HELLENIC REPUBLIC
UNIVERSITY OF MACEDONIA

Code Quality and Hotspot Prioritization USING REPOSITORY MINING

Student: Archontis E. Kostis | Advisor: Alexander Chatzigeorgiou
Department of Applied Informatics

14/02/2024

TABLE OF CONTENTS

01 **INTRODUCTION AND OBJECTIVES**

Introduction of the thesis topic and objectives.

02 **THEORETICAL BACKGROUND**

Theoretical foundations related to software quality.

03 **EVALUATION TOOL DEMO**

Introduce CodeInspector, the evaluation tool developed.

04 **ARCHITECTURE & IMPLEMENTATION**

Details about the technical implementation of the developed tool

01

INTRODUCTION & OBJECTIVES

Software Development

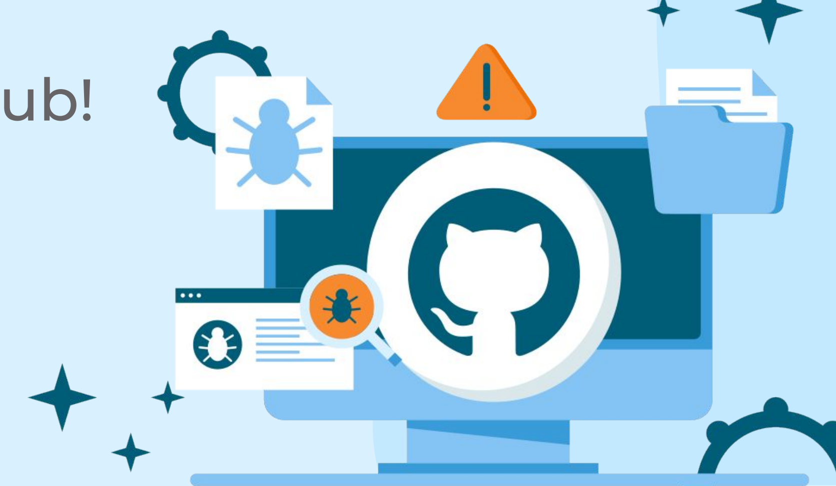
- The field is characterized by continuous modifications to produce more efficient, feature-rich software.
- Large-scale software systems are becoming the norm rather than the exception.
- As software systems get more complex, maintaining high-quality code and identifying hotspots is necessary.
- GitHub repositories are large information warehouses containing the whole history of a project's life cycle

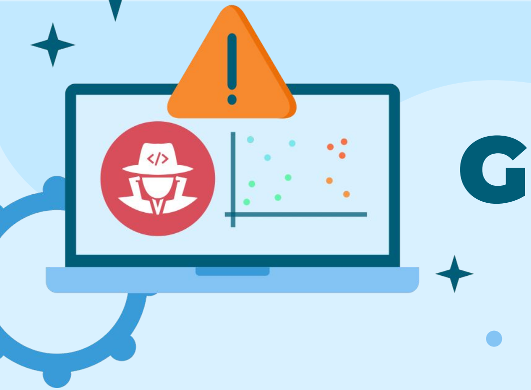
more than

300 million

repositories are hosted on Github!

Archived from the original on January 25, 2023. Retrieved October 5, 2020.





Github & Software Quality

- GitHub’s repository base opens up a world of possibilities for software engineering research.
- All these repositories hold a massive amount of data, including commit modifications, code reviews, conversations, and issue tracking.
- In software not all components are created equal and some classes or files tend to be more problematic than others.
- Identifying and prioritizing such units is necessary.
- Version Control Systems can help us find how many times a file has been modified (churn).

OBJECTIVES

1

CREATE A TOOL

2

MINE REPOS

3

PRIORITIZE HOTSPOTS

4

RATE COMMITS

02

THEORETICAL BACKGROUND

Software Quality

“Quality is **hard to define, impossible to measure, easy to recognize**”

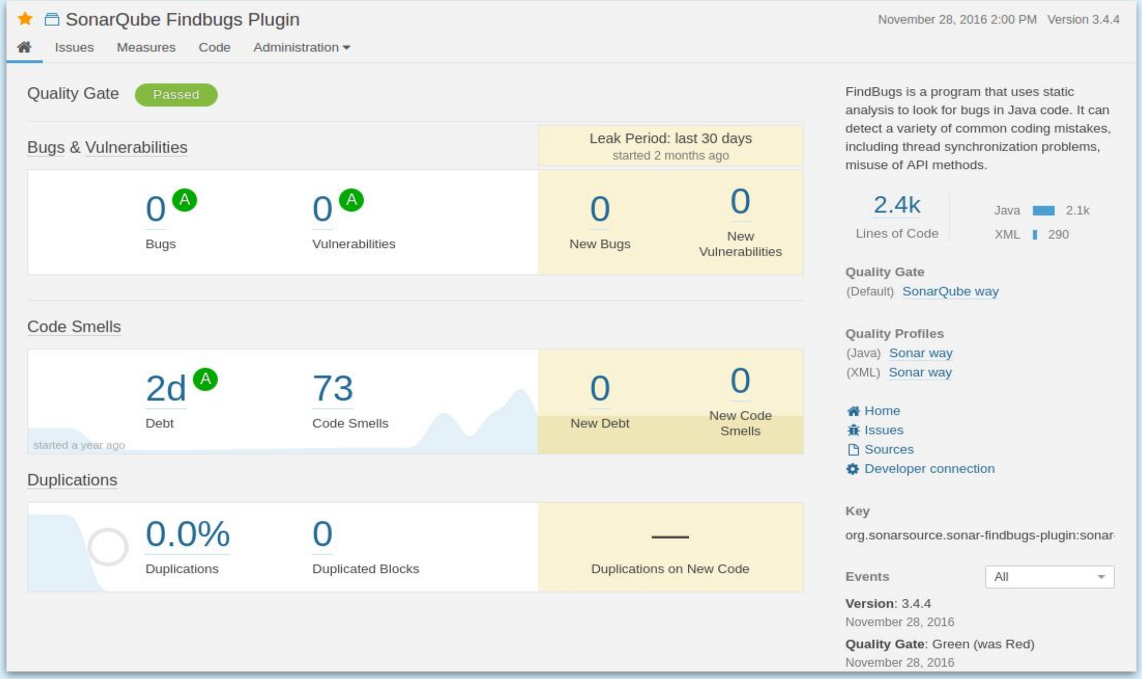
—KITCHENHAM

Software Quality

“Quality is generally **transparent when present,** but **easily recognized in its absence**”

—GILLES

Software Quality Tools



SonarQube Findbugs Plugin

November 28, 2016 2:00 PM Version 3.4.4

Quality Gate Passed

Bugs & Vulnerabilities

Leak Period: last 30 days
started 2 months ago

0 A Bugs

0 A Vulnerabilities

0 New Bugs

0 New Vulnerabilities

Code Smells

2d A Debt

73 Code Smells

started a year ago

0 New Debt

0 New Code Smells

Duplications

0.0% Duplications

0 Duplicated Blocks

— Duplications on New Code

FindBugs is a program that uses static analysis to look for bugs in Java code. It can detect a variety of common coding mistakes, including thread synchronization problems, misuse of API methods.

