TIME EFFICIENT AUTOMATED CODE QUALITY OPTIMIZATION

Guided by Dr. Babu Karupaiah A, Professor /ECE ,VCET

Gopi Krishnan P.R, Surya Kumar N.S, Srinivas A

Electronics and Communication Engineering, Velammal College of Engineering and Technology, Madurai

Abstract- Developers working with hard deadlines to deliver the required functionality to the customer. It is so important for developers that many times they compromise with the code quality, potential bugs, code duplications, and bad distribution of complexity. Additionally, they tend to leave unused variables, methods, etc. In this scenario, the code would work in the desired way. To avoid these issues in code, developers should always follow the good coding practice, but sometimes it is not possible to follow the rules and maintain the good quality as there may be many reasons. In order to achieve continuous code integration and deployment, developers need a tool that not only works once to check and tell them the problems in the code but also to track and control the code to check continuous code quality. To satisfy all these requirements, here comes SonarQube in the picture. The purpose of this project is to automate the SonarQube scanning process, reducing the time consumption of the process using SonarQube tool, GitLab and Java programming.

INTRODUCTION

Code quality metrics defines code that is good (high quality) — and code that is bad (low quality). While code quality is measured differently by every development team, in general it comes down to a subjective measure of common factors like maintainability, testability, readability, security, and more. Some automated tools can also analyze source code and provide a code quality score based on a number of metrics that measure complexity and functionality.

Why is code quality important for developers?

Code quality is crucial for developers because poorly written code can lead to technical debt and security issues. Technical debt refers to the idea that bugs and code deficiencies can accrue over time to the point where the cost of new code changes becomes much greater. By making short-term sacrifices in code quality, it becomes more difficult to deliver quality software later on.

Automated Code Quality Tool:

The Automated Code Quality that we used here is SonarQube.

Eventhough SonarQube automatically analyses the code quality of our source code, the time it takes for analyzing is high. So, we came up with a way to reduce the time taken for the analysis without reducing the quality of the analysis.

SONARQUBE

SONARQUBE is an open-source tool used for continuous code quality and inspection which aid developers in static code analysis.

It empowers all developers to write cleaner and safer code.

It contains thousands of automated Static Code Analysis rules to help us with catch tricky bugs to prevent undefined behaviour from impacting end-users and Fix vulnerabilities that compromise your app, and learn AppSec along the way with Security Hotspots.

SonarQube is written in java but it can analyze and manage code of 29 programming languages, including Java, HTML, JS etc by default and also supports C/C++, PL/SQL, Cobol etc., through plugins. Plugins extend the functionality of SonarQube. More than 50 plugins are available.

SonarQube is maintained by SonarSource.

It also makes sure your codebase is clean and maintainable, to increase developer velocity.

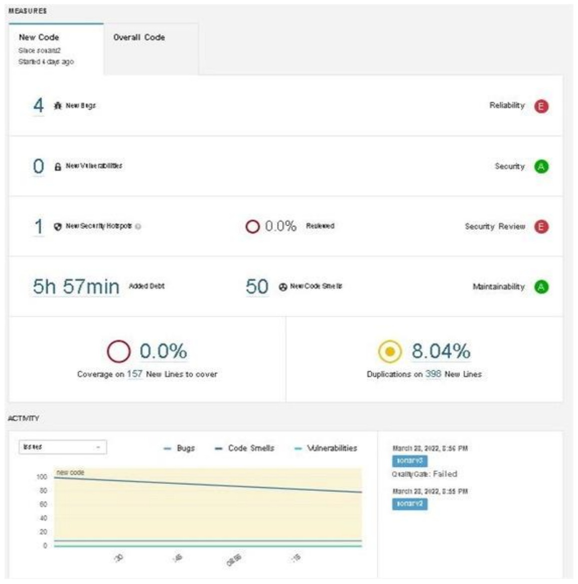
Sonar covers the 7 sections of code quality

Architecture and Design

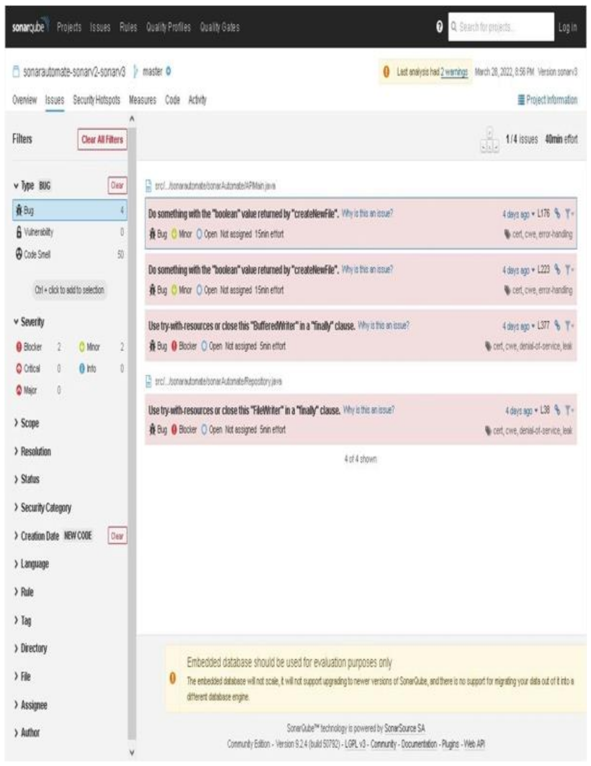
- Unit tests
- Duplicated code
- Potential bugs
- Complex code
- Coding standards
- Comments

SonarQube receives files as an input and analyzes them along with barriers. Then calculates a set of metrics, stores them in a database and shows them on a dashboard. This recursive implementation helps in analysis of code quality and how code improves over time.

The following image shows the dashboard of SonarQube which contains the number of bugs, vulnerabilities, security hotspots, code spells of a project, you can navigate to the issue by clicking on them.



The following image shows the issues of a project, clicking on them navigates you to the code where the issue is found.


sonarqube
Projects
Issues
Rules
Quality Profiles
Quality Gates
Log in
Overview
Issues
Security Hotspots
Measures
Code
Activity
Filters
Clear All Filters
Type
Bug
Vulnerability
Code Smell
Severity
Blocker
Minor
Critical
Info
Major
Scope
Resolution
Status
Security Category
Creation Date
Language
Rule
Tag
Directory
File
Assignee
Author
Do something with the "boolean" value returned by "createNewFile".
Do something with the "boolean" value returned by "createNewFile".
Use try-with-resources or close this "BufferedWriter" in a "finally" clause.
Use try-with-resources or close this "FileWriter" in a "finally" clause.
4 of 4 shown
Embedded database should be used for evaluation purposes only

New Code Quality is what we need

For a developer working in a particular part of a software product, it is enough for him to check the code quality of the newly developed code alone. For example, if a developer makes a branch from base branch and adds or makes some changes to it, then it is enough for him to check the quality of that newly modified code only.

Importance of Work

Modern Technology teams operate in a fast-paced environment. Time management in software design has two components. One of those is coding, and the other is project management. So, it becomes important to speed up things to get things earlier and be free from the stress of project manager.

So to further increase the efficiency of code quality analysis process and to reduce the time taken for it, we built a framework for SonarQube.

Normally, To get the code quality of developed/newly added code. You have to analyse the base branch and then analyse the developer branch on top of that. SonarQube recognizes the new changes and shows the code quality of those new changes alone(overall project code quality is displayed seperately) in the dashboard.

Eventhough it recognizes the newly added code in the developer branch, it requires to scan the entire contents of the branch again to get those code quality which makes the normal process time consuming and repetitive.

