Membranesoft: Static Java Code Checker

(Automatic Identification of False Alarms)

Synopsis:

Static Code Analysis (SCA) tools try to find bugs by analyzing the source code using some static analysis techniques which do not require executing the code. Different static analysis techniques such as syntactic pattern matching, data flow analysis, model checking and verification theorems have been used by these tools to discover a wide range of bugs. While intending to help quickly identifying bugs in the program, these tools however, usually generate an unduly enormous number of warnings due to the use of underlying approximate analysis techniques. This information overload can easily hinder the potential benefits of such tools. Understanding the warnings and their categorization can help to perceive the strengths and limitations of these SCA tools. Our project aims to develop a tool which automatically filters the false alarms generated by the SCA tools, which will help testers to only concentrate on actual errors.

The Project:

Software contains bugs. A software bug is a defect in the software. As a result, the functionality of the software might get altered and disrupted. Some bugs are easy to find whereas others are almost impossible to figure out as the code having these bugs may never get exercised, or their execution may not result in observed failures. Some bugs that come up may even go unnoticed because they are not apprehended as bugs or are not enough severe. Software bugs are caused by several types of errors that are made while coding. An error is a mismatch between the program and its specifications. For instance, bugs may be resulted due to the programming errors made intentionally or unintentionally, like logical inconsistencies (e.g., a conditional test that cannot possibly be true), runtime errors (e.g., dereferencing a null pointer), resource leaks (performance of the program degrades until the program crashes) or potential security violations (e.g., SQL injection). Software bugs can outlay companies' large amounts of money, specifically when they induce to software failures. Hence, fixing bugs before the software is put to use, is very important.

SCA can be viewed as an automated code review process. It is used for detecting bugs in the source code without the need of executing the code. It looks for violations within the code and some specific type of programming errors. Additionally it can help in maintaining the coding conventions.

Success Criteria:

