Carnegie Mellon University
Software Engineering Institute

TECHNICAL DEBT ANALYSIS METHOD

Stephany Bellomo, James Ivers, Robert Nord, Ipek Ozkaya, Christopher Seifried

October 2018

Introduction

As the capabilities of static code analyzers expand, many organizations are looking into incorporating static code analysis tools in their development activities to perform quality checks. The challenge for development teams in using static analysis tools is making sense of the results to prioritize what to fix and what to ignore. Our goal is to extract potential design issues from the results of static code analysis. Design issues have previously been shown to constitute hardest to resolve technical debt. To extract these issues, we supplement the results from static code analysis with information about the design characteristics of the violations and combine this with data based on co-changing file relationships. Files that change at the same time during development could provide clues about underlying design issues. By combining the two data sets, the developer is able to view groups of files with possible design issues and design issues that are endemic throughout the codebase. Using this approach, we have created a pipeline that identifies the most significant design issues that contribute to technical debt, giving the developer a clearer picture of what should be fixed. This document serves as a procedure for using the pipeline to find technical debt items (TDIs).

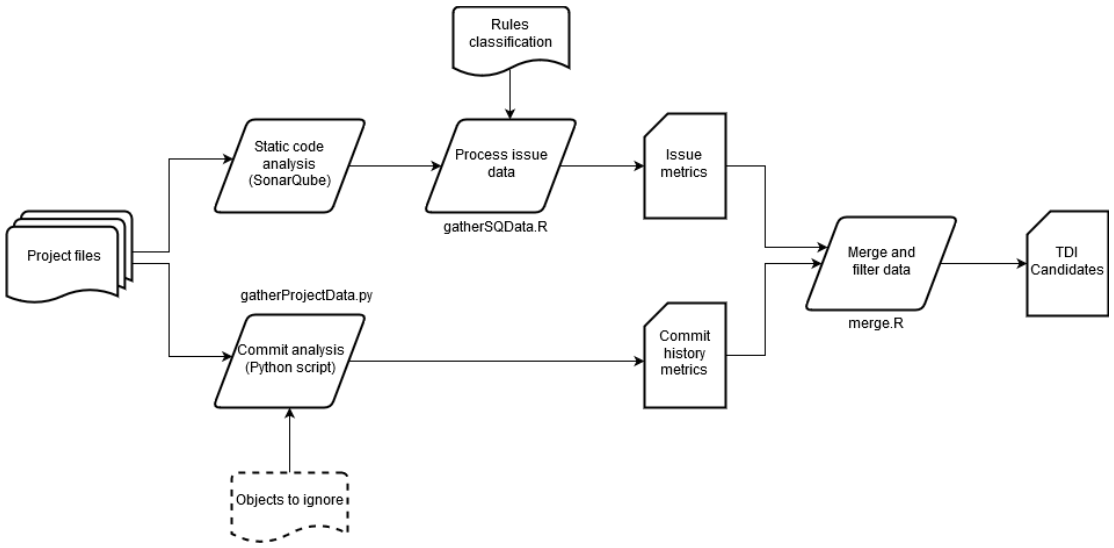

Figure 1: The flow of data through the analysis pipeline

Prerequisites

SonarQube

SonarQube can be installed on bare metal or with Docker. See Installing the Server for installing directly onto a machine.

To install and run SonarQube with Docker, run the following commands:

```
$ sudo docker pull sonarqube:6.7.4
$ sudo docker run -d --name sonarqube -p 9000:9000 -p 9092:9092 -p 8000:8000
sonarqube:6.7.4
```

This creates a SonarQube server that is accessible via port 9000. If behind a proxy, see the Docker documentation for instructions on how to properly configure proxy settings. Note that 6.7.4 is the most recent Long Term Supported (LTS) version of SonarQube. Also, the database that comes with the Docker image is not suited for production and may require more configuration (see the Docker image page).

Within SonarQube, install the plugins that provide support for the languages used in the project to be scanned. These can be found under **Administration → Marketplace** in the SonarQube web interface. Note that some plugins are free while others are only available in the commercial versions of SonarQube.

Maven 3+

We will demonstrate the SonarQube scanner using Maven. This requires Maven to know where the SonarQube server is located. Edit the *settings.xml* file for Maven, usually found in the *~/.m2* folder, and add the following:

```
<settings>
    <pluginGroups>
        <pluginGroup>org.sonarsource.scanner.maven</pluginGroup>
    </pluginGroups>
    <profiles>
        <profile>
            <id>sonar</id>
            <activation>
                <activeByDefault>true</activeByDefault>
            </activation>
            <properties>
                 <!-- Optional URL to server; default: http://localhost:9000 -->
                <sonar.host.url>
                  http://myserver:9000
                </sonar.host.url>
            </properties>
```

```
        </profile>
      </profiles>
</settings>
```

R Language and Packages

Make sure R is installed, as well as the tidyverse, writexl, and optparse packages. Run the following in an R command line to download and install the packages:

```
install.packages(c("tidyverse", "writexl", "optparse"))
```

(This may throw errors, so check the output. Packages often require some system libraries to be present first.)

Note that your library path must be writable, or else R cannot install packages. The scripts have been tested with R version 3.5.0. If you want to edit the R scripts, consider downloading and installing the RStudio IDE as well.