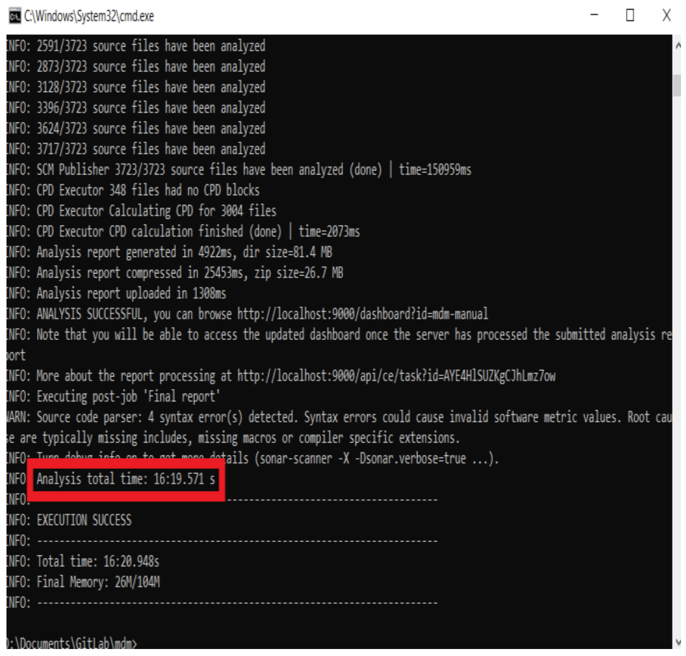
So we thought of a way to make a list modified files between the base and developer branch and to scan those files alone. This will reduce the time consumed in normal method.

Meanwhile we also programmatically automated the manual process done.

An example of how much time that can be saved is shown below.

In Existing Method:

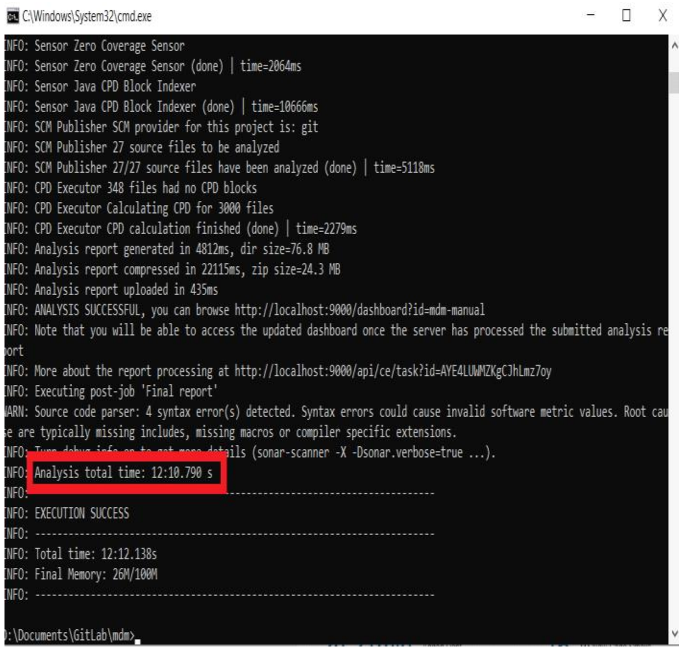
Time Taken to scan 1st branch:



```
C:\Windows\System32\cmd.exe
INFO: 2591/3723 source files have been analyzed
INFO: 2873/3723 source files have been analyzed
INFO: 3128/3723 source files have been analyzed
INFO: 3396/3723 source files have been analyzed
INFO: 3624/3723 source files have been analyzed
INFO: 3717/3723 source files have been analyzed
INFO: SCM Publisher 3723/3723 source files have been analyzed (done) | time=150959ms
INFO: CPD Executor 348 files had no CPD blocks
INFO: CPD Executor Calculating CPD for 3004 files
INFO: CPD Executor CPD calculation finished (done) | time=2073ms
INFO: Analysis report generated in 4922ms, dir size=81.4 MB
INFO: Analysis report compressed in 25453ms, zip size=26.7 MB
INFO: Analysis report uploaded in 1308ms
INFO: ANALYSIS SUCCESSFUL, you can browse http://localhost:9000/dashboard?id=mdm-manual
INFO: Note that you will be able to access the updated dashboard once the server has processed the submitted analysis re
port
INFO: More about the report processing at http://localhost:9000/api/ce/task?id=AYE4HlSUZKgCJhLmz7ow
INFO: Executing post-job 'Final report'
WARN: Source code parser: 4 syntax error(s) detected. Syntax errors could cause invalid software metric values. Root cau
se are typically missing includes, missing macros or compiler specific extensions.
INFO: Turn debug info on to get more details (sonar-scanner -X -Dsonar.verbose=true ...).
INFO: Analysis total time: 16:19.571 s
INFO: ------------------------------------------------------------------------
INFO: EXECUTION SUCCESS
INFO: ------------------------------------------------------------------------
INFO: Total time: 16:20.948s
INFO: Final Memory: 26M/104M
INFO: ------------------------------------------------------------------------

D:\Documents\GitLab\mdm>
```

