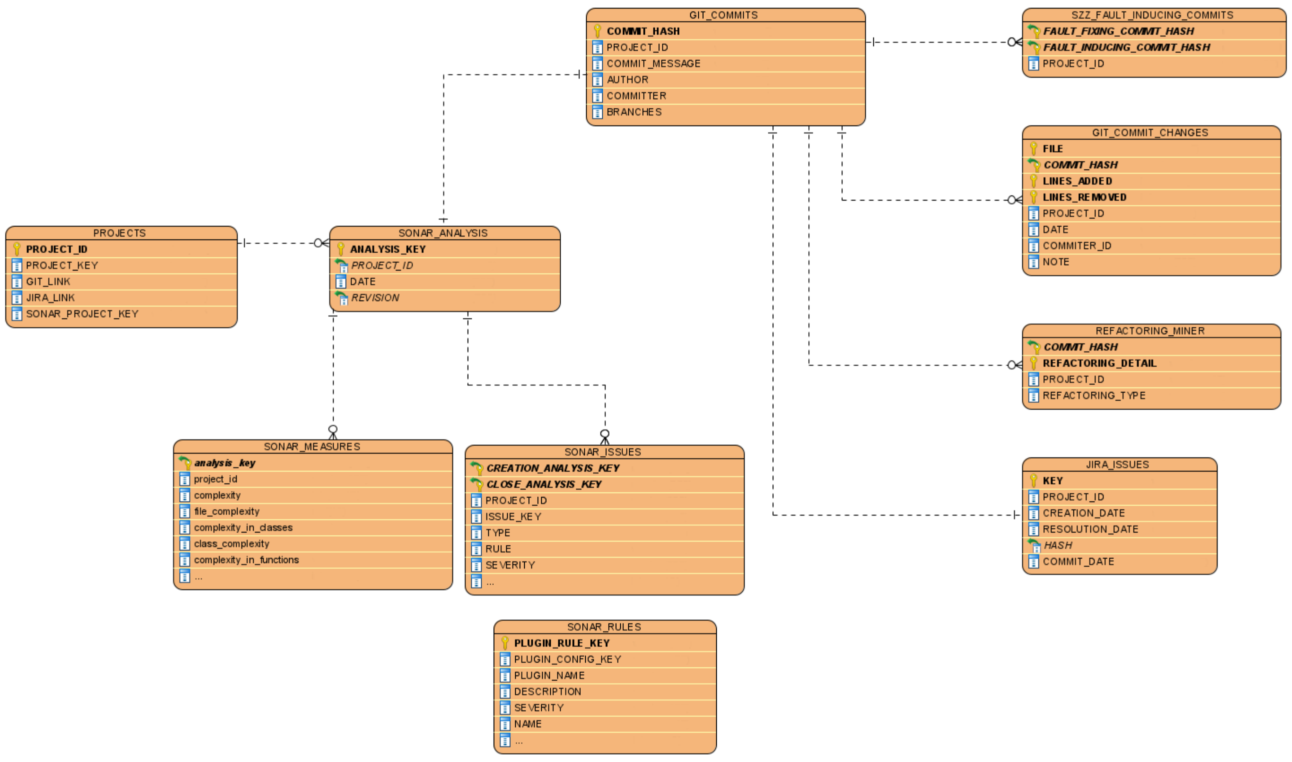


Figure 9: Entity Relationship Diagram of the dataset.

[1] Valentina Lenarduzzi, Nyyti Saarimäki, Davide Taibi. The Technical Debt Dataset. Proceedings for the 15th Conference on Predictive Models and Data Analytics in Software Engineering. Brazil. 2019. Download the paper