Tampere University

Md Sazzad Ul Islam

TECHNICAL DEBT DATASET CLEANSING

Information Technology
Master of Science Thesis
November 2020

ABSTRACT

Md Sazzad Ul Islam: Technical Debt Dataset Cleansing
Master of Science Thesis
Tampere University
Software Engineering - Web and Cloud
November 2020

Technical debt is a common talk among developers and researchers in the software industry. This concept is used to define and find out dirty codes and weak architecture because of shortcuts which will cost to refactor that code in future. There are several tools which analyze the code based on rules and give Technical debt data in returns. This technical debt data helps developers to find the code violation and fix them at an early stage to avoid future refactoring complications. Researchers also use this technical debt dataset for their research purpose. However, not all companies provide public access into their projects technical debt dataset, but there are some already analyzed technical debt dataset available by some research work. The Technical Debt Dataset [3] is one of them which has analyzed technical debt data of 33 Apache Software Foundations Java projects using different tools and methods; besides, this dataset is available in both CSV file and SQLite database format. This dataset itself has hundreds of thousands of data but it has been found that they are not properly formatted. There are different branches technical debt data presents in the dataset and analysis data of those branches was not consistent. That is why there is missing data as well as non-matching data in between the dataset.

In this 'The Technical Debt Dataset Cleansing', we demonstrate the process of the technical debt data fetching from sonarqube, what was the reason behind the cleaning redundant data, how we made the analysis on the data and removed unnecessary data from the dataset. To do that, we fetch the available technical debt data of 33 java projects of Apache software Foundation from SonarQube, then save them into CSV files. Next, we check data in the dataset of individual projects, look for any missing information and non-matching data by comparing within the related dataset of each project. We fill the missing data in the files and try to find matches for non-matching data in the projects' dataset. We also check if any duplicate data is present in the dataset and remove them. This is how we get perfectly matched, no redundant dataset of those 33 projects. Finally, we make the cleaned dataset available in CSV format and an SQLite database which will help others to make queries and get results in a convenient way. The Technical Debt Dataset Cleansing aims to provide non-duplicate, non-redundant, perfectly matched dataset of technical debt data to the researchers. After the cleansing, we successfully generated 67k+ clean sonar analysis data, 1M+ clean sonar issues data and 66k+ clean sonar measures data in new dataset.

Keywords: Technical Debt, Dataset, Cleansing

PREFACE

I want to give the main credit to my supervisor, Davide Taibi, who gave me the opportunity to do my thesis under his supervision. From the very first day, he helped me to understand the problem and shared the idea with me to solve whenever I needed it. His guidance about how to look and understand the problem of my thesis subject helped me a lot to finish my work in time. Also, during my thesis writing, his guidance and motivation of how to write saved me a lot of time. I loved our online meeting (during this Corona pandemic we tried to avoid in-person meetings) where he patiently listened to me and showed me the path to achieve my thesis goal. I really hope that this work and output will help others by providing diverse opportunities.

In Tampere, 20th November 2020

Md Sazzad Ul Islam

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF PROGRAMS AND ALGORITHMS

LIST OF SYMBOLS AND ABBREVIATIONS

CSV	Comma Separated value
SQL	Structured Query Language
TD	Technical Debt

1 INTRODUCTION

In 1992, by Ward Cunningham [2], the Technical debt term was introduced. According to Ward Cunningham, to get a short-term benefit, if a company approaches a quick solution at that moment by violating code or architecture convention, the company just made technical debt. Avgeriou et al. [1] recently defined TD as “In software-intensive systems, technical debt is a collection of design or implementation constructs that are expedient in the short term, but set up a technical context that can make future changes more costly or impossible. Technical debt presents an actual or contingent liability whose impact is limited to internal system qualities, primarily maintainability and evolvability”.

A technical debt causes the decrease of company growth and team productivity. To overcome this downfall of productivity, the company has to pay the interest on the TD. Companies might have no other choice to use quick solutions in some unexpected situations. A quicker solution could be poor coding pattern, duplicate lines, not enough testing, no proper error handling and error log. Not only short-cut solutions, but there are several ways which could lead a company to embrace TD, like, weak software architecture, poor database design, ignore frameworks suggested syntax, memory leak, no recovery plan from a disaster, no backup of code and database to recover after a disaster or no good user documentation. If any of these or most of these happens during the software development cycle and being neglected or unnoticed during that time, the consequences of that negligence will cost the company. If these can not be fixed immediately at the very first stage, then each day passes by, the time and effort to fix that issue will increase and turn into interest in the technical debt.

Since the introduction of TD term, TD is popular and one of the most concerning terms in the software industry. Companies spend most of their time to write code in a way to avoid TD by any means, yet they end up making TD intentionally or unintentionally. Unintentional TD happens if the programmer or intern is not experienced enough and writes less standard code, or a third party tool has been used in the software and the responsible people of that tool stopped to give support. Intentional TD is very common in software development cycles. Sometimes, to release a new feature quickly to possess the market, companies take insignificant decisions. So, the company releases a product with flaws (could be weak architecture or design) in it, try to get short-term benefits and be in top position among its competitors. But, when the time comes to add upon or modify to that weak architecture, at this time it will cost the company extra time, money, resource and effort which slows down the productivity. Besides, sometimes there might

be an urgency to add a new feature on the software as early as possible but there is not enough time or not enough resource to manage, then the company would go for a short-cut solution to cover the present crisis, but it would ultimately create a TD which needs to be paid later.

Software companies always try to follow the software development convention to make the code clean to improve the code quality which makes the software less TD prone. However, the software development cycle is a continuous process, and modifications can come at any point of development, so it is not possible to keep application away from TD during the software development phase. And TD is something that can not be neglected, also has to be fixed as early as possible, it is important to find out the TD at the early stage. There are several TD measurement tools (e.g. Coverity[1], SonarQube[2], Squore[3]) available to perform static analysis on code to have the TD information. Among these tools, SonarQube is the popular one among researchers and developers and this tool is frequently used by 100k+ users [3]. Multi-language static analysis with Reliability, Security, and Maintainability, multiple branches analysis, multiple tools to write clean and safe code makes SonarQube to gain this much popularity in the community. SonarQube has a set of rules and SonarQube monitors the code to check if there is any violation of those rules and alerts developers with proper information (e.g. categories, priorities, amount of time to refactor the violated rules, suggestions of refactor) which helps the developers to refactor the code. [3].

There has been some research on TD data generated from specific TD measurement tools. Mostly, those research has been done on custom TD dataset and that dataset as well as the research result can not be publicly available [3]. So, to do new investigations on TD dataset, either researcher has to use one of TD measurement tools on a custom or open source project to get the TD dataset or depend on the available datasets containing TD data of one or multiple projects. One TD dataset has been found which has been generated after analyzing the TD data of 33 Java projects from Apache Software Foundation. The purpose of that analysis was to generate a technical debt dataset for researchers to work on a common dataset as well as comparing their own research result.

The intention in this thesis work is to find the faulty and redundant data in the previous dataset [3] and then clean the redundant values. Also, there are missing values in the dataset and in this thesis work, several approaches has been taken to find the missing values in the dataset and placed those missing values into the right place. The work is done by Python programming language[4]. Using GitPython[5] library from PyPI[6], all the 33 java projects projects have been cloned dynamically. Besides cloning, at the same time with PyDriller[7] python package, all the commits of a project has been fetched from project

[1] https://scan.coverity.com/
[2] https://www.sonarqube.org/
[3] https://www.squoring.com/
[4] https://www.python.org/
[5] https://pypi.org/project/GitPython/
[6] https://pypi.org/
[7] https://pypi.org/project/PyDriller/

repository. Then the TD data, e.g. sonar measures, sonar analysis and sonar issues, of each project are fetched using respective API from SonarQube cloud and saved as CSV file format. After fetching all the required TD data of each project, the matching has been done in between commits and sonar analysis, sonar analysis and sonar issues, sonar analysis and sonar measures dataset of each project to find and fill missing values and remove all the mismatch and redundant items from the dataset. This is how we got valid dataset without unmatched and redundant values. Finally, this dataset is provided in two different forms, a set of CSV files and one SQLite[8] database. This makes it more convenient to access the dataset for anyone.