Time taken to scan 2nd branch :

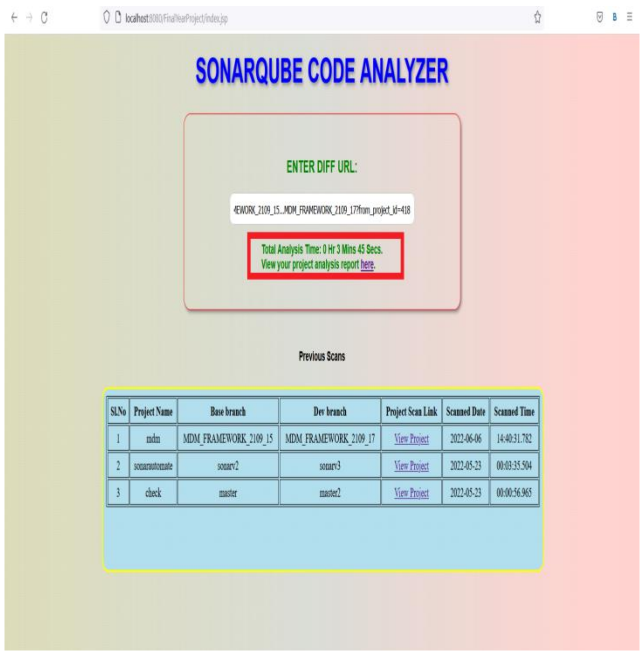


```
C:\Windows\System32\cmd.exe
INFO: Sensor Zero Coverage Sensor
INFO: Sensor Zero Coverage Sensor (done) | time=2064ms
INFO: Sensor Java CPD Block Indexer
INFO: Sensor Java CPD Block Indexer (done) | time=10666ms
INFO: SCM Publisher SCM provider for this project is: git
INFO: SCM Publisher 27 source files to be analyzed
INFO: SCM Publisher 27/27 source files have been analyzed (done) | time=5118ms
INFO: CPD Executor 348 files had no CPD blocks
INFO: CPD Executor Calculating CPD for 3000 files
INFO: CPD Executor CPD calculation finished (done) | time=2279ms
INFO: Analysis report generated in 4812ms, dir size=76.8 MB
INFO: Analysis report compressed in 22115ms, zip size=24.3 MB
INFO: Analysis report uploaded in 435ms
INFO: ANALYSIS SUCCESSFUL, you can browse http://localhost:9000/dashboard?id=mdm-manual
INFO: Note that you will be able to access the updated dashboard once the server has processed the submitted analysis re
port
INFO: More about the report processing at http://localhost:9000/api/ce/task?id=AYE4LUWMZKgCJhLmz7oy
INFO: Executing post-job 'Final report'
WARN: Source code parser: 4 syntax error(s) detected. Syntax errors could cause invalid software metric values. Root cau
se are typically missing includes, missing macros or compiler specific extensions.
INFO: Turn debug info on to get more details (sonar-scanner -X -Dsonar.verbose=true ...).
INFO: Analysis total time: 12:10.790 s
INFO: ------------------------------------------------------------------------
INFO: EXECUTION SUCCESS
INFO: ------------------------------------------------------------------------
INFO: Total time: 12:12.138s
INFO: Final Memory: 26M/100M
INFO: ------------------------------------------------------------------------

D:\Documents\GitLab\mdm>
```

Proposed Method:

Time taken in proposed method :



EWORK_2109_15...MDM_FRAMEWORK_2109_17?from_project_id=418

Total Analysis Time: 0 Hr 3 Mins 45 Secs.
View your project analysis report here.

