Recitation 4: Static Analysis

Paulo Canelas
Venkata Nikitha Machineni

Carnegie Mellon University

What is Static Analysis?

Static: without executing

Analysis: studying an artifact or phenomenon by decomposing it into its constituent parts

Defects can be on uncom ths:

- Which is why it's hard
- Executing (or interpre ncretely to find such defects is infeasible.

What we really want to do is check the entire possible state space of the program for particular properties.

Brief Overview from Lecture

What is Static Analysis?

Static: without executing

Analysis: studying an artifact or phenomenon by decomposing it into its constituent parts

Defects can be on uncommon or difficult-to-force execution paths:

- Which is why it's hard to find them via testing;
- Executing (or interpreting/otherwise analyzing) all paths concretely to find such defects is infeasible.

What we really want to do is check the entire possible state space of the program for particular properties.

What is Static Analysis?

Program

+

Set of properties

Static Analysis Tool

Results from Static Analysis

True Positive: The error condition warned about can occur at run time, for some program input.

True Negative: Analysis correctly tells us that the program does not contain a given defect.

False Positive: The error condition cannot occur reality, no matter what the program input is.

False Negative: The program can get into an error condition for an attribute covered by the tool for some input, but the tool does not warn of it.

Examples of Static Analysis Tools

Infer

PMD

SpotBugs

Recitation Outline

Recitation Exercises:

- Checking Array Index Out of Bounds with **Infer**;
- Checking Best Practices, Code Style and Error Prone practices with **PMD**;
- Checking any potential bugs with **SpotBugs**.

https://github.com/MSE-QualityAssurance/recitation-4-f23

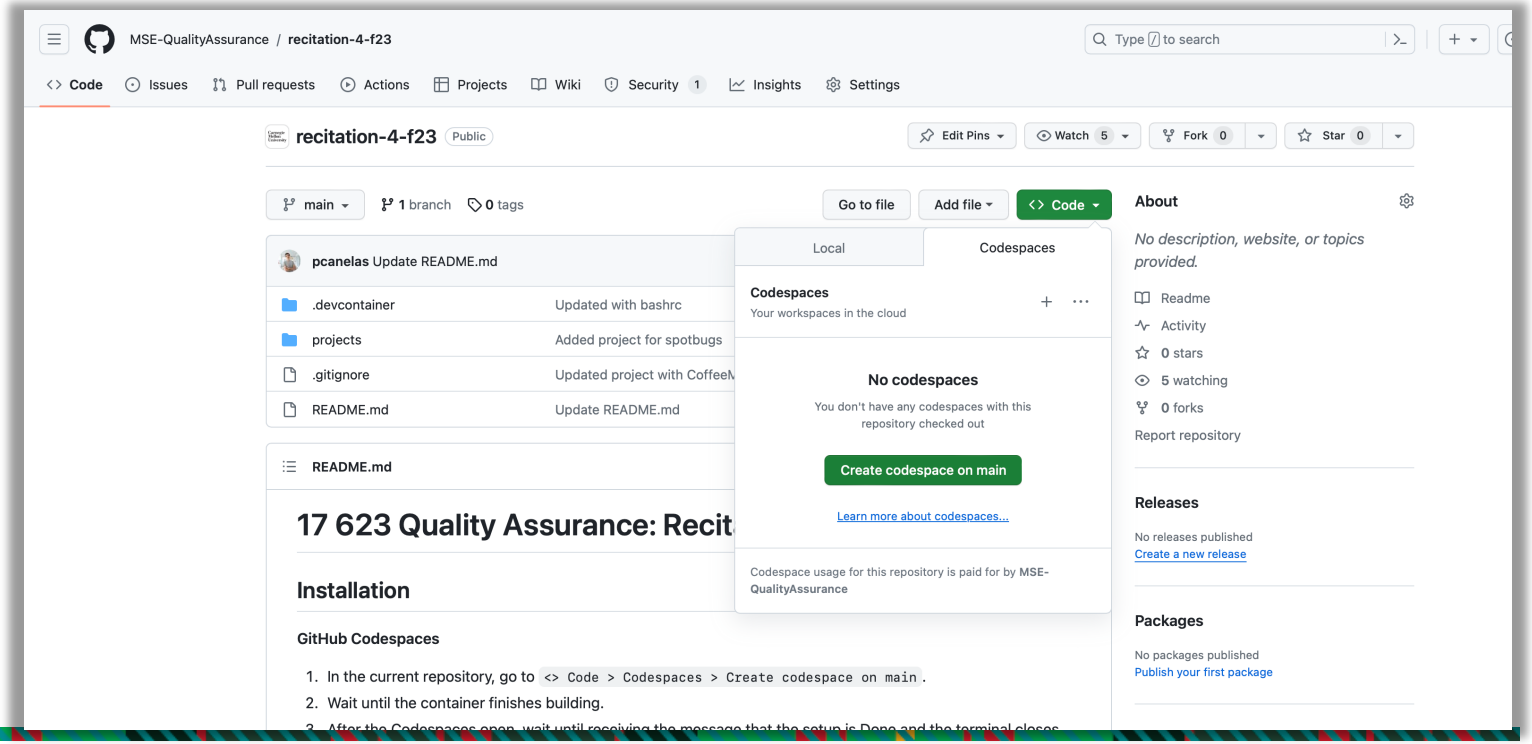
Beginning of Homework 4:

- Build and Setup of the Homework;
- Execution of the static analysis tools in the Simple-Java-Text-Editor.

https://canvas.cmu.edu/courses/36250/assignments/614274

Create a Codespace from Github

https://github.com/MSE-QualityAssurance/recitation-4-f23



Carnegie Mellon University

Exercise 1: Detecting Array Index Out of Bounds with Infer

Exercise 1 (Infer):

1. Open the classes `Student` and `App` and analyze what each one of them does.
2. In the terminal, change to the infer project directory (`projects/infer`).
3. Execute the following command in the terminal: `infer --bufferoverrun -- mvn clean package`
4. Analyze the output generated in the `infer-out` folder. Does it raise any error, and, if so, what program points are responsible?