2.4k Lines of Code

Java 2.1k

XML 290

Quality Gate
(Default) SonarQube way

Quality Profiles
(Java) Sonar way
(XML) Sonar way

Key
org.sonarsource.sonar-findbugs-plugin:sonar-

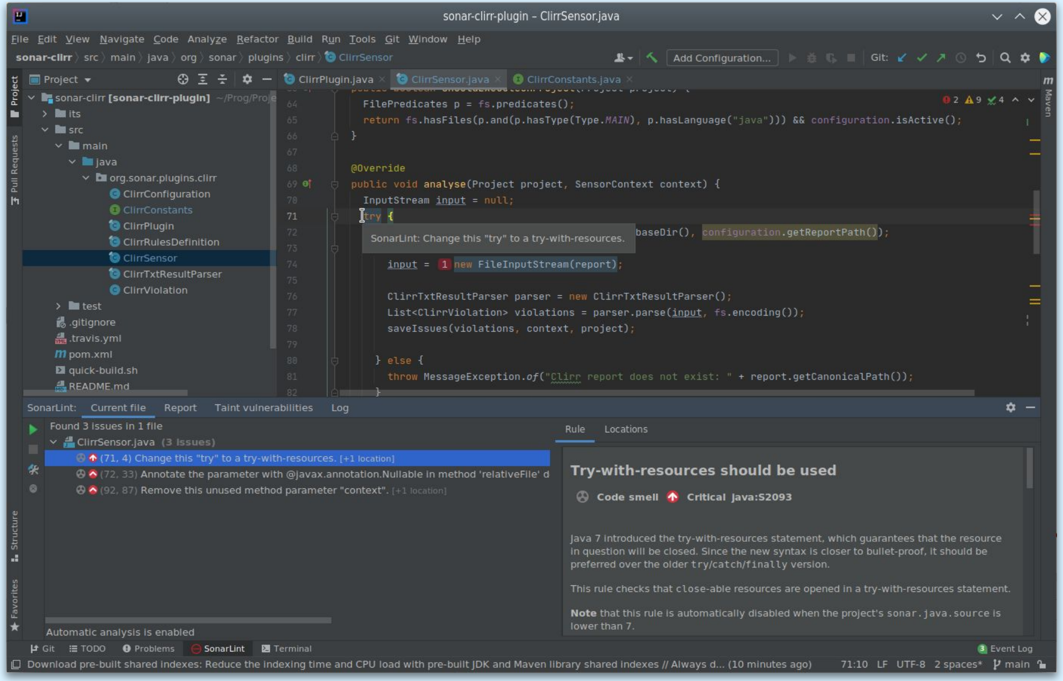
Events All

Version: 3.4.4
November 28, 2016

Quality Gate: Green (was Red)
November 28, 2016

SonarQube

Software Quality Tools



```
    FilePredicates p = fs.predicates();
    return fs.hasFiles(p.and(p.hasType(Type.MAIN), p.hasLanguage("java"))) && configuration.isActive();
  }

  @Override
  public void analyse(Project project, SensorContext context) {
    InputStream input = null;
    try {
                                                 baseDir(), configuration.getReportPath());

        input = 1 new FileInputStream(report);

        ClirrTxtResultParser parser = new ClirrTxtResultParser();
        List<ClirrViolation> violations = parser.parse(input, fs.encoding());
        saveIssues(violations, context, project);

      } else {
        throw MessageException.of("Clirr report does not exist: " + report.getCanonicalPath());
      }
```

SonarLint: Change this "try" to a try-with-resources.

Found 3 issues in 1 file

ClirrSensor.java (3 issues)

- (71, 4) Change this "try" to a try-with-resources. [+1 location]
- (72, 33) Annotate the parameter with @javax.annotation.Nullable in method 'relativeFile' d
- (92, 87) Remove this unused method parameter "context". [+1 location]

Automatic analysis is enabled

Try-with-resources should be used

Code smell Critical java:S2093

Java 7 introduced the try-with-resources statement, which guarantees that the resource in question will be closed. Since the new syntax is closer to bullet-proof, it should be preferred over the older try/catch/finally version.

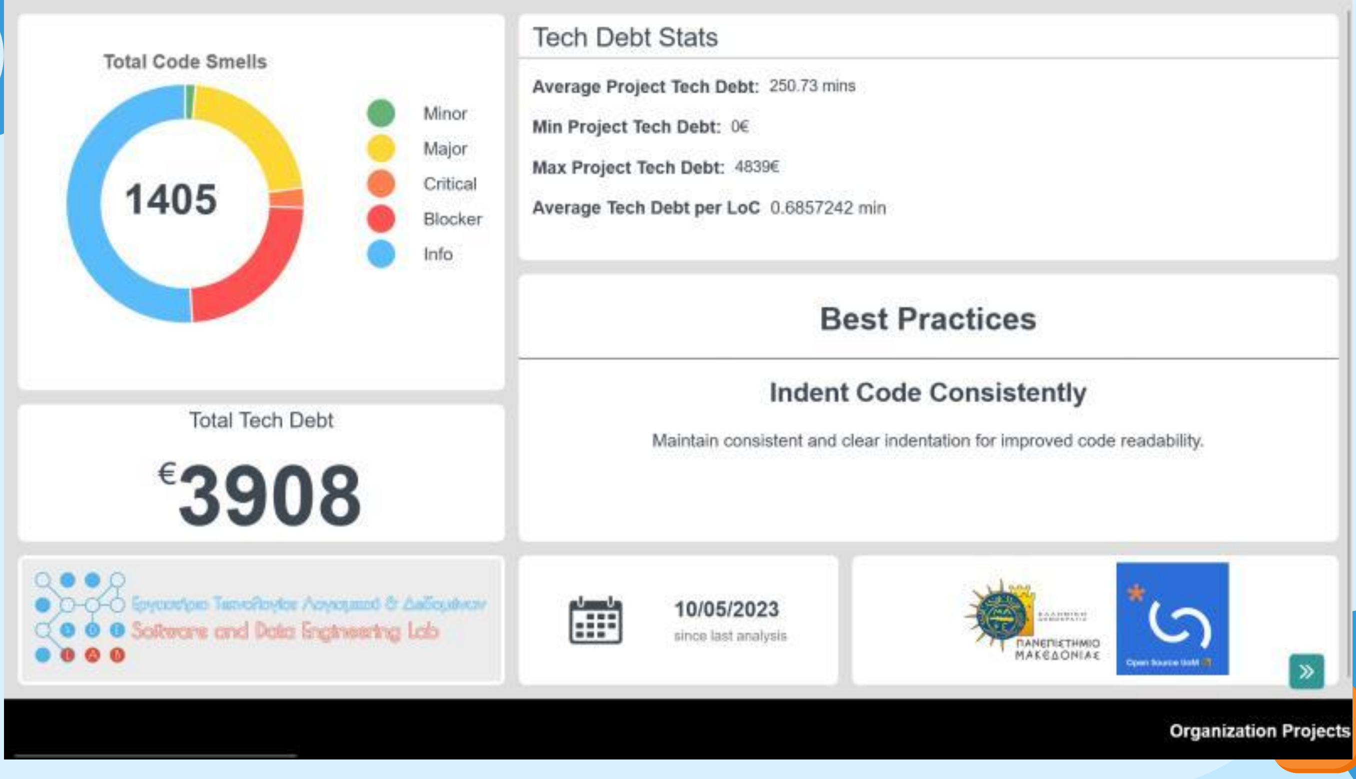
This rule checks that close-able resources are opened in a try-with-resources statement.

Note that this rule is automatically disabled when the project's sonar.java.source is lower than 7.

Download pre-built shared indexes: Reduce the indexing time and CPU load with pre-built JDK and Maven library shared indexes // Always d... (10 minutes ago) 71:10 LF UTF-8 2 spaces* main

SonarLint

Software Quality Tools



Total Code Smells

1405

Total Tech Debt

€3908

Tech Debt Stats

Average Project Tech Debt: 250.73 mins

Min Project Tech Debt: 0€

Max Project Tech Debt: 4839€

Average Tech Debt per LoC 0.6857242 min

Best Practices

Indent Code Consistently

Maintain consistent and clear indentation for improved code readability.

Εργαστήριο Τεχνολογίας Λογισμικού & Δεδομένων

Software and Data Engineering Lab

10/05/2023

since last analysis

Uom Quality Dashboard

Delta Maintainability Model

Delta Maintainability Model

- The Delta Maintainability Model (DMM) is a set of metrics that assess the maintainability of code changes in a software system.

- It was published by M. di Biase et al. in the 2019 IEEE/ACM International Conference on Technical Debt.

- The model is based on determining the impact of an individual code change on maintainability.

DMM Calculation

- It views changes as the addition or removal of lines of code to units and modules implicated in the change.
- A low DMM score indicates a large number of complex alterations. And all DMM values are between 0.0 and 1.0
- The model utilizes the SIG-MM system properties.
- Each property is defined with a specific description and criteria for qualifying code as low risk.