Previous Scans

Sl.No	Project Name	Base branch	Dev branch	Project Scan Link	Scanned Date	Scanned Time
1	mdm	MDM_FRAMEWORK_2109_15	MDM_FRAMEWORK_2109_17	View Project	2022-06-06	14:40:31.782
2	sonarautomate	sonarv2	sonarv3	View Project	2022-05-23	00:03:35.504
3	check	master	master2	View Project	2022-05-23	00:00:56.965

So, as you can see in the above screenshots, the time taken is reduced from 28 mins 29 secs(16:19 + 12:10) to 3 mins 45 secs(including both base and developer branch scans) (saved ~86% of time). Thus, the time taken is greatly reduced by using the proposed method.

Working of proposed methodology

The existing method takes a lot of time (100MB code takes 10 to 15 mins to scan considering a decently performing machine) as it is scanning every files in the two branches of the repository.

To get the code quality of new code, it is enough to scan only the modified files between the two branches/versions.

But to achieve that, we need to check every files in both branch of the repo to see if there is a change and make a list of modified files and get those files from remote repository.

Manually checking all these files for similarity and download those one by one is not possible.

So, a web application to automate the normal process of scanning source code via SonarQube while also including the proposed methodology would be a apt solution.

The application takes a url from the compare page of the GitLab and gives a url of the scanned project after processing these steps programmatically.

The Application uses Java, Jersey and Tomcat(server) to achieve all these above mentioned steps.

How to use the application?

To get the required url,

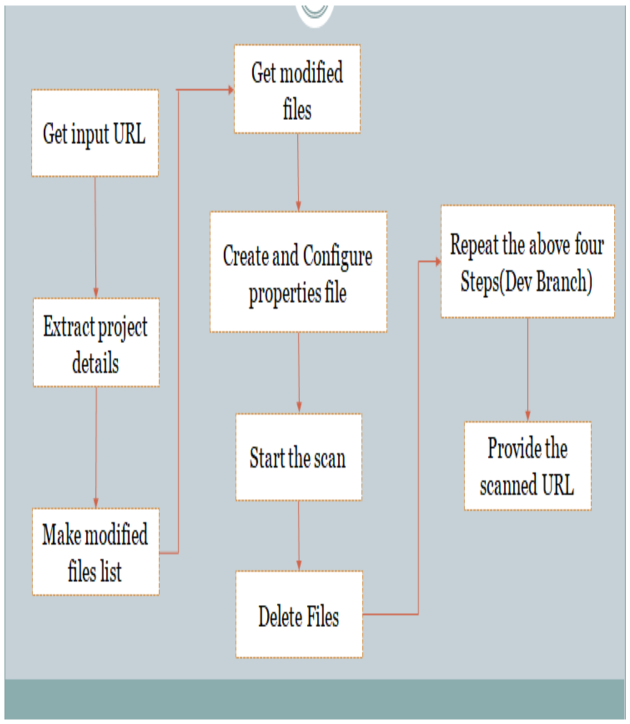
1. Go to the GitLab project HomePage, Choose 'Compare' under 'Repository' from the sidebar(Left Panel).

2. Then Choose your base branch/tag/commit-hash in the target dropdown and developer branch/tag/commit-hash in the source dropdown and hit 'Compare'.

3. After the result shows up in a few seconds, just copy the URL of that page and paste it in the web portal form.

4. You'll get the link to the analysis results in a few minutes.

General Layout



References

- *https://docs.sonarqube.org/latest/* - SonarQube

- *https://tomcat.apache.org/tomcat-9.0-doc/* - Apache Tomcat

- *https://eclipse-ee4j.github.io/jersey/* - Eclipse RESTful Web Services

- *https://docs.sonarqube.org/latest/analysis/scan/sonarscanner/* - SonarScanner

- *https://www.diva-portal.org/smash/get/diva2:947354/FULLTEXT03* - Static Code Analysis: A Systematic Literature Review and an Industrial Survey

- *https://www.postgresql.org/docs/* - PostgreSQL Documentation

- *https://eclipseee4j.github.io/jersey.github.io/documentation/latest/jaxrs-resources.html* - Jersey User Guide

- *https://tomcat.apache.org/tomcat-5.5doc/servletapi/index.html#:~:text=The%20javax.,by%20a%20conforming%20servlet%20container.&text=The%20javax.,-servlet.* - Servlet API Documentation