The two contributions of this thesis work are:

1. The Technical Debt Dataset Cleansing. The technical debt data of selected apache projects from SonarQube has gone through multi layer checking of missing values of a project, finding the missing values, filling up that missing values, removal of redundant and duplicate values.

2. Providing Two different data formats. Several CSV files and one SQLite database with modified valid dataset makes it easier for researchers to access and go through the dataset and retrieve what they want using functions and query language.

There are other 6 sections remaining in this paper. In Section 2, we describe the dataset in terms of software engineering, describe the previous dataset on which we made our modification and describe some other datasets available online. Section 3 represents what were the problems in the previous dataset and why it needed a cleansing. In Section 4, we go through all the steps we went to to make the previous dataset fault-data free. Section 5 gives the idea of what is new in the new dataset and what was missing in previous dataset. Section 6 provides the importance of this new dataset and how it can be used for further research. Finally we conclude this paper in Section 7.

[8] https://www.sqlite.org/index.html

2 BACKGROUND

It is obvious from the name itself that a 'dataset' is a pool of data, more specifically a set of related information which has been collected, formatted and documented for one or more discrete purposes . A Dataset can be found in different formats: in a file or in a relational database. In both formats, data is presented as tabular format. A tabular-format dataset has one or multiple column(s) and row(s). Dataset can have multiple files, or tables in a database and each file or table contains category wise data. Multiple files or documents will have co-relation in between them. There is no boundary of the number of data or rows in a dataset, it can be hundreds to millions of data in a single file or table in a dataset. The source of data in a dataset can be various as well as the aim of a dataset is, it depends on the intention of the person or team who use that dataset. Not all companies make their datasets available to the rest of the world; majority of them make their own datasets and do research on that. Some companies make their own dataset public for others, so interested minds can access that available dataset and investigate for something new from it.

GH Archive[1] is an example of a dataset. This is a github[2] project which fetches public repositories 20 different types of event data and archives it as hourly. The data contains repositories public information along with JSON encoded events and these events are fetched from GitHub API. This archived dataset of public github repositories events can be downloaded by HTTP client. This data is also available in Google Big Query where the data is stored year, month and day wise. Using SQL query, yearly, monthly or daily data of repositories events can be easily fetched and no need to download the whole dataset. From the dataset, we can have list of the repositories sorted by highest numbers of start ratings, number of commits done by a contributor in a repository and so on. As GH Archive dataset archive GitHub data hourly which helps researchers to get the idea about the evolution of data. The downside of this dataset is it is not up-to-date as the data is evolving every hour, so there is no final version of the dataset.

A research work [3] provides a dataset of technical debt data. This dataset contains analyzed TD data of 33 Java projects from Apache Software Foundation. Different tools and algorithms had been used to collect Technical Debt information from SonarQube, detect code smells, finding fault-point in code and fault-fixing in the available commits of all 33 projects. This dataset is available in both CSV files and SQLite database format.

[1] https://www.gharchive.org/
[2] https://github.com/

The primary intention about generating this dataset was to give an analyzed common dataset to the researchers which will help them for various research works. Though this dataset has good collection of TD data from a good number of projects, but it will be challenging to do further work on it. Mostly because of the dataset has TD data from different branches of a project. As the dataset has multiple branches TD data, this multi branch TD data makes this dataset inconsistent. To make this dataset more research friendly, it needs to get the redundant data to be removed and have a perfectly match among TD data of a project.

3 TECHNICAL DEBT DATASET ISSUES

The Technical Debt Dataset[3] is a very good collection of technical debt data of 33 java projects. All the commits of the projects have been analyzed properly. Also, sonar issues, which has been created and saved by SonarQube during an analysis in a project after a commit, fetched from SonarQube server and stored in the dataset. As per our knowledge, this is the largest dataset[3] contains well-organized technical debt data of java projects which is publicly available in both CSV and SQLite database format.

However, even the dataset is well-organized and easily accessible, but there are a considerable amount of redundant and mismatched data available which might make the work hard for a researcher on this dataset. Mostly, there are three issues have been found in the dataset.

3.1 Multiple branch analyzed

According to the documentation of SonarQube, it has multiple branch analysis support. SonarQube allows the developers or the company to keep the constant code maintainability in different branches. By default, SonarQube runs the analysis in main or master branch, but it is also possible to add more branch to analysis in the configuration. The projects in the current dataset has multiple branches and based on the configuration to run analysis in multiple branches, SonarQube did run analysis on different branche(s) of these projects. As a result, in the current dataset there are TD data of multiple branches. But, in the dataset it is required to have the TD data of a single branch of a project.

3.2 Inconsistent analysis on branch

The SonarQube analysis result in the dataset of a project, sometimes, inconsistent. There are some analysis data which are valid. And there are some analysis data which are missing from the dataset, or even there are sonar analysis data available in the dataset but the other correspondent data of that sonar analysis data could not be found because there are wrong data in the right place in several cases. As a result, there are considerable amount of data in the dataset which origin is ambiguous. This kind of inconsistency is found in the dataset and it makes this dataset hard to work on.

3.3 Different time frame analysis data

As SonarQube starts analysis the project exactly after a commit or merger has been made in the repository branch. Sometimes, the analysis does not starts at the same time when the commit has been made, there might be a small delay of time from commit time to analysis time. If this case happened in a project analysis, then not all technical debt data would match with commit event time. In the dataset, there are some cases where the commit and sonar analysis match could not be found because there are time difference between the commit date time and analysis date time of those cases.

As there are considerable number of redundant, missing and mismatched data in the current dataset, so the dataset needs checking and cleansing to remove idle data, match data where it is needed.

4 TECHNICAL DEBT DATASET CLEANSING

This whole section covers the steps and scenarios have been used to checking and cleaning redundant data, find match of the mismatch data in the current dataset and generate a clean new dataset with perfect matching and valid data. The projects in the current dataset are open-source project and the code of these projects evolves continuously, as well as the technical debt data evolves too in SonarQube. So, at first we will extract all the projects data from different sources. Then we will try different scenario to solve the problems in the current dataset. Finally we will chose the best solution which will give highest number of clean data ratio and we will use it to clean the current dataset.

4.1 Data extraction

Cleansing of TD dataset requires the following information:

- *Projects* - List of all java projects which technical debt data is available in current dataset;
- *Commit Logs* - List History of logs of those projects. The list contains the information of each logs, such as commit hash value, data, message and so on;
- *Sonar Measures* - The metrics history which contains the information about code duplication, complexity, issues, bugs, size and other in a project. The list of metrics report is know as measures;
- *Sonar Analysis* - The list of all sonar analysis event triggered by SonarQube. An analysis includes an unique key and date when the analysis did happens;
- *Sonar Issues* - The list of all sonar issues created by SonarQube when an analysis happens. An issue data includes the project name, file name where the issue found, creation date when the issue created, close date when the issue was closed, type of issue and other information as well;

Now we will describe the data extraction pipeline and stored procedure in depth in the upcoming subsections.

4.1.1 Projects

In the current dataset, there is a CSV file filled with the information about the open source java projects which are available in apache software foundation. Each project has projec-

tID, gitLink, jiraLink and sonarProjectKey. Among these information projectId, gitLink and sonarProjectKey will be used to extract further data. projectId is an unique name given to the project to identify it among the list of the projects. gitLink is the repository url which will be used to clone the project repository and fetch commit history of a project. Finally, sonarProjectKey is an identifier given by SonarQube to each project. This identifier will be used to extract TD data from SonarQube server project wise.