DMM Calculation

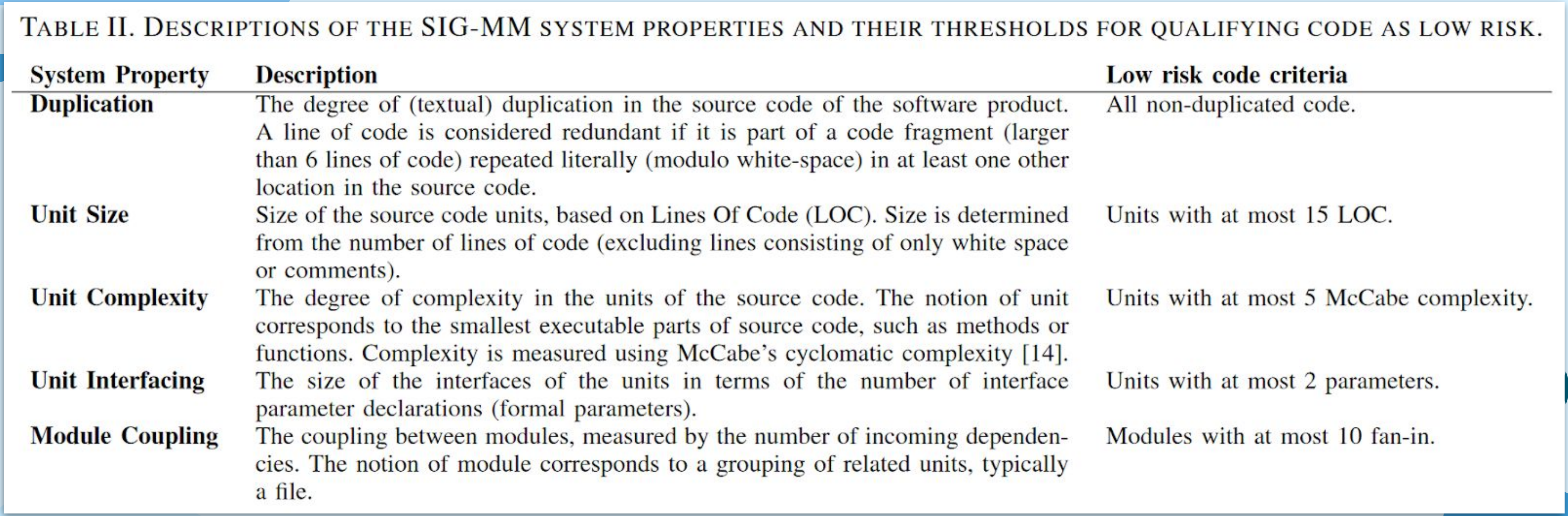


TABLE II. DESCRIPTIONS OF THE SIG-MM SYSTEM PROPERTIES AND THEIR THRESHOLDS FOR QUALIFYING CODE AS LOW RISK.

System Property	Description	Low risk code criteria
Duplication	The degree of (textual) duplication in the source code of the software product. A line of code is considered redundant if it is part of a code fragment (larger than 6 lines of code) repeated literally (modulo white-space) in at least one other location in the source code.	All non-duplicated code.
Unit Size	Size of the source code units, based on Lines Of Code (LOC). Size is determined from the number of lines of code (excluding lines consisting of only white space or comments).	Units with at most 15 LOC.
Unit Complexity	The degree of complexity in the units of the source code. The notion of unit corresponds to the smallest executable parts of source code, such as methods or functions. Complexity is measured using McCabe's cyclomatic complexity [14].	Units with at most 5 McCabe complexity.
Unit Interfacing	The size of the interfaces of the units in terms of the number of interface parameter declarations (formal parameters).	Units with at most 2 parameters.
Module Coupling	The coupling between modules, measured by the number of incoming dependencies. The notion of module corresponds to a grouping of related units, typically a file.	Modules with at most 10 fan-in.

Source: The Delta Maintainability Model: Measuring Maintainability of Fine-Grained Code Changes

DMM Calculation

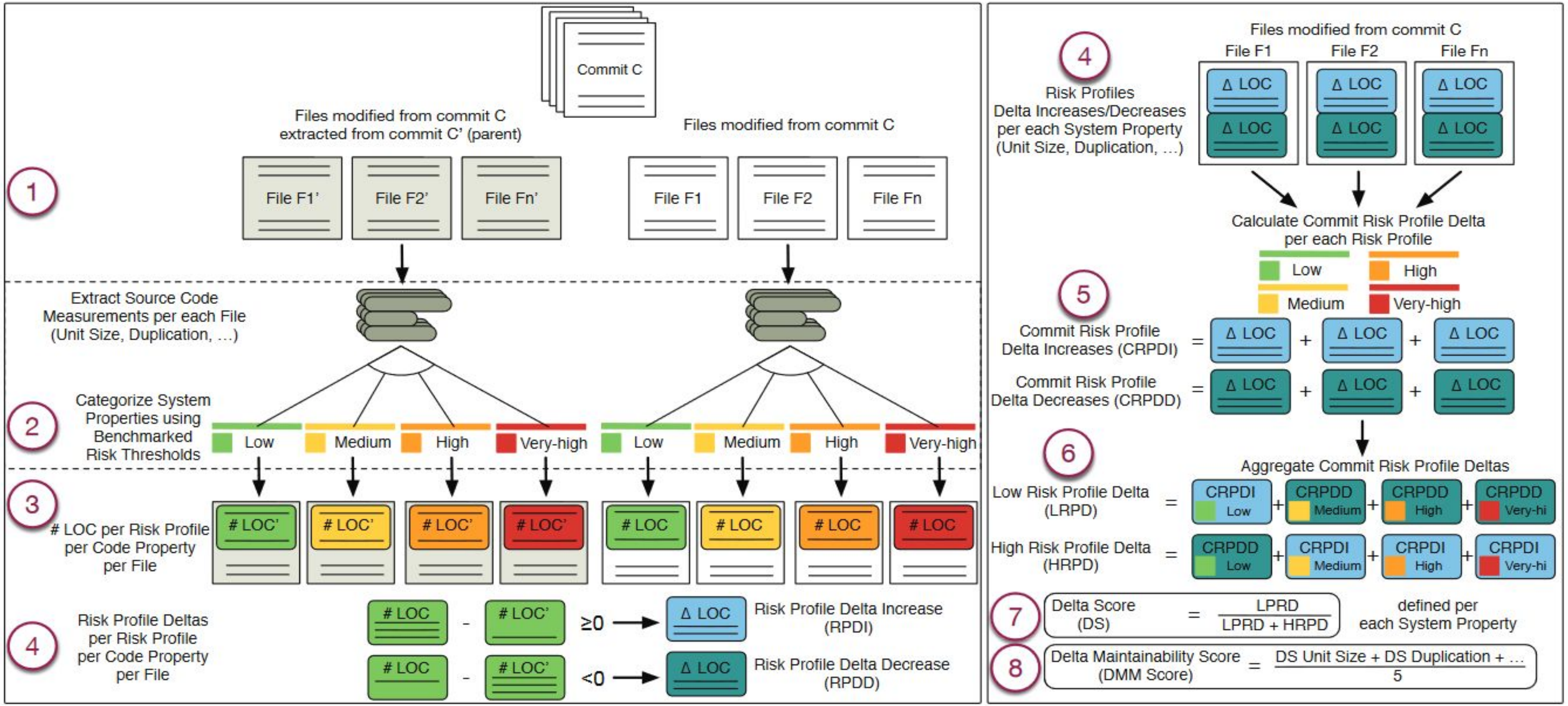
1. Risk Profile Mapping

How a code change translates
into a Risk Profile

2. DMM Score Generation

This level combines all Risk Profiles for a
code change to generate a DMM score

DMM Calculation


Commit C
Files modified from commit C extracted from commit C' (parent)
Files modified from commit C
1
File F1'
File F2'
File Fn'
File F1
File F2
File Fn
Extract Source Code Measurements per each File (Unit Size, Duplication, ...)
2
Categorize System Properties using Benchmarked Risk Thresholds
Low
Medium
High
Very-high
Low
Medium
High
Very-high
3
# LOC per Risk Profile per Code Property per File
# LOC'
# LOC'
# LOC'
# LOC'
# LOC
# LOC
# LOC
# LOC
4
Risk Profile Deltas per Risk Profile per Code Property per File
# LOC - # LOC' ≥0 → Δ LOC
Risk Profile Delta Increase (RPDI)
# LOC - # LOC' <0 → Δ LOC
Risk Profile Delta Decrease (RPDD)
4
Risk Profiles Delta Increases/Decreases per each System Property (Unit Size, Duplication, ...)
Files modified from commit C
File F1
File F2
File Fn
Δ LOC
Δ LOC
Δ LOC
Δ LOC
Δ LOC
Δ LOC
Calculate Commit Risk Profile Delta per each Risk Profile
Low
High
Medium
Very-high
5
Commit Risk Profile Delta Increases (CRPDI) = Δ LOC + Δ LOC + Δ LOC
Commit Risk Profile Delta Decreases (CRPDD) = Δ LOC + Δ LOC + Δ LOC
6
Aggregate Commit Risk Profile Deltas
Low Risk Profile Delta (LRPD) = CRPDI Low + CRPDD Medium + CRPDD High + CRPDD Very-hi
High Risk Profile Delta (HRPD) = CRPDD Low + CRPDI Medium + CRPDI High + CRPDI Very-hi
7
Delta Score (DS) = LPRD / (LPRD + HRPD) defined per each System Property
8
Delta Maintainability Score (DMM Score) = (DS Unit Size + DS Duplication + ...) / 5

(A) Level 1: risk profile deltas.