A list of properties Infer checks can be found here.

Exercise 2: Detecting Bad Practices, Incorrect Code Styles and Error Prone statements with PMD

Exercise 2 (PMD):

1. Open the class `CoffeeMachine` to understand its functionality.
2. In the terminal, change to the pmd-tool project directory (`projects/pmd-tool`) and execute the command:

```
pmd check --rulesets=ruleset.xml -d src/main/java --report-file pmd-report.tx
```

4. Analyze the `pmd-report.txt` report file generated. Which of the rules in `ruleset.xml` were triggered and why?
5. Add three new rules to the `ruleset.xml` that analyzes the code and detects an error. The Index for Java Rules can be found here. Each rule should detect one of the following properties:
   - **Best Practices:** `switch` statements should always contain a `default` case, allowing it to process undefined cases.
   - **Code Style:** Variable names should be descriptive of the information type they contain. For example, `hasWater` is an `int` value, but the prefix assumes the variable is a `boolean` .
   - **Error Prone:** Values in `if` conditions should not be hardcoded, as changes to a value may require manual change into multiple lines, which is prone to errors. For example, changing the amount of water when calling `buyCappuccino` requires changes to line `87` and `95` .

Carnegie Mellon University

Exercise 2: Detecting Bad Practices, Incorrect Code Styles and Error Prone statements with PMD

Exercise 2 (PMD):

1. Open the class `CoffeeMachine` to understand its functionality.
2. In the terminal, change to the pmd-tool project directory (`projects/pm` command:

```
pmd check --rulesets=ruleset.xml -d src/main/java --report-f
```

4. Analyze the `pmd-report.txt` report file generated. Which of the rules triggered and why?
5. Add three new rules to the `ruleset.xml` that analyzes the code and de for Java Rules can be found here. Each rule should detect one of the follo
   - **Best Practices:** `switch` statements should always contain a `defa` process undefined cases.
   - **Code Style:** Variable names should be descriptive of the information type they co example, `hasWater` is an `int` value, but the prefix assumes the variable is a `boolean`.
   - **Error Prone:** Values in `if` conditions should not be hardcoded, as changes to a value may require manual change into multiple lines, which is prone to errors. For example, changing the amount of water when calling `buyCappuccino` requires changes to line `87` and `95`.

ruleset.xml

projects > pmd-tool > ruleset.xml

```xml
<?xml version="1.0"?>

<ruleset name="Custom Rules"
    xmlns="http://pmd.sourceforge.net/ruleset/2.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://pmd.sourceforge.net/ruleset/2.0.0 https://pmd.sourceforge.io/ruleset_2_0_0.xsd">

    <description>
        In this file, you describe your own custom rules for the project.
    </description>


    <!-- Your rules will come here -->
    <rule ref="category/java/errorprone.xml/AvoidMultipleUnaryOperators" />
    <rule ref="category/java/errorprone.xml/IdempotentOperations" />
    <rule ref="category/java/design.xml/CollapsibleIfStatements" />



</ruleset>
```

Add rules to ruleset.xml

Carnegie Mellon University

Exercise 3: Checking any potential bugs with SpotBugs

Exercise 3 (SpotBugs):

1. The `spotbugs` folder contains a project that allows you to visualize the participants and spectators of a Volleyball game, as well as, any incoming games between players.
2. In the terminal, change to the spotbugs project directory (`projects/spotbugs`) and execute the command:

```
mvn spotbugs:check
```

3. Analyze the types of bugs raised after executing the command.

Homework 4: Static Analysis

Carnegie Mellon University

Account
Dashboard
Courses
Calendar
Inbox
History
Studio
Help

17623 › Assignments › Assignment 4: Static Analysis

Fall 2023

Home
Announcements
Syllabus
Quizzes
Assignments
Files
Grades
Zoom
Piazza
People
Gradescope

Assignment 4: Static Analysis

Due Nov 27 by 11:59pm **Points** 100 **Submitting** an external tool **Available** after Nov 15 at 12:30pm

Learning Goals:

- Gain practical experience of using various static analysis tools to measure quality and uncover issues in a real-world project.
- Build an understanding of the relative strengths and weaknesses of static analysis tools, and the appropriate ways to use those tools for quality assurance purposes.

Overview & Resources

Static analysis is a powerful tool for revealing defects, design flaws, and poor quality code. In this homework, you gain practical experience with static analysis tools and build an appreciation for the relative strengths and limitations of those tools. We will be using the following tools:

- SpotBugs
- PMD
- Facebook Infer

Your task is to run these three static analysis tools on the JavaSimpleEditor source code and compare their results. You should appropriately customize the tools, run them, and analyze the results.

You will run the analysis on the on the base source code of the SimpleJavaTextEditor. The instructions below include instructions on where to get the tools, to help you with this assignment we've created a GitHub classroom repository that already has the required tools installed: https://classroom.github.com/a/jzZ0s1SS.

You can clone this repo to get access to the tools in codespaces, but you should not need to submit or execute any code for this assignment.