4.1.2 Commit Logs

The commit logs of a projects' repository has been extracted by using PyDriller [4]. Initially, the specific project has been cloned using the gitLink value from projects list. And then we run a simple custom script on the project repository where PyDriller extracted the commit and changes information from the commit history of a project. We run the script on all the projects and fetched the commit and change information, then we saved the information into a CSV file by giving name using sonarProjectKey value to distinguish this file from other file in a directory.

In saved commit information, we have project name, branch info, commit hash value, author of the commit info, list of files which had been modified during that commit, number of lines added and removed, and others info as well. Table 4.1 shows the collected and saved commit information.

4.1.3 Sonar Measures

239 metrics data has been collected using sonar measures api[1] which are available in SonarQube server and then we saved it. The metrics data altogether named as Measures. Each project of the project list has its own measures information populated by metrics data. These metrics are categorized in different category. Table 4.2 shows some categorized metrics which are very common among the 239 metrics.

4.1.4 Sonar Analysis

The available sonar analysis information all the project has been fetched from SonarQube server using sonar analysis api[2]. and then saved as CSV file. After every commit or merge, the sonar scanner starts analyze the projects. Every analysis event is saved by SonarQube. SonarQube saves an analysis event with an unique key as analysis key and a date exactly when the analysis happend. In the analysis file of each project, we have the information shown in Table 4.3.

[1] http://sonar63.rd.tut.fi/web_api/api/measures/search_history
[2] http://sonar63.rd.tut.fi/web_api/api/project_analyses/search

Table 4.1. *Commit and Change information*

Commit Info Type
HASH
AUTHOR NAME
AUTHOR EMAIL
AUTHOR DATE
AUTHOR TIMEZONE
COMMITTER NAME
COMMITTER EMAIL
COMMITTER DATE
COMMITTER TIMEZONE
BRANCHES
IN MAIN BRANCH
IS MERGE COMMIT
MODIFIED FILES
NUM LINES ADDED
NUM LINES REMOVED
COMMIT PARENTS
PROJECT NAME
DMM UNIT SIZE
DMM UNIT COMPLEXITY
DMM UNIT INTERFACING

4.1.5 Sonar Issues

From the SonarQube, we extracted the available issues of every project using sonar issues api[3] and saved them in a CSV file. Each sonar issue data represents the technical debt information created by SonarQube during an analysis of a project. The information we saved in the CSV file is shown in Table 4.4.

4.2 Data matching

We have all the commit information and technical debt data of each project. Now, we will try different methods to match and find valid data. We will try to get maximum number of matching data and then we will use the best strategy to generate a new dataset from the current one.

[3] `http://sonar63.rd.tut.fi/web_api/api/issues/search`

Table 4.2. *Category wise Collected Metrics. Details available in SonarQube Documentation*

Complexity	
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.
Class complexity	The average of Class complexity
Function complexity	The average of Function complexity
Duplication	
Duplicated blocks	Number of duplicated blocks of lines.
Duplicated files	Number of files involved in duplications.
Duplicated lines	Number of lines involved in duplications.
Issues	
New issues	Number of issues raised for the first time in the New Code period.
Issues	Total count of issues in all states.
Maintainability	
Code Smells	Total count of Code Smell issues.
Technical Debt	Effort to fix all Code Smells.
Quality Gates	
Status	Possible state values are : ERROR, OK
Details	Which condition is failing and which is not.
Reliability	
Bugs	Number of bug issues.
New Bugs	Number of new bug issues.
Security	
Vulnerabilities	Number of vulnerability issues.
New Vulnerabilities	Number of new vulnerability issues.
Security remediation effort	Effort to fix all vulnerability issues.
Size	
Classes	Number of classes
Comment lines	Number of lines containing either comment or commented-out code.
Directories	Number of directories.
Files	Number of files.
Functions	Number of functions.
Tests	
Coverage	It is a mix of Line coverage and Condition coverage.
Line coverage	Has this line of code been executed during the execution of the unit tests?
Lines to cover	Number of lines of code which could be covered by unit tests
Uncovered lines	Number of lines of code which are not covered by unit tests.
Unit tests	Number of unit tests.

Table 4.3. *Sonar Analysis Information*

Name	Description
Project	sonarProjectKey as value
Analysis Key	Unique key of an analysis event
Date	Date and time when the analysis happened
Project version	Version of the project. Empty if no value available.
Revision	Hash value of that commit of which the analysis triggered. Empty if no value available.

4.2.1 Match commit, sonar analysis and sonar issues based on modified files and component file

In this method, we try to match and find valid data based on date and file comparison. First, we compare the sonar analysis date value with commit AUTHOR DATE value and then take the data from commit which have been matched. Table 4.5 is an example of sonar analysis data of a project named commons-jexl. Table 4.6 is an example of commit data of project commons-jexl. Now we compare analysis 4.5 date value and commits 4.6 AUTHOR DATE value.

After this comparison we found 2018-01-08 15:04:04 match in both sonar analysis date and commits AUTHOR DATE. We copy the hash value from commit data of matched row and put it in the revision column of matched row in sonar analysis data. Finally we made a combination of data with both sonar analysis data and commit data like figure 4.1 to compare this combined data with issues data.

Figure 4.1. *Commit Analysis match result*

```
{
    "project": "org.apache:commons-jexl",
    "analysis_key": "AWLW41Mz6JGjZv5TYXXn",
    "date": "2018-01-08 15:04:04",
    "revision": "9c113b429bb47e6464450d413178242f9aafc935",
    "modified_files": [
        'RELEASE-NOTES.txt',
        'Operators.java',
        'changes.xml',
        'IssuesTest200.java'
    ]
}
```

Now we take the date value from figure 4.1 and compare it with the creation date and close date of sonar issues 4.7 data. Each row in sonar issues has the information about the file in component column where the issue has been found during the sonar analysis event. Also it is very normal that after each analysis event, multiple sonar issues have

Table 4.4. *Sonar Issues Information*

Name	Description
Project	sonarProjectKey as value
Issue key	An unique key assigned for each issue by SonarQube
Type	There are three types of issues: Bug – A coding error that will break your code and needs to be fixed immediately. Vulnerability – A point in your code that's open to attack. Code Smell – A maintainability issue that makes your code confusing and difficult to maintain.
Rule	SonarQube executes rules on source code to generate issues. 'Rule' value is the id of a rule which has been vioated and found by sonar analysis.
Severity	Each issue has one of five severities: BLOCKER, CRITICAL, MAJOR, MINOR, INFO
Status	After creation, issues flow through a lifecycle, taking one of five possible statuses: Open, Confirmed, Resolved, Reopened, Closed
Resolution	Closed issues will have one of two resolutions: Fixed, Removed
Effort	Needed time to fix the issue
Debt	Effort value measured as debt
Tags	Issues inherit the tags of the rules that created them, but the tag set on an issue is fully editable. Tags can be created, added and removed at will for users with the Browse permission on the project.
Creation date	Date and time when the issue is created
Update date	Date and time when the issue is updated
Close date	Date and time when the issue is closed
Current analysis key	The analysis key from analysis data after matching between update date of issue and date of analysis
Creation analysis key	The analysis key from analysis data after matching between creation date of issue and date of analysis
Close analysis key	The analysis key from analysis data after matching between close date of issue and date of analysis
Message	A description of the issue
Component	The path of file where the issue is found
Start line	Start line of the issue in file
End line	End line of the issue in file
Start offset	Start column in the line of the issue
End offset	End column in the line of the issue
Hash	Checksum value for issue for compare
From hotspot	If the issue is listed in the list of Security Hotspot or not

Table 4.5. *Analysis*

project	analysis key	date
org.apache:commons-jexl	AWLW41Mz6JGjZv5TYXXn	2018-01-08 15:04:04
org.apache:commons-jexl	AWLW4ofA6JGjZv5TYXXi	2018-01-08 14:51:10

Table 4.6. *Commit*

HASH	AUTHOR DATE	MODIFIED FILES
13178242f9aafc935	2018-01-08 15:04:04	['RELEASE-NOTES.txt', 'Operators.java', 'changes.xml', 'IssuesTest200.java']
2cc134431609e4bc8	2018-01-11 14:01:48	['Introspector.java', 'MethodKey.java', 'IssuesTest200.java']

been created by SonarQube. So, if the date in Commit Analysis match result 4.1 is found in either creation date or close date of an sonar issue or multiple sonar issues, we track those rows and copy all the component values of each matched rows and make a set of unique component value, because multiple sonar issue row might have the same component value and we need only unique ones. After extracting the unique component values from the matched sonar issues data, we have the result like figure 4.2.