Python 3 and Packages

Ensure that Python 3.6.5+ is installed, as well as the xlsxwriter, wordcloud, argparse, and datetime packages. The packages can be installed using this shell command:

```
$ pip3 install xlsxwriter wordcloud argparse datetime
```

CLOC

Cloc is a tool that counts lines of code in a project. Install for use with the commit analysis script.

Technical Debt Package

The *td-analysis.zip* package contains all of the files required for the analysis. Extract the package anywhere on the system. Note that the *rule-classification.xlsx* file lists only Java rules. However, more rules can be added to this file to support new Java rules and new languages.

Violation Analysis

Scan with SonarQube

SonarQube provides several ways to invoke the analyzer using build automation tools. Here we will show how to use Maven to analyze a Java-based project. This requires Maven version 3+ and Java 8+. See Analyzing Source Code for an overview of the analysis process. SonarQube can hook into other build automation tools as well, or can be invoked using a command-line tool (see Scanning).

Run the Maven build in the root directory of the project. This will build and scan the project:

```
$ mvn clean sonar:sonar
```

See Analyzing with Scanner for Maven for more details.

Generate Violation Data

Now that the project has been scanned, *gatherSQData.R* can retrieve the violations from SonarQube, and organize them into multiple useful views that are written to *viol-data.xlsx*.

Element	Item	Description
Input	rules-classification.xlsx	The list of SonarQube rules classified as design rule (DR) or not design rule (ND)
Function	gatherSQData.R	Fetches and aggregates SonarQube violation data
Output	viol-data.xlsx	Labelled violations and simple views of the data

Enter the *viol-analysis* folder and run the following from a shell:

```
$ Rscript --vanilla gatherSQData.R -p my:project -s http://server:port/api/
```

where *my:project* is the SonarQube project key and *server:port* is the SonarQube server URL. Use the *--help* switch to see the help message.

Commit Analysis

Make Ignore File (optional)

The user may create a file called *.ignore*, which contains a list objects such as source files, directories, and commits to exclude from the commit analysis. Objects are listed one per line, and wildcards are allowed. If it is created, this file will be used as input to the next step. The *.ignore* file may look like this:

```
*.txt
*.sh
*.xml
*/test/*
*/example/*
*/docs/*

[af3261f2]
[e234914a]
```

Generate Commit History Data

The *gatherProjectData.py* script aggregates several metrics such as source lines of code (SLOC), number of commits, and coupling for each file. The script generates a list of files with associated metrics as well as a list of file pairings, where each pair is coupled.

Element	Item	Description
Input	.ignore	(Optional) File with list of objects to ignore in the commit analysis
Function	gatherProjectData.py	Aggregates project-wide commit history data
Output	commit-data.xlsx	Multiple views of the commit history data

In the *commit-analysis* folder, run the following in a shell:

```
$ python3 gatherProjectData.py <project> -i .ignore
```

where <*project*> is the path to the Git repository of the project. Use the *--help* switch to see the help message. Note that it is not uncommon for warnings to occur. The warnings can usually be ignored and the analysis will continue.

Merging the Data

The last step in the process utilizes *merge.R*, which consolidates the violation and commit history data and applies filtering criteria to generate TDI candidates.

Element	Item	Description
Input	viol-data.xlsx	Labelled violations and simple views of the data
	commit-data.xlsx	Multiple views of the commit history data
Function	merge.R	Combines violation data with the commit history data
Output	merged.xlsx	Combined views of the data and TDI candidates

Go to the folder where *merge.R* is located and run the following command in a shell:

```
$ Rscript --vanilla merge.R
```

Provide the *--help* switch to see the help message, which lists several options to configure input, output, and filtering criteria.

Appendix: Output Files and Sheets

viol-data.xlsx

Sheet	Description
Raw cleansed data	All violation data pulled directly from SonarQube
FBF-Total Viols	Files sorted by total violations (DR and ND)
FBF-DR Viols	Files sorted by number of DR violations
FBF-ND Viols	Files sorted by number of ND violations
FBF-DR Unique	File with the most unique rule violations, with each rule listed
FBF-Directories	Total violations in each top-level subdirectory
Comp Viol Counts	Violations listed by rule for each file
RBF-All Viols	DR and ND rules sorted by number of violations
RBF-DR Viols	DR rules sorted by number of violations
RBF-Paradigm	Design paradigms sorted by number of violations
New Rules	Rules that were violated, but are not listed in rule-classification.xlsx

commit-data.xlsx

Sheet	Description
Demographics	General data about the project and parameters used in the commit history analysis
By File	Raw Git log data by file
By Year	Lines of code (LOC) and authorship data organized by year
Couplings	Pairs of files that meet the minimum coupling parameter set by the user (default = 25%)
Hotspots	Ten top files as a result of considering LOC, complexity, number of authors, number of commits, and sum of coupling (each value relative to the maximum found in the project)

merged.xlsx

Sheet	Description
Raw File Data	Same as “By File” sheet in commit-data.xlsx
Files	Unfiltered list of files with violation and commit data merged
Couplings	Unfiltered list of couplings with violation and commit data merged
Filtered Files	Filtered list of files based on criteria
Filtered Couplings	Filtered list of couplings based on criteria

Contact Us

Software Engineering Institute
4500 Fifth Avenue, Pittsburgh, PA 15213-2612

Phone: 412/268.5800 | 888.201.4479
Web: www.sei.cmu.edu
Email: info@sei.cmu.edu

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM18-1419