(B) Level 2: delta scores.

Source: The Delta Maintainability Model: Measuring Maintainability of Fine-Grained Code Changes

DMM in PyDriller

- PyDriller provides an implementation of the Open Source Delta Maintainability Model (OS-DMM) to assess the maintainability implications of commits.
- The OS-DMM implementation of PyDriller supports three commit-level metrics related to risk in size, complexity, and interfacing.
- It rewards making things better, and penalizes making them worse.
- DMM metrics have a value from 0.0 to 1.0

Hotspot Prioritization & Software Quality

The Eisenhower Matrix

- For our Hotspot Prioritization Technique we will implement an approach similar to the Eisenhower Matrix.

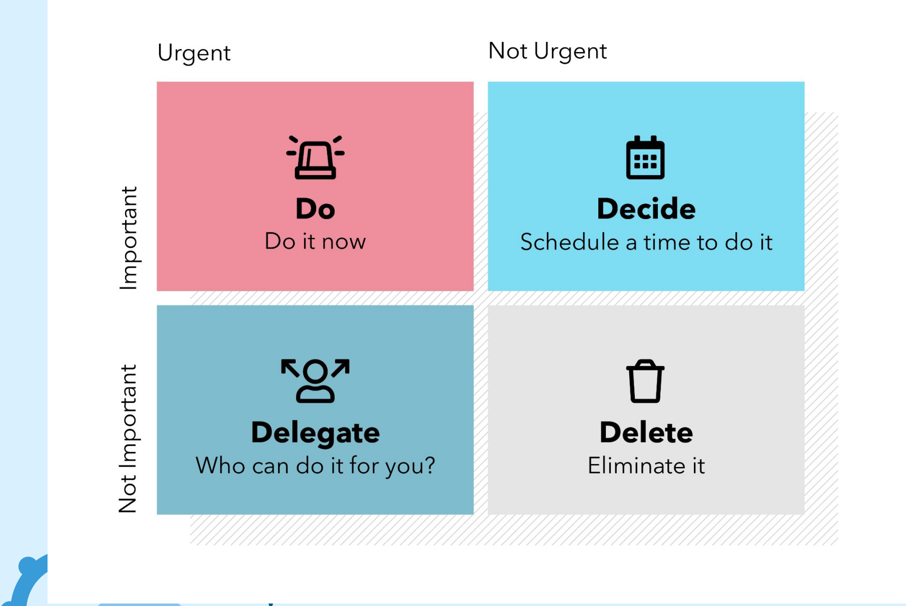
- The Eisenhower Matrix is mostly used in Project Management

- It is a time management model that categorizes tasks into four quadrants based on how urgent and important they are.

- In our implementation we prioritize hotspots using complexity and churn as the dimensions for categorizing hotspots.

The Eisenhower Matrix VS Our Approach

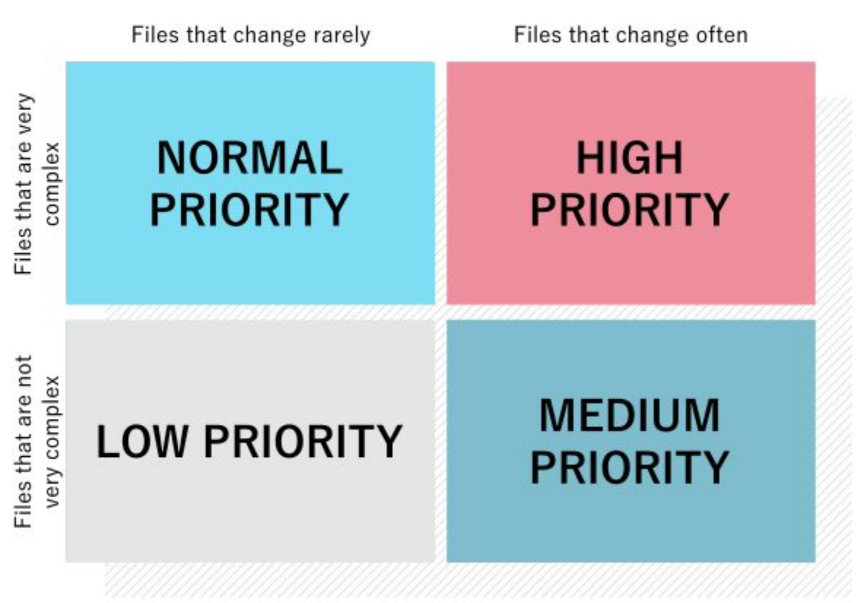
The Eisenhower Matrix



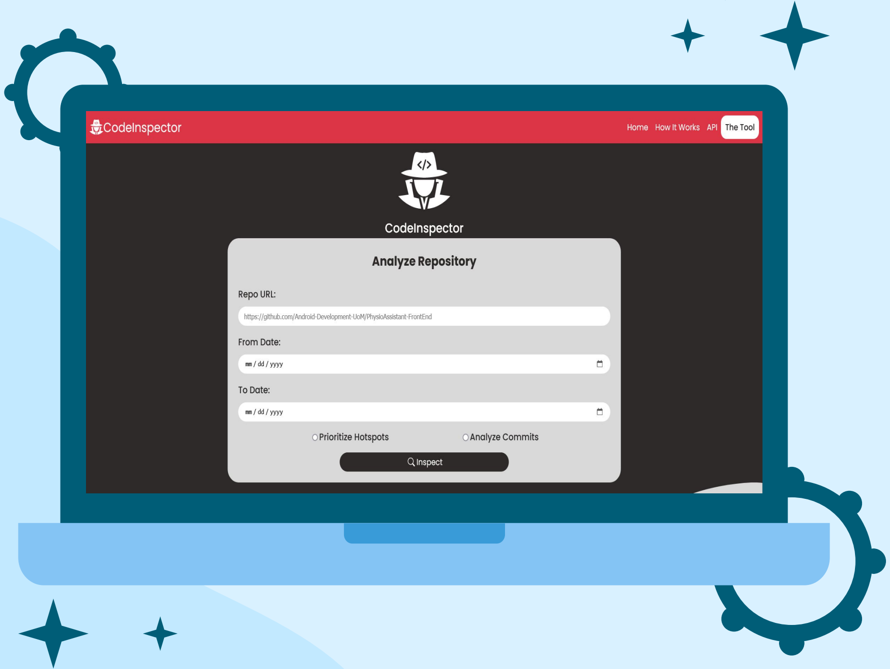
	Urgent	Not Urgent
Important	**Do** Do it now	**Decide** Schedule a time to do it
Not Important	**Delegate** Who can do it for you?	**Delete** Eliminate it

Source: The Eisenhower matrix – A popular prioritization framework

Our Approach



	Files that change rarely	Files that change often
Files that are very complex	NORMAL PRIORITY	HIGH PRIORITY
Files that are not very complex	LOW PRIORITY	MEDIUM PRIORITY