Figure 4.2. *Unique component list after matching sonar analysis and sonar issues*

```
['IssuesTest200.java', 'Operators.java']
```

Now we need to check if all the items in figure 4.2 list is found in the MODIFIED FILES list in figure 4.1. In this case, all the items in 4.2 is found in the MODIFIED FILES list in figure 4.1. So that we can come into a decision this date is perfect match within commit, sonar analysis and sonar issues data of the project. If all the component items in 4.2 was not found in the MODIFIED FILES list in figure 4.1 then we would mark this analysis date as invalid data.

We used this scenario in some of the project and the match result is shown in table 4.8. From the table, we can see that the match ratio in between commit and analysis is good. But, the match ration in between matched-commit-analysis and issues is not really expected. If we consider felix project, all the analysis date is found in commits AUTHOR DATE, but when we compared that analysis with issues, among 3490 analysis, only 1348 analysis has corresponding issues. As this result is not what we are expecting, we need some in-deep approaches.

We checked the matched and not matched sonar analysis rows in analysis file. In between two matched sonar analysis rows, there are several not matched sonar analysis found and the pattern is shown in table 4.9. In the tabletable 4.9, the first and last analysis

Table 4.7. *Issues*

project	creation date	close date	component
org.apache:commons-jexl	2017-11-29 16:27:04	2018-01-08 15:04:04	IssuesTest200.java
org.apache:commons-jexl	2017-11-29 16:27:04	2018-01-08 15:04:04	IssuesTest200.java
org.apache:commons-jexl	2017-11-29 16:27:04	2018-01-08 15:04:04	IssuesTest200.java
org.apache:commons-jexl	2017-11-29 16:27:04		JexlFeatures.java
org.apache:commons-jexl	2017-11-29 16:27:04	2017-11-29 16:27:04	ScriptVisitor.java
org.apache:commons-jexl	2018-01-08 15:04:04	2018-01-16 19:34:30	IssuesTest200.java
org.apache:commons-jexl	2018-01-08 15:04:04	2018-01-16 19:34:30	Operators.java

Table 4.8. *Match result by date and file in between commit, sonar analysis and sonar issues*

Project	Total analysis	Total analysis and commit match	Total analysis and issue match
commons-jelly	1939	1937	555
commons-jexl	1581	1532	396
felix	3490	3490	1348
httpcomponents-client	2905	2714	951
commons-vfs	2080	2079	507
thrift	2950	1944	145

row are the matched one. In between these two row, the other six sonar analysis rows has matched row in commits but not in sonar issues.

Figure 4.3. *Combined modified files list of not matched analysis*

```
['pom.xml', 'Activator.java', 'ServiceImpl.java']
```

Now we try to find the match of these six unmatched row. First we get the MODIFIED FILES list from commits data of these six unmatched analysis. Then we combine the list of the fetched MODIFIED FILES. Figure 4.3 shows the combined of MODIFIED FILES of the unmatched sonar analysis rows . We then take each unmatched sonar analysis row and find match between the row date and sonar issues create date and close date. For each unmatched sonar analysis comparison with sonar issues, we make a list of unique component values of matched sonar issues rows. If all the item in the list is found in

Table 4.9. *Pattern of matched and not matched analysis rows in project felix*

project	analysis key	date	matched
felix	AWMOmtpnB1UEtThgvljp	2006-04-06 11:49:56	True
felix	AWMOl-TfB1UEtThgvleh	2006-04-06 09:03:39	False
felix	AWMOlYYTB1UEtThgvlah	2006-04-06 08:59:34	False
felix	AWMOk611B1UEtThgvlXZ	2006-04-06 07:29:35	False
felix	AWMOkBgAB1UEtThgvlJT	2006-04-05 17:25:42	False
felix	AWMOiPptB1UEtThgvlG1	2006-04-05 17:20:16	False
felix	AWMOhvBTB1UEtThgvlE8	2006-04-05 16:49:27	False
felix	AWMOdxZUB1UEtThgvke-	2006-04-05 14:19:38	True

the MODIFIED FILES list we previously made 4.3, then we mark that unmatched sonar analysis row in matched. If all the item in the list not found, then that unmatched analysis row remains unmatched.

We run this approach on each unmatched analysis 4.9, and what we found is:

- 2006-04-06 09:03:39 = No matched rows found in issue file, no match analysis
- 2006-04-06 08:59:34 = No matched rows found in issue file, no match analysis
- 2006-04-06 07:29:35 = No matched rows found in issue file, no match analysis
- 2006-04-05 17:25:42 = No matched rows found in issue file, no match analysis
- 2006-04-05 17:20:16 = No matched rows found in issue file, no match analysis
- 2006-04-05 16:49:27 = No matched rows found in issue file, no match analysis

We found sonar issues matched for one date only.

- **2006-04-05 14:57:49**: We have combined all the component columns values of rows which creation date and close date matched with this date. The combined unique components list for this date is looks like figure 4.4. We can see that all the item in this list is not found what we have in 4.3 list. So, we can say that 2006-04-05 14:57:49 analysis is not perfect match.

Figure 4.4. *Combined component list of date 2006-04-05 14:57:49*

```
['InstanceManager.java', 'ServiceImpl.java']
```

As none of six sonar analysis date did not match our criteria, so these six unmatched analysis remains unmatched. If there are matched rows in sonar issues file for these unmatched sonar analysis and all the item in the combined component list found in the combined modified list, we would mark these rows as matched. So far, we only checked six rows, but not the whole felix project sonar analysis. We run this method on project felix's data and the result we found is shown in table 4.10. From the table 4.10 we can see that, the number of matched analysis number (1529) is higher than the previous matched

Table 4.10. *Second approach result to match unmatched analysis in project felix*

Project	Total analysis	Total analysis and commit match	Total analysis and issue match
felix	3490	3490	1529

value (1348) we have in table 4.8. Though the number is higher then previous, but still almost half of the analysis data remains unmatched which is not expected at all.

4.2.2 Match analysis and issues based on dates only

In previous method, we tried to make match using date and file comparison. As per the result, the method did not return a good ratio of match. We have to try another approach. This time we use only date comparison and ignore the commit files and issue file match.

In previous method, we compared commits AUTHOR DATE and sonar analysis date to have match in between a commit row and a sonar analysis row. Also, we copied commit hash value from matched commit row and save it in sonar analysis revision row. Now, we need to check the number and percentage of matching in between commit and sonar analysis data of each project. We run a simple script to get the number of not matched analysis, number of matched analysis and the percentage of match ratio in between commit and sonar analysis of each project. The result of the script is shown in table 4.11. From the ratio we can say that there are good number of matching data in between commit and sonar analysis.

As we now know that, there are good match ration in between commit and sonar analysis, we have now delete the rows from sonar analysis data where the revision value is empty. The revision value empty means those sonar analysis row does not have a match row in commit data. Now we have a clean sonar analysis dataset of all the projects without redundant data. Next, we compare sonar issues with newly created sonar analysis data. Our goal is to clean sonar issues data of all projects by having all the rows in sonar issues data match with newly updated sonar analysis data. While comparison between sonar issues and updated sonar analysis, we strictly checked the following case:

- Count the number of total sonar issues in data file
- Count the number of creation date of sonar issues not found in updated sonar analysis
- Count the percentage of creation date of sonar issues not found in updated sonar analysis
- Count the number of close date of sonar issues not found in updated sonar analysis
- Count the percentage of close date of sonar issues not found in updated sonar analysis

Table 4.11. *Commit hash sonar analysis match report*

Project	**Total analysis**	**Matched**	**%**
batik	2164	2164	100
commons-bcel	1325	1324	99
commons-beanutils	1211	1210	99
cocoon	10210	10206	99
commons-codec	1731	1730	99
commons-collections	2982	2765	93
commons-cli	896	847	95
commons-exec	617	617	100
commons-fileupload	922	864	94
commons-io	2124	1910	90
commons-jelly	1939	1937	99
commons-jexl	1581	1532	96
commons-configuration	2930	2929	99
commons-daemon	982	981	99
commons-dbcp	1912	1555	82
commons-dbutils	645	602	94
commons-digester	2145	2142	99
felix	3490	3490	100
httpcomponents-client	2905	2714	94
httpcomponents-core	1941	1901	97
commons-jxpath	597	596	99
commons-net	2095	2089	99
commons-ognl	608	608	100
santuario	2719	2718	99
commons-validator	1341	1340	99
commons-vfs	2080	2079	99
zookeeper	424	222	53
archiva	4677	4677	100
cayenne	1252	1251	99
hive	5768	5767	99
thrift	2950	1944	66

- Count the number of either creation date and close date of sonar issues not found in updated sonar analysis
- Count the percentage of either creation date and close date of sonar issues not found in updated sonar analysis

The result of this comparison is shown in table 4.12. We have counted total number of sonar issues of every project, compared sonar issues creation date with sonar analysis date and count not-matched sonar issues number, compared sonar issues close date with sonar analysis date and count not-matched sonar issues number, compared either sonar issues creation date or close date with sonar analysis date and count not-matched sonar issues number. We also calculated the ratio of these comparison and not-matched sonar issues for a better understanding how much mismatched is present in the dataset. From the table 4.12, we can see that most of the projects sonar issues has good number of matching in terms of creation date, but the match ratio of sonar issues close date with sonar analysis date is not good. A good number of sonar issues rows did not match when we tried to matched them with close date. Also, there are some projects of which creation date match ratio is not good enough though, for example commons-jelly, commons-digester, sqoop projects' ratio value of missing creation dates row are relatively high enough.

At this moment, we need to find out why every projects' sonar issues close date not-matched ratio is relatively high and why some of the projects sonar issues create date is not-found match with sonar analysis. What we found is, there are many creation date and close date in sonar issues which has matched row in commit data but those dates are not available in sonar analysis. That means, in sonar issues data there are rows which corresponding sonar analysis data is missing from analysis dataset. But, if we check if the creation date and close date of those not matched issues are available in commit data or not, we found a good number of matching. So we can assume that, after a commit of a project, SonarQube starts analysis of the project and during that analysis some issues have been found. Those issues has been added and but somehow analysis event data of those sonar issues were not saved. That is why we could not find a match of sonar issues creation date and close date with sonar analysis date, because sonar analysis data for those issues is not available in the dataset to be matched. Now, in this situation we could not left out those non-matched sonar issues from our dataset, we need to find a solution to keep them in the dataset.

What we do now, we take all the unique creation date and close date from sonar issues which do not have matched sonar analysis rows yet and make a list of them first. Then we try to find the matching commit row of each date from the date list. We compare the date with the AUTHOR DATE value of the commit. If we find a match row of that date, then we copy the HASH value from the commit row, and save project name, the date and copied hash value in a new row in sonar analysis dataset. Table 4.13 shows what it would look like after this approach. In the table, the newly added issues dates in analysis are those rows where there is no analysis key data, because we collected these rows data

Table 4.12. *Missing sonar issues information - 1*

Project	Total issues	Missing creation date	%	Missing creation and close date	%	Missing close date	%
batik	29167	0	0	0	0	14596	51
commons-bcel	8796	0	0	0	0	4671	54
commons-beanutils	5544	0	0	0	0	3482	63
cocoon	77314	0	0	0	0	33106	43
commons-codec	2340	0	0	0	0	1411	61
commons-collections	11777	657	6	657	6	4243	37
commons-cli	37869	880	3	533	2	3812	11
commons-exec	1009	0	0	0	0	561	56
commons-fileupload	1088	359	33	359	33	657	61
commons-io	6266	2188	35	2186	35	4133	66
commons-jelly	8841	3180	36	3180	36	5924	68
commons-jexl	35615	247	1	247	1	1741	5
commons-configuration	7513	0	0	0	0	3663	49
commons-daemon	796	0	0	0	0	703	89
commons-dbcp	7191	3591	50	3591	50	5247	73
commons-dbutils	868	231	27	231	27	575	67
commons-digester	8574	2910	34	2910	34	4818	57
felix	49640	0	0	0	0	34424	70
httpcomponents-client	11123	1002	10	1002	10	4211	38
httpcomponents-core	9531	282	3	282	3	3834	41
commons-jxpath	5826	0	0	0	0	3194	55
commons-net	41635	0	0	0	0	2844	7
commons-ognl	5838	0	0	0	0	1953	34
santuario	51229	0	0	0	0	13286	26
commons-validator	2574	0	0	0	0	1392	55
commons-vfs	4167	0	0	0	0	1758	43
zookeeper	5836	576	10	572	10	5443	94
archiva	75668	0	0	0	0	9752	13
cayenne	22011	0	0	0	0	14011	64
hive	509496	0	0	0	0	59509	12
sqoop	21278	4358	21	4358	21	15711	74
thrift	5226	37	1	37	1	2316	45

Table 4.13. *Add missing sonar issues date to analysis after commit match*

Project	analysis key	date	revision
thrift	AWej8uRlZ1vVHhGU4lHP	2008-06-10 22:54:32	da4671033e333c9df
thrift	AWd6GVTCC4KKKThc-w6	2008-05-16 20:33:47	a5f1d11b69d7cdd1
thrift		2006-09-27 20:51:11	9a8fa9e33b31a68ce
thrift		2008-02-18 02:11:44	fdf6c1ee84b37bd44
thrift		2007-02-21 05:17:48	14214aa6b83ba1d91
thrift		2007-06-01 23:04:09	b83307a8268449f58
thrift		2007-11-03 05:30:32	a42ac6d9987ea5e2d
thrift		2006-08-24 23:37:36	f32d7e780cfe319a6

from commit data and issues date match where there is no analysis key data available in either commit or issues dataset.

Now we run this approach on each project and add the issues dates in analysis data which are missing at this moment. After adding the missing issues in sonar analysis dataset, the new analysis data of each project is shown in table 4.14. From the table, it is obvious that there are some projects (e.g. batik, archiva, cocoon) which now has new rows in the modified sonar analysis data. We can also see the number of previously missing issues, which had match with commits AUTHOR DATE but not with sonar analysis, has been added to the sonar analysis dataset. At this moment, we can say that the analysis data of each project is clean and well matched with commit and issues data.