Analyze Repository

Repo URL:

https://github.com/Android-Development-UoM/PhysioAssistant-FrontEnd

From Date:

mm / dd / yyyy

To Date:

mm / dd / yyyy

○ Prioritize Hotspots ○ Analyze Commits

Inspect

03

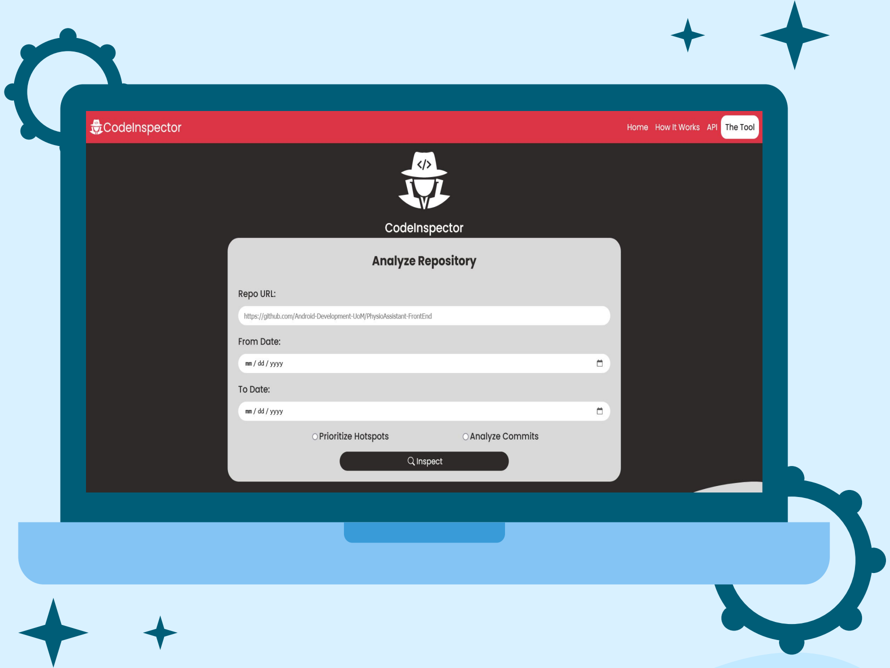
EVALUATION TOOL DEMO

04

ARCHITECTURE & IMPLEMENTATION

CodeInspector

CodeInspector is a full-stack application that provides software quality analysis based on hotspot prioritization and commits.



Analyze Repository

Repo URL:

https://github.com/Android-Development-UoM/PhysioAssistant-FrontEnd

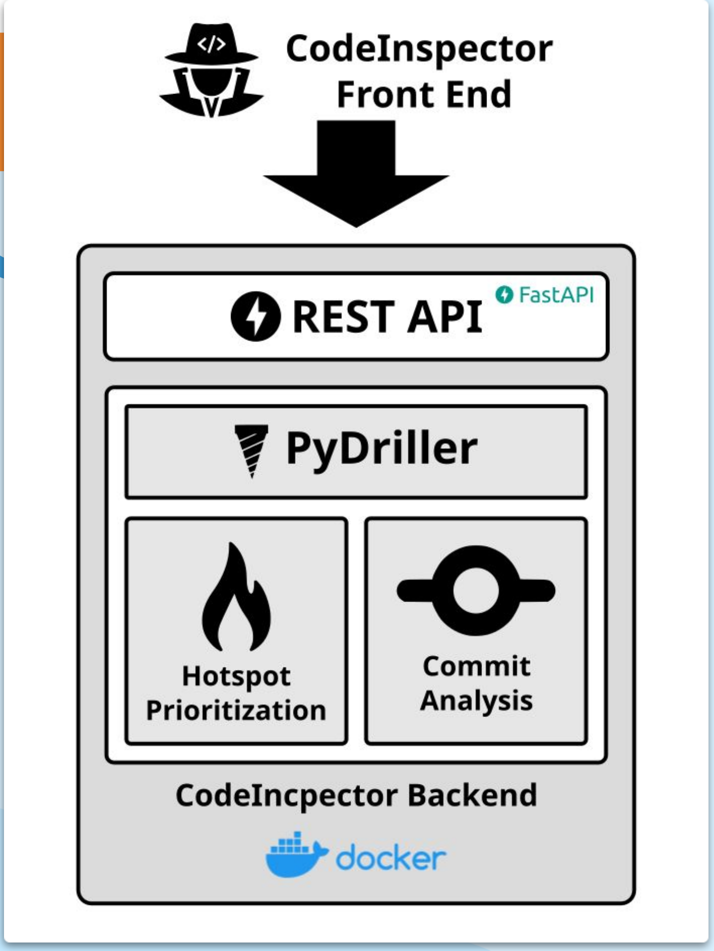
From Date:

mm / dd / yyyy

To Date:

mm / dd / yyyy

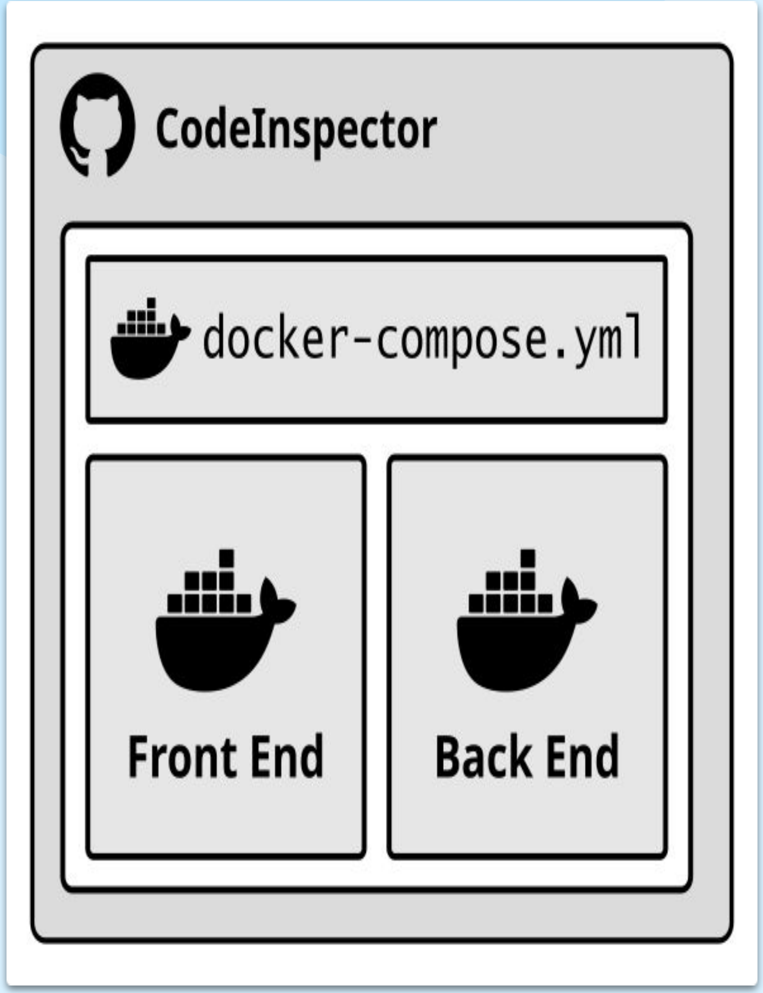
○ Prioritize Hotspots ○ Analyze Commits



CodeInspector Architecture

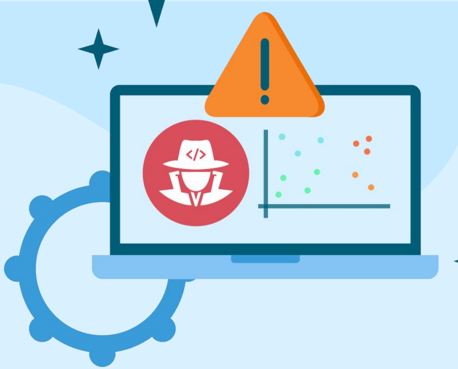
CodeInspector follows a client-server architecture, where the Frontend acts as the client, and the Backend acts as the server.