So far we have cleaned our sonar analysis data. In the newly modified sonar analysis data, we do not have the rows which does not have hash value in revision column. Besides the analysis data has new rows of missing issues dates which previously had match row with commit data but missing from analysis dataset. Now, this time we will compare sonar issues dataset with new sonar analysis dataset and get the information of still how many rows in sonar issues do not have match with sonar analysis. We followed the following conditions:

- Compare issues creation date only with sonar analysis date data, count how many issues rows does not have an analysis date match
- Compare issues creation date and close date both with sonar analysis date data, if any of the creation or close date does not have an sonar analysis match, then count the issue row as not matched

Table 4.14. *Clean sonar analysis result*

Project	Total analysis (Initial)	After commit-analysis match	After adding missing issues	Total missing issues added
batik	2164	2164	2178	14
commons-bcel	1325	1324	1324	0
commons-beanutils	1211	1210	1210	0
cocoon	10210	10206	10221	15
commons-codec	1731	1730	1731	1
commons-collections	2982	2765	2816	51
commons-cli	896	847	847	0
commons-exec	617	617	617	0
commons-fileupload	922	864	865	1
commons-io	2124	1910	1910	0
commons-jelly	1939	1937	1938	1
commons-jexl	1581	1532	1533	1
commons-configuration	2930	2929	2930	1
commons-daemon	982	981	982	1
commons-dbcp	1912	1555	1556	1
commons-dbutils	645	602	603	1
commons-digester	2145	2142	2143	1
felix	3490	3490	3490	0
httpcomponents-client	2905	2714	2714	0
httpcomponents-core	1941	1901	2110	209
commons-jxpath	597	596	597	1
commons-net	2095	2089	2090	1
commons-ognl	608	608	608	0
santuario	2719	2718	2720	2
commons-validator	1341	1340	1340	0
commons-vfs	2080	2079	2080	1
zookeeper	424	222	608	386
archiva	4677	4677	4693	16
cayenne	1252	1251	1259	8
hive	5768	5767	5836	69
thrift	2950	1944	2001	57

- Compare issues close date only with sonar analysis date data, count how many sonar issues rows does not have an sonar analysis date match

We run a script to compare sonar analysis date and sonar issues creation and close date, and the result of not matched sonar issues with sonar analysis is shown in table 4.15.

From the table 4.12, we can see that previously the batik projects' 14596 sonar issue rows close date did not have a match with sonar analysis data. But, in table 4.15, the missing close date sonar issues is 0. So far our new approach worked well. But, still there are some projects where the non matched rows number is either not zero or not close to zero yet. Our aim to bring down the missing row number to either zero or close to zero. So we need to check the pattern of non-matched create date and close date in sonar issues.

After deep observation in the sonar issues file, we found the problem why still some projects' sonar issue data does not have match with sonar analysis. The reason is, what we did earlier, we compared sonar issues row with sonar analysis dataset and if no match found we immediately marked that issue row as not matched.

But, it should not be like that. What we need to do is shown graphically in figure 4.5. If we do not find sonar analysis match for sonar issue row creation date, we mark that row as not matched. If the close date data is empty in a issue row, and that rows creation date has match with sonar analysis, we marked that row as matched row. If both creation date and close date of a issue row found in sonar analysis data, then we marked that row as matched too. Now, if creation date has a matched row in sonar analysis, but close date does not (like number 1 row in 4.5), we remember the component value. Then we go to the second row 4.5 and first check in the component value is same as what we remembered. If the component value is same, then we try to find if both creation and close date of second row has matched sonar analysis or not. If not, then we go the third row, do exactly the same what we did for second row. If both creation and close date of third row does not have matched sonar analysis, we go to the fourth row. Now, the fourth row does not have value in close date cell, so we ignore the fourth row in this special match case. We then go to the fifth row, and do exactly the same what we did for row number 2 and 3. This is how we continue iterating to next row at the end of the list until we find sonar analysis matched for both creation and close date of the issue row. If we do not find a matched sonar issues in the rest of the file, we marked all the non-matched sonar issue rows as not found. In figure 4.5, we have shown that we found a matched row at number seven where both creation and close date are found in sonar analysis, then we replace the creation date value row seven with row one creation date value, mark row 1, 2, 3, 5, and 6 as not found. At the end of this match method, we remove all the not found sonar issue rows.

Using this approach, we will two types of rows in sonar issues.

- Sonar issue row has both creation and close date value, and both date has matched sonar analysis row

Table 4.15. *Missing sonar issues information - 2*

Project	Total issues	Missing creation date	%	Missing creation and close date	%	Missing close date	%
batik	29167	0	0	0	0	0	0
commons-bcel	8796	0	0	0	0	0	0
commons-beanutils	5544	0	0	0	0	0	0
cocoon	77314	0	0	0	0	1	1
commons-codec	2340	0	0	0	0	0	0
commons-collections	11777	657	6	92	1	254	3
commons-cli	37869	880	3	33	1	2753	8
commons-exec	1009	0	0	0	0	0	0
commons-fileupload	1088	359	33	8	1	59	6
commons-io	6266	2188	35	1020	17	1098	18
commons-jelly	8841	3180	36	0	0	2744	32
commons-jexl	35615	247	1	158	1	182	1
commons-configuration	7513	0	0	0	0	0	0
commons-daemon	796	0	0	0	0	0	0
commons-dbcp	7191	3591	50	64	1	797	12
commons-dbutils	868	231	27	0	0	10	2
commons-digester	8574	2910	34	424	5	2198	26
felix	49640	0	0	0	0	0	0
httpcomponents-client	11123	1002	10	127	2	490	5
httpcomponents-core	9531	282	3	22	1	202	3
commons-jxpath	5826	0	0	0	0	0	0
commons-net	41635	0	0	0	0	226	1
commons-ognl	5838	0	0	0	0	0	0
santuario	51229	0	0	0	0	0	0
commons-validator	2574	0	0	0	0	0	0
commons-vfs	4167	0	0	0	0	0	0
zookeeper	5836	576	10	37	1	269	5
archiva	75668	0	0	0	0	0	0
cayenne	22011	0	0	0	0	3	1
hive	509496	0	0	0	0	0	0
thrift	5226	37	1	37	1	350	7

No	project	creation date	close date	component
1	org.apache:commons-jexl	2011-11-29 16:18:59	2018-08-08 16:45:45	pom.xml
2	org.apache:commons-jexl	2018-12-01 16:45:45	2018-08-09 16:45:45	pom.xml
3	org.apache:commons-jexl	2018-12-02 16:45:45	2018-08-10 16:45:45	pom.xml
4	org.apache:commons-jexl	2018-12-03 16:45:45	2018-08-11 16:45:45	pom.xml
5	org.apache:commons-jexl	2018-12-04 16:45:45	2018-08-12 16:45:45	pom.xml
6	org.apache:commons-jexl	2018-12-05 16:45:45	2018-08-13 16:45:45	pom.xml

Figure 4.5. *How to solve sonar issue and sonar analysis mismatch*

- Sonar issue row has only creation date value, close date cell is empty, but the creation date has matched sonar analysis row

Now, it is time to run this method on all our projects. We have run this approach on all of the projects and the results is shown in table 4.16.

From the table 4.16, we can see that the not-matched creation and close date of sonar

Table 4.16. *Matched sonar issues information*

Project	Total issues	Missing creation date	%	Missing creation and close date	%	Missing close date	%
batik	29167	0	0	0	0	0	0
commons-bcel	8796	0	0	0	0	0	0
commons-beanutils	5544	0	0	0	0	0	0
cocoon	77313	0	0	0	0	0	0
commons-codec	2340	0	0	0	0	0	0
commons-collections	10958	0	0	0	0	0	0
commons-cli	34269	0	0	0	0	0	0
commons-exec	1009	0	0	0	0	0	0
commons-fileupload	678	0	0	0	0	0	0
commons-io	4000	0	0	0	0	0	0
commons-jelly	2917	0	0	0	0	0	0
commons-jexl	35344	0	0	0	0	0	0
commons-configuration	7513	0	0	0	0	0	0
commons-daemon	796	0	0	0	0	0	0
commons-dbcp	2867	0	0	0	0	0	0
commons-dbutils	627	0	0	0	0	0	0
commons-digester	3890	0	0	0	0	0	0
felix	49640	0	0	0	0	0	0
httpcomponents-client	9758	0	0	0	0	0	0
httpcomponents-core	9069	0	0	0	0	0	0
commons-jxpath	5826	0	0	0	0	0	0
commons-net	41409	0	0	0	0	0	0
commons-ognl	5838	0	0	0	0	0	0
santuario	51229	0	0	0	0	0	0
commons-validator	2574	0	0	0	0	0	0
commons-vfs	4167	0	0	0	0	0	0
zookeeper	5028	0	0	0	0	0	0
archiva	75668	0	0	0	0	0	0
cayenne	22008	0	0	0	0	0	0
hive	509496	0	0	0	0	0	0
thrift	4876	0	0	0	0	0	0

issues with sonar analysis is zero. That means now in our sonar issues, we do not have any rows which does not have a match with sonar analysis anymore. From this result, we can say that we have cleaned our sonar issues dataset which is now fault free dataset.

So far, in our sonar issues data table 4.4, we do have two columns named current analysis key and creation analysis key. If the creation date of a sonar issue row has match with sonar analysis, then we copied the analysis key from the matched sonar analysis row and put it in the creation analysis key value. The same goes for current analysis key, which is the analysis key for update date of sonar issue row. But, we do not have analysis key for close date of sonar issue. We have used creation date and close date for matching with sonar analysis, so close analysis key column needs to be added in sonar issues dataset. At the end of sonar issues datasest cleansing, we removed current analysis key and update date and added close analysis key column with matched sonar analysis key value.

4.2.3 Match sonar analysis and sonar measures

We have already cleaned redundant data from sonar analysis and sonar issues and made perfect match between these two. We need to clean sonar measures data also. Each sonar measure is a combination of several rule metrics. During an analysis, SonarQube count the number of a specific rule violation and save the number and the analysis date in the rule wise metric. When we extracted the sonar measures, we had date of analysis when that metric counted and the value of the metric. We used that date to compare with sonar analysis date and when find a match, we save the sonar analysis key of matched analysis row in the measures data of a project. We have analysis key in both sonar measures and sonar analysis. We now compare the sonar measures analysis key column value with the clean sonar analysis datasets' analysis key. We have done the comparison and finally removed not matched rows from sonar measures of every project. This is how we have a clean sonar measures dataset where every row has its matched sonar analysis.

5 TECHNICAL DEBT DATASET V2

In the previous section, we have used different method to clean redundant data and match related sonar analysis and sonar issues data from The Technical Debt Dataset. In our modified dataset, we have valid and properly matched technical debt data of the projects. This new dataset is the version 2 of The Technical Debt dataset. We have saved this new dataset in both CSV files and a SQLite database which gives the benefits to fetch data from the dataset file very easily.

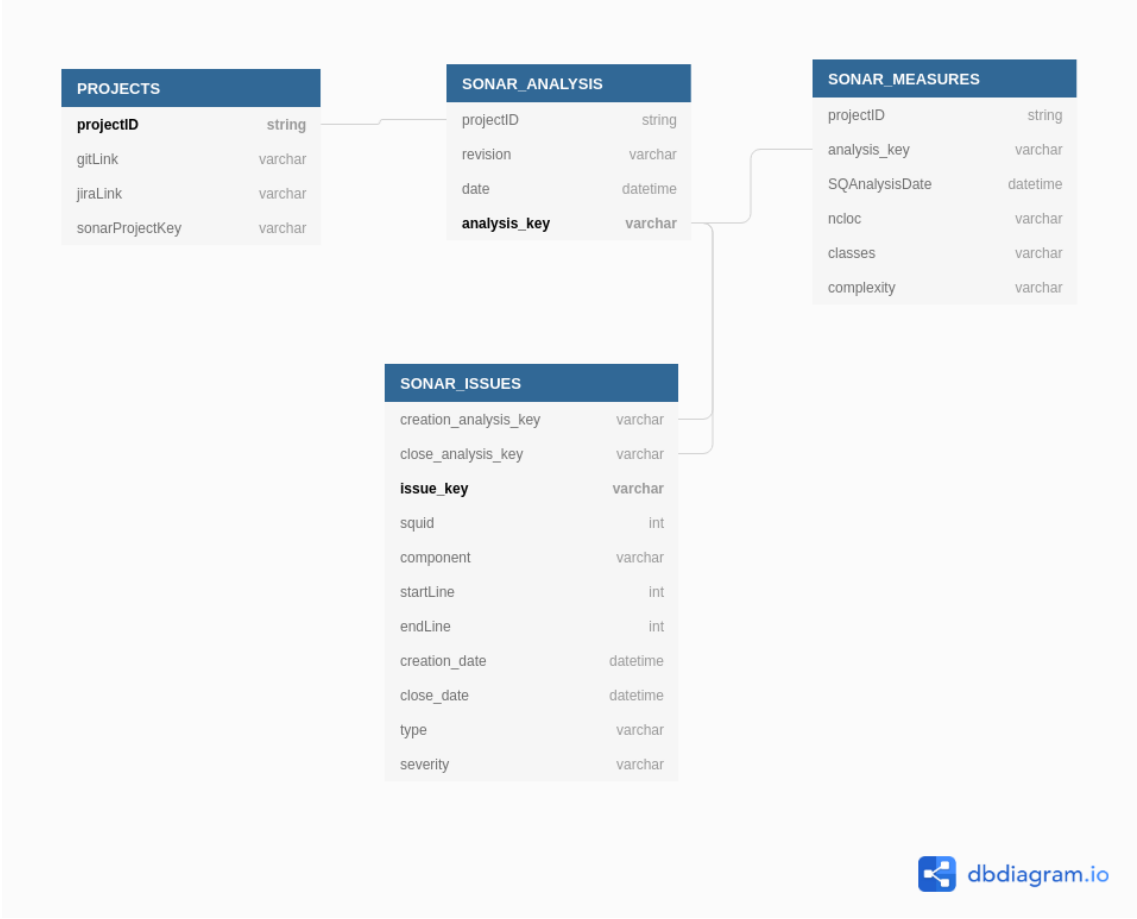

Figure 5.1. *Entity relationship diagram of dataset V2*

We have a projects table in the dataset, we did the cleansing on these projects' data. Each project has a unique projectID. This projectID value is also available in sonar measures and sonar analysis table. So, to get the sonar analysis value of a project using SQL query would be like 5.1.

```
SELECT * FROM SONAR_ANALYSIS
JOIN PROJECTS
ON SONAR_ANALYSIS.projectID = PROJECTS.projectID
WHERE SONAR_ANALYSIS.projectID = 'project-id';
```

Listing 5.1. *An SQL Query to get all the sonar analysis of a project*

To get the sonar measures value of a project using SQL query would be like 5.2.

```
SELECT * FROM SONAR_MEASURES
JOIN PROJECTS
ON SONAR_MEASURES.projectID = PROJECTS.projectID
WHERE SONAR_MEASURES.projectID = 'project-id';
```

Listing 5.2. *An SQL Query to get all the sonar measures of a project*

Previously, we compared sonar analysis and sonar issues using date value. As we have now analysis key in both creation analysis key column and close analysis key, we do not have to use date comparison any more. If we want to get the issues of a project created by analysis, the SQL query would be like 5.3.

```
SELECT * FROM SONAR_ISSUES
JOIN SONAR_ANALYSIS
ON SONAR_ISSUES.creation_analysis_key = SONAR_ANALYSIS.analysis_key
JOIN PROJECTS
ON SONAR_ANALYSIS.projectID = PROJECTS.projectID
WHERE PROJECTS.projectID = 'project-id';
```

Listing 5.3. *An SQL Query to get the sonar issues create by analysis of a project*

We can compare our newly generated technical debt dataset with previous one. Figure 5.2 shows the entity relationship of the previous dataset. In previous dataset, it does not have sonar analysis data. Because sonar measures has the commit hash info already. commitHash value in the sonar measures entity is used to make join in between sonar GIT COMMITS, SONAR MEASURES and SONAR ISSUES data tables.

Also, unlike our new dataset, previous dataset does not have analysis key inormation in SONAR ISSUES data table. Instead, it has commit hash information in creation commit hash column and close commit hash column. These hash values of SONAR ISSUES in previous dataset is used to make join with SONAR MEASURES, because unlike our dataset, previous dataset has commit hash value in SONAR MEASURES where we have analysis key in our SONAR MEASURES data table.