CodeInspector Architecture



- The backend folder and frontend contains all the server-side logic.
- The frontend folder contains the client-side code and assets responsible for the user interface and experience.
- Both folders have their own dockerfile that dictates how the component is containerized.
- A docker-compose.yml file defines and configures the services orchestrating the deployment and management of both components

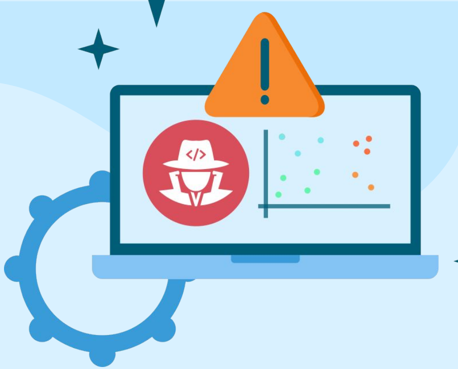
CodeInspector Tech-Stack



React.js

FastAPI

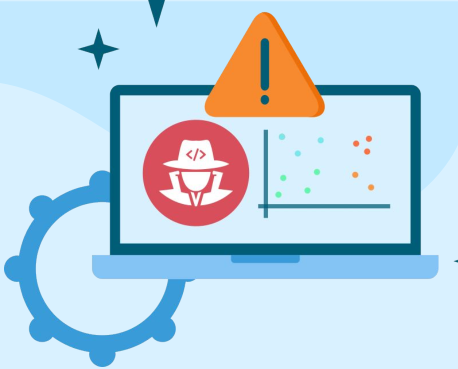
docker



CodeInspector Tech-Stack

PyDriller

for repository mining



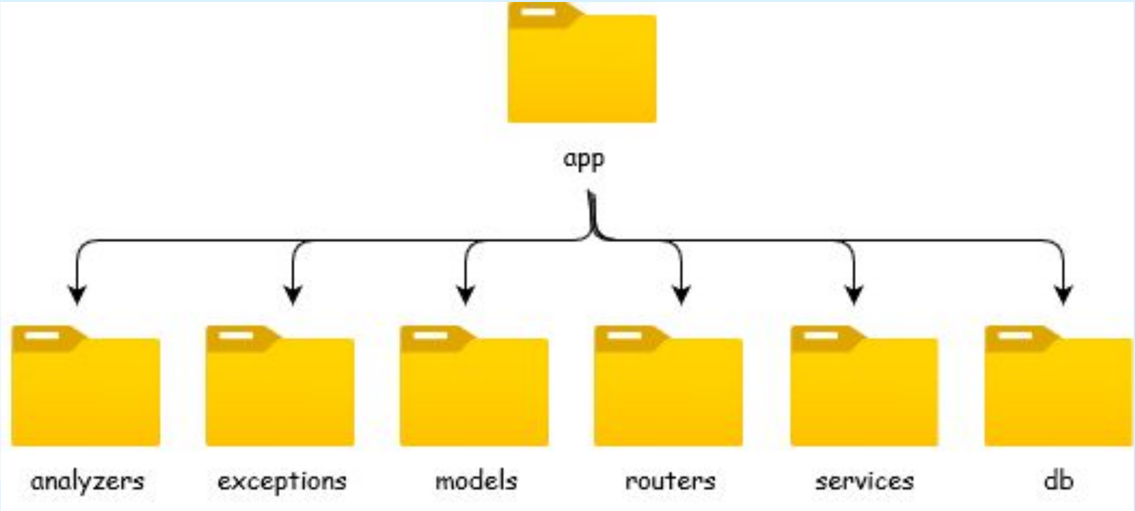
CodeInspector Tech-Stack

MySQL™

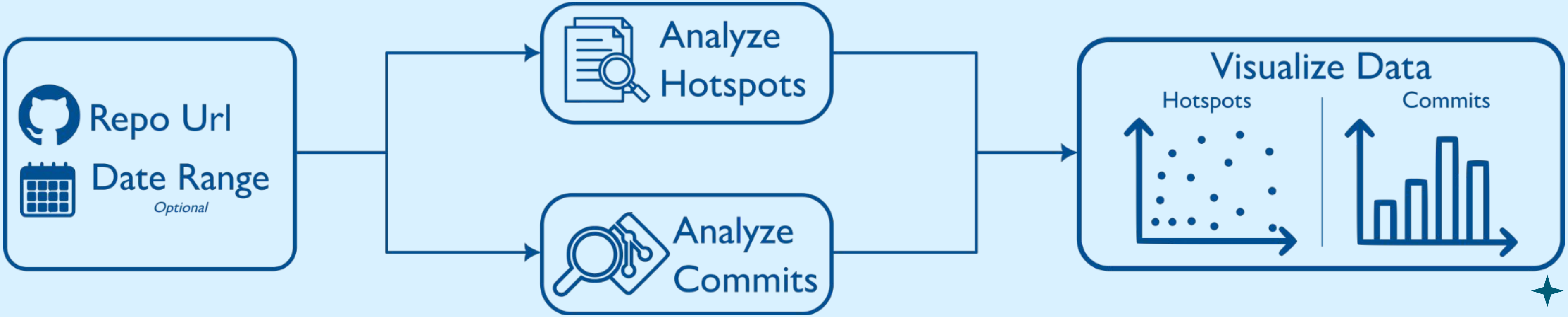
as a Relational Database

CodeInspector Backend

The tool’s backend is organized into different directories, each serving a specific purpose.



CodeInspector Analysis



Repo Url

Date Range

Optional

Analyze Hotspots

Analyze Commits

Visualize Data

Commit Analysis

For the commit analysis, we aim to accomplish 2 things:

1

Find the DMM Score of a Commit

DMM score is a value that "aggregates" the values of all the dmm properties we get from PyDriller

2

Give a rating to the Commit

The Commit Rating should reflect how "good" the included change is.

1. Calculate DMM Score

To calculate the DMM Score of the commit we will use the three DMM metrics retrieved from PyDriller:

- DMM Unit Size
- DMM Complexity
- DMM Interfacing

Then we consider DMM Score to be:

dmm_size + dmm_complexity + dmm_interfacing

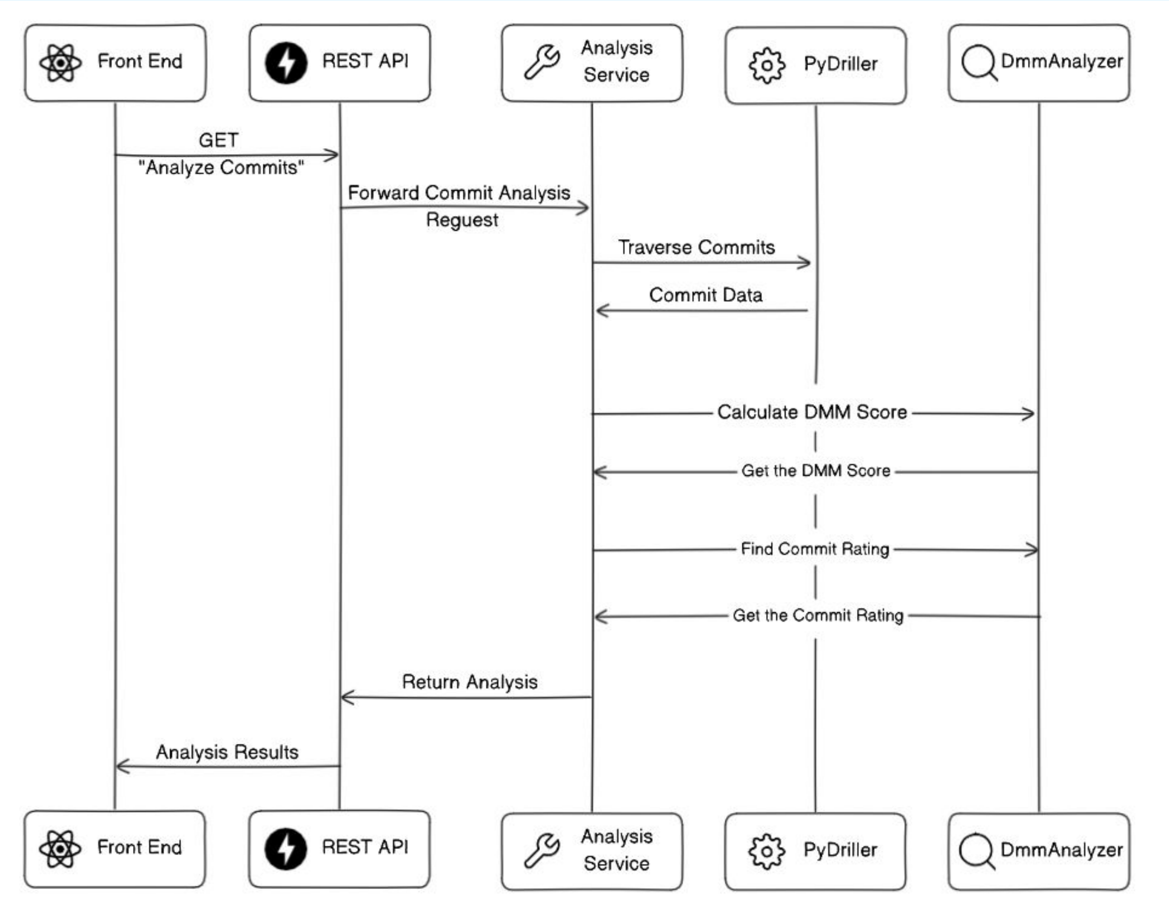
2. Rate the Commit

To rate the commit we will use the DMM Score we calculated.

Since **all DMM Metrics have a value between 0.0 and 1.0**, and the DMM Score is the sum of these metrics, we can assume that **DMM Score will always be between 0.0 and 3.0.**

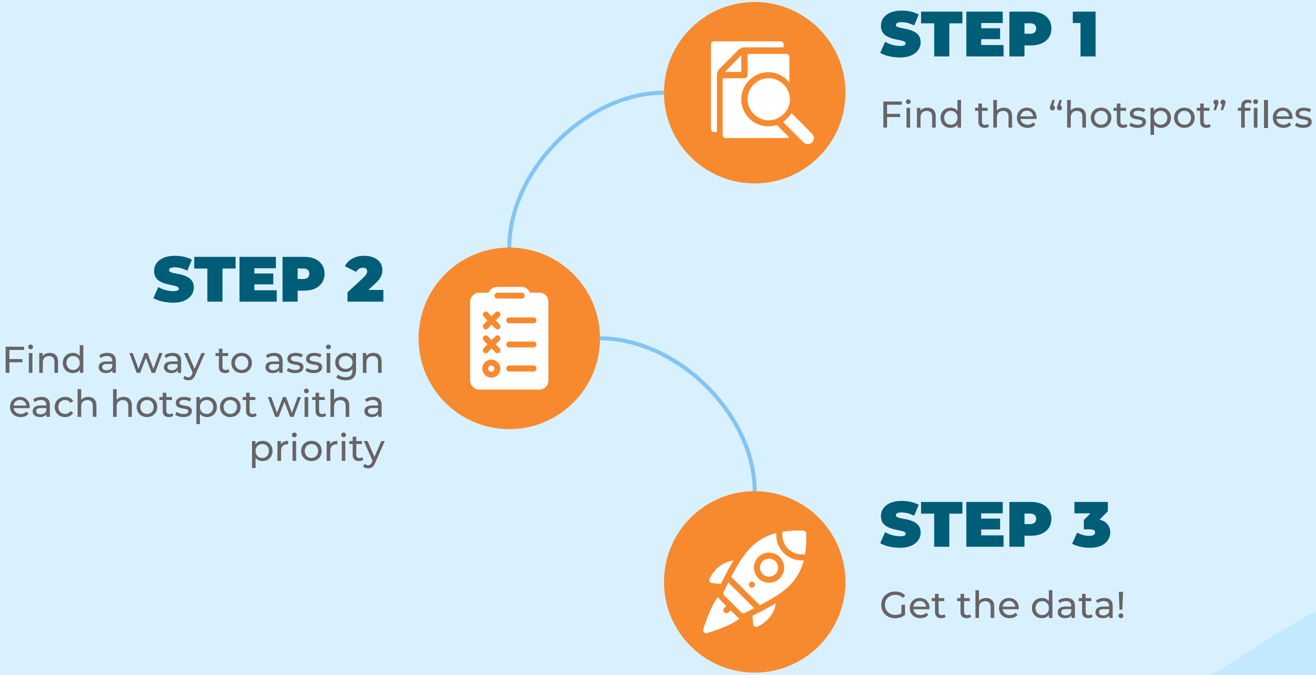
0.0 represents the lowest maintainability and **3.0 the highest.**

Commit Analysis Workflow



Hotspot Prioritization

How to prioritize the Hotspots?



STEP 1

Find the “hotspot” files

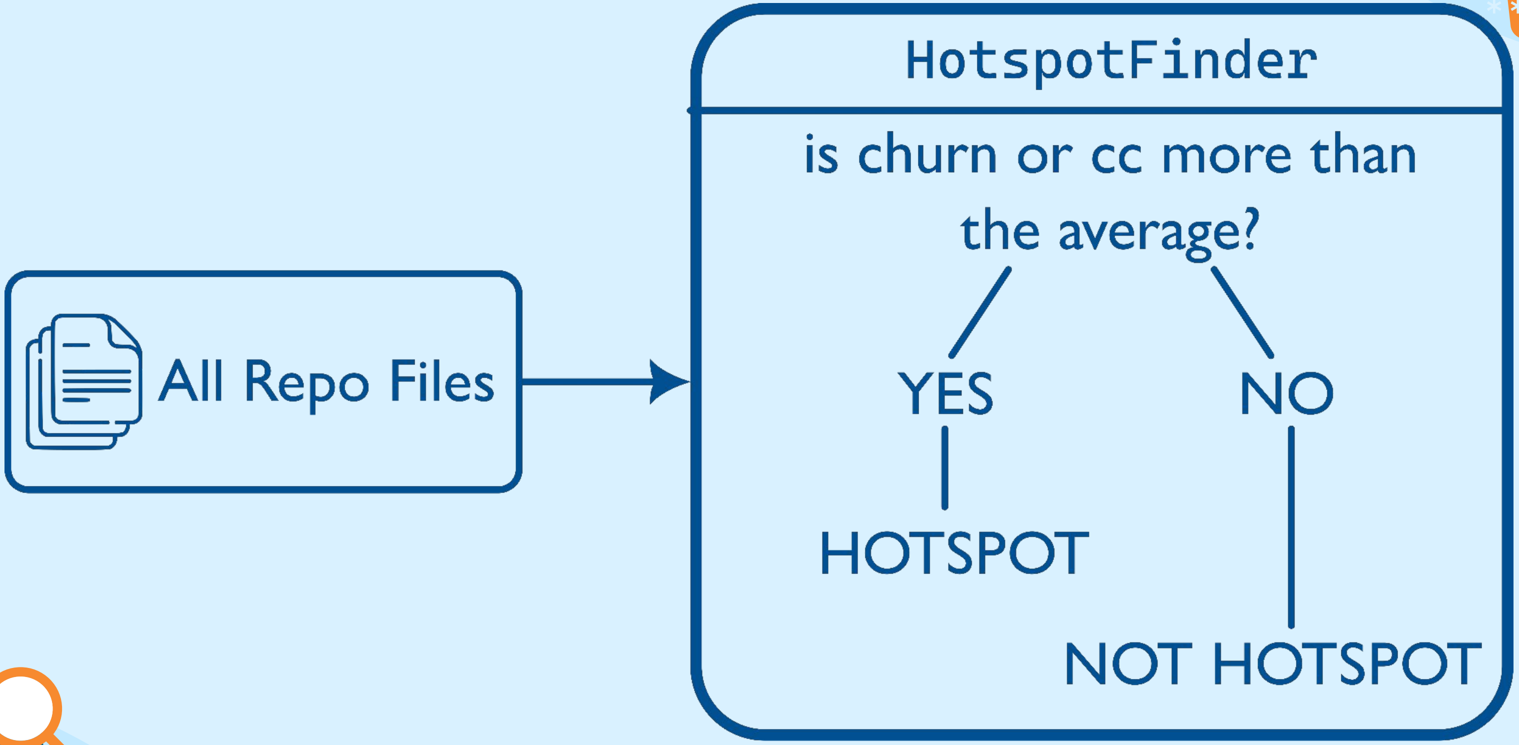
STEP 2

Find a way to assign each hotspot with a priority

STEP 3

Get the data!

1. Find the Hotspot files



All Repo Files

HotspotFinder

is churn or cc more than the average?

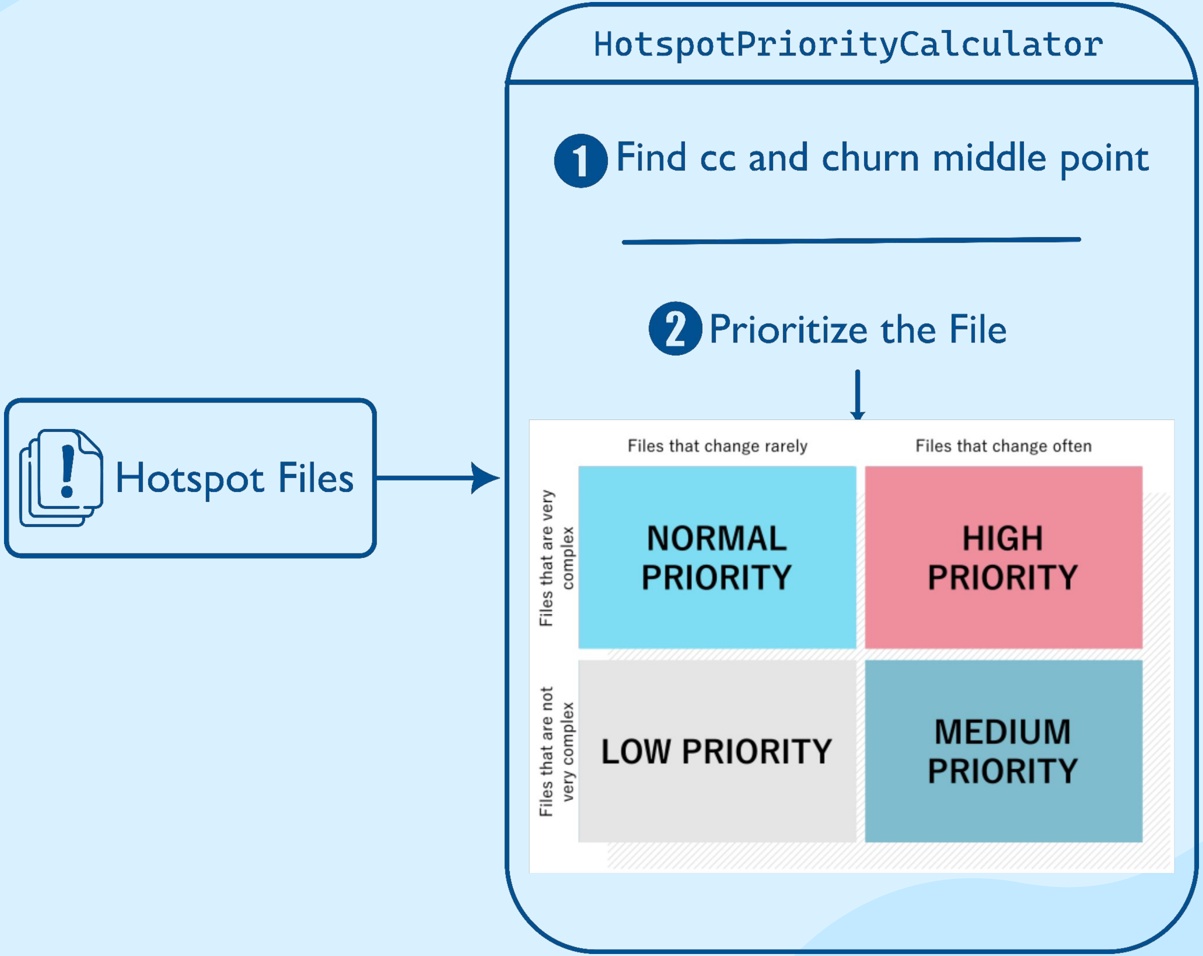
YES

HOTSPOT

NO

NOT HOTSPOT

2. Assign Priority to Hotspots



Hotspot Files

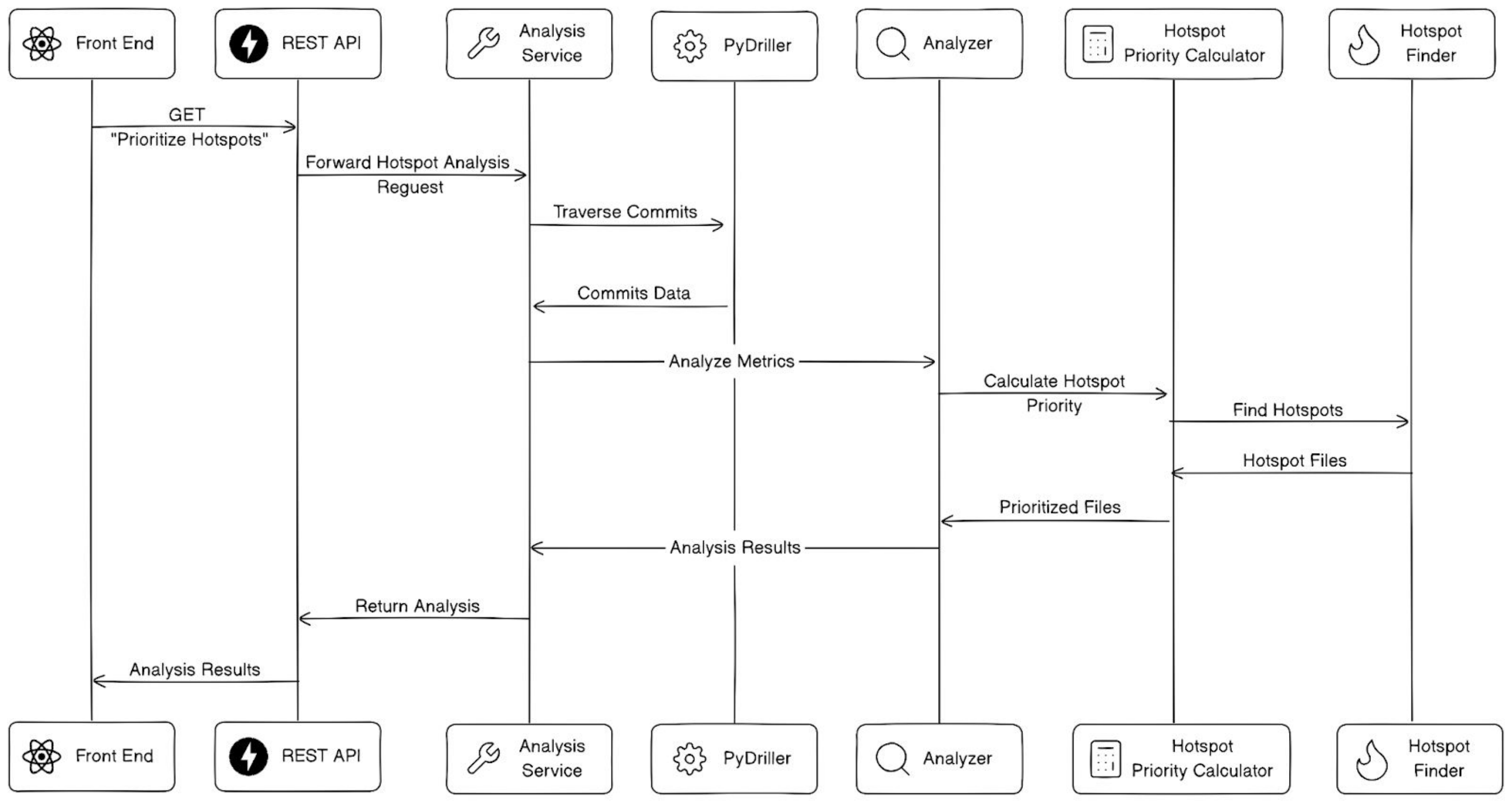
`HotspotPriorityCalculator`

1 Find cc and churn middle point

2 Prioritize the File

	Files that change rarely	Files that change often
Files that are very complex	NORMAL PRIORITY	HIGH PRIORITY
Files that are not very complex	LOW PRIORITY	MEDIUM PRIORITY

Hotspot Prioritization Workflow



Future Steps & Research

STEP 1 MORE EXTERNAL SERVICES

We want to use more external services for the analysis process such as Sonar, PyAssess and Quality Dashboard

STEP 2 VIEW CODE +more data

We want to be able to add a feature that allows users to view the code for specific files and changes and more data related to quality and the repository

STEP 3 VALIDATION +BENCHMARKING

Validation and benchmarking studies to evaluate the tool’s effectiveness in real-world scenarios.

THANKS!

Do you have any questions?

ics21044@uom.edu.gr
arxontisk02@gmail.com
Github Repository

CREDITS: This presentation template was created by **Slidesgo**, including icons by **Flaticon**, infographics & images by **Freepik**

slidesgo