In our newly generated clean technical debt dataset, we have some fluctuation of data before and after cleansing. The previous and current number of sonar analysis, sonar issues and sonar measures are shown in table 5.1. We have so far successfully generated 67k+ clean sonar analysis dataset, 1M+ clean sonar issues dataset and 66k+ clean sonar measures dataset.

Table 5.1. *Comparison of data before and after cleansing*

Project	Old analysis number	New analysis number	Old issues number	New issues number	Old measure number	New measures number
batik	2164	2178	29167	29167	2164	2164
commons-bcel	1325	1324	8796	8796	1325	1324
commons-beanutils	1211	1210	5544	5544	1211	1210
cocoon	10210	10221	77314	77313	10210	10206
commons-codec	1731	1731	2340	2340	1731	1730
commons-collections	2982	2816	11777	10958	2982	2765
commons-cli	896	847	37869	34269	896	847
commons-exec	617	617	1009	1009	617	617
commons-fileupload	922	865	1088	678	922	864
commons-io	2124	1910	6266	4000	2124	1910
commons-jelly	1939	1938	8841	2917	1939	1937
commons-jexl	1581	1533	35615	35344	1581	1532
commons-configuration	2930	2930	7513	7513	2930	2929
commons-daemon	982	982	796	796	982	981
commons-dbcp	1912	1556	7191	2867	1912	1555
commons-dbutils	645	603	868	627	645	602
commons-digester	2145	2143	8574	3890	2145	2142
felix	3490	3490	49640	49640	3490	3490
httpcomponents-client	2905	2714	11123	9758	2905	2714
httpcomponents-core	1941	2110	9531	9069	1941	1901
commons-jxpath	597	597	5826	5826	597	596
commons-net	2095	2090	41635	41409	2095	2089
commons-ognl	608	608	5838	5838	608	608
santuario	2719	2720	51229	51229	2719	2718
commons-validator	1341	1340	2574	2574	1341	1340
commons-vfs	2080	2080	4167	4167	2080	2079
zookeeper	424	608	5836	5028	424	222
archiva	4677	4693	75668	75668	4677	4677
cayenne	1252	1259	22011	22008	1252	1251
hive	5768	5836	509496	509496	5768	5767
thrift	2950	2001	5226	4876	2950	1944
Total:		67550		1024614		66711

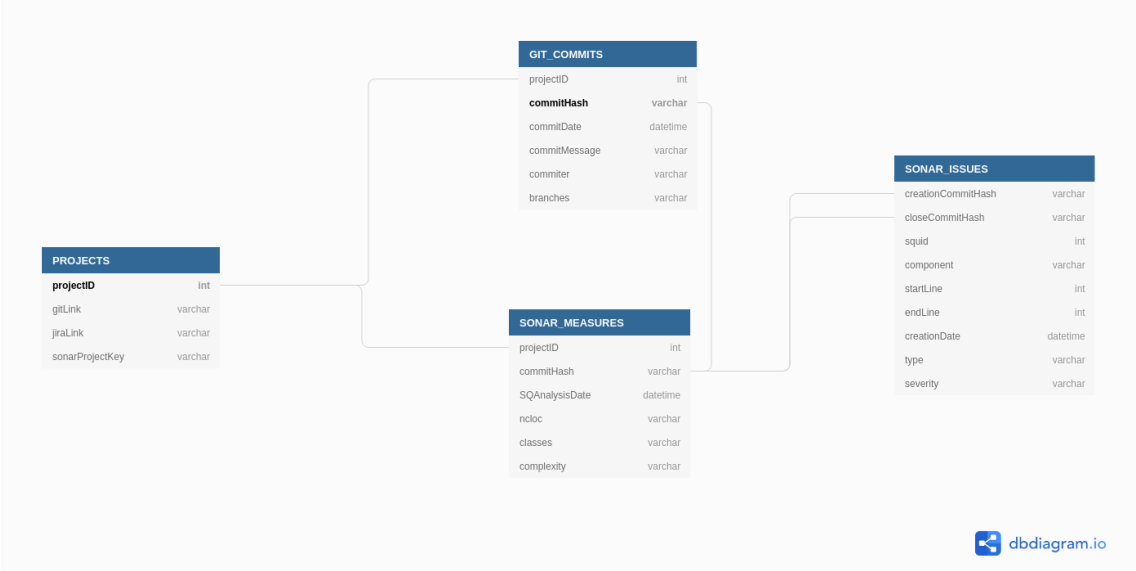

Figure 5.2. *Entity relationship diagram of old dataset*

6 DISCUSSION

To technical debt, companies always try to take careful step. Most of the time, it is hard to catch are technical debt. That is why companies approaches using technical debt measurement tool to catch technical debt in their source code at the early stage of the development. TD Measurement tool are gaining popularity among the companies. SonarQube is being used by more than 100 users worldwide. Information about analysis, metrics measures and issues generated by SonarQube on project are very important to developers as well as researchers. Not all companies make their source code dataset of technical debt publicly available. Also, make a good number of TD data in a dataset is a time consuming and complex task as it requires patience and other external tool to fetch the TD data and analyze it. This is why a clean common TD dataset with good number of data is somewhat researchers looking for who are interested to do more analysis on TD data.

The dataset we have generated after several layer of approaches is clean data set and has good number of data in it. With this amount of clean data, this dataset can help in data engineering or artificial engineering area. The aim of this clean TD dataset is same as the previous The Technical Debt Dataset [3]. Researchers can check different type of TD issues and can find out the effects of those issues on the project, product development and company as well.

This clean TD dataset can be used to find different types of research questions related to project development. Some of the possible queries are:

- Can it be possible to find a commit which creates technical debt from the technical debt data?
- Which are the technical debts developers create most?
- Does a technical debt issue cause the evolution of other issue?
- Should all the faults be treated as technical debt?
- How a company handle the technical debt issue refactoring?
- Does a technical debt measurement tool find out proper fault in the code?

7 CONCLUSION

The aim of this work was to generate a clean Technical Debt dataset. To do that, we extrated Technical debt data as sonar analysis, sonar issues and sonar measures of selected projects from SonarQube server using specific API. We also extracted the commit logs of those selected project using PyDriller [4]. We compared commits and sonar analysis. Then we added missing analysis event from sonar issues into sonar analysis data. We removed all the analysis from dataset which do not have commit hash data. Then we did matching sonar issues with sonar analysis. We applied different method, and on each method the matching results were getting better. We successfully cleaned redundant data from sonar issues and made perfect match with sonar analysis. Finally we made two version of dataset, CSV and SQLite so that others can run query and get data in the easiest way. This whole work took around 100 days, so other people who will work on this dataset will not have to wait 100 days to generate a dataset by him/herself.

This dataset will help researchers in different ways. A clean technical debt dataset with this amount of data is not that much available today. The data is saved and the relation in between different entities makes this dataset easier to perferm query and get the desired data. Researchers can perform different type of studies as per their interest and compare their own result.

REFERENCES

[1] P. Avgeriou, P. Kruchten, I. Ozkaya and C. Seaman. Managing Technical Debt in Software Engineering (Dagstuhl Seminar 16162). *Dagstuhl Reports* 6.4 (2016). Ed. by P. Avgeriou, P. Kruchten, I. Ozkaya and C. Seaman, 110–138. ISSN: 2192-5283. DOI: 10.4230/DagRep.6.4.110. URL: http://drops.dagstuhl.de/opus/volltexte/2016/6693.

[2] W. Cunningham. The WyCash portfolio management system. *SIGPLAN OOPS Messenger* 4.2 (1993), 29–30.

[3] V. Lenarduzzi, N. Saarimäki and D. Taibi. The Technical Debt Dataset. *Int. Conf. on Predictive Models and Data Analytics in software engineering (PROMISE'19)*. Sept. 2019.

[4] D. Spadini, M. Aniche and A. Bacchelli. PyDriller: Python framework for mining software repositories. Oct. 2018, 908–911. DOI: 10.1145/3236024.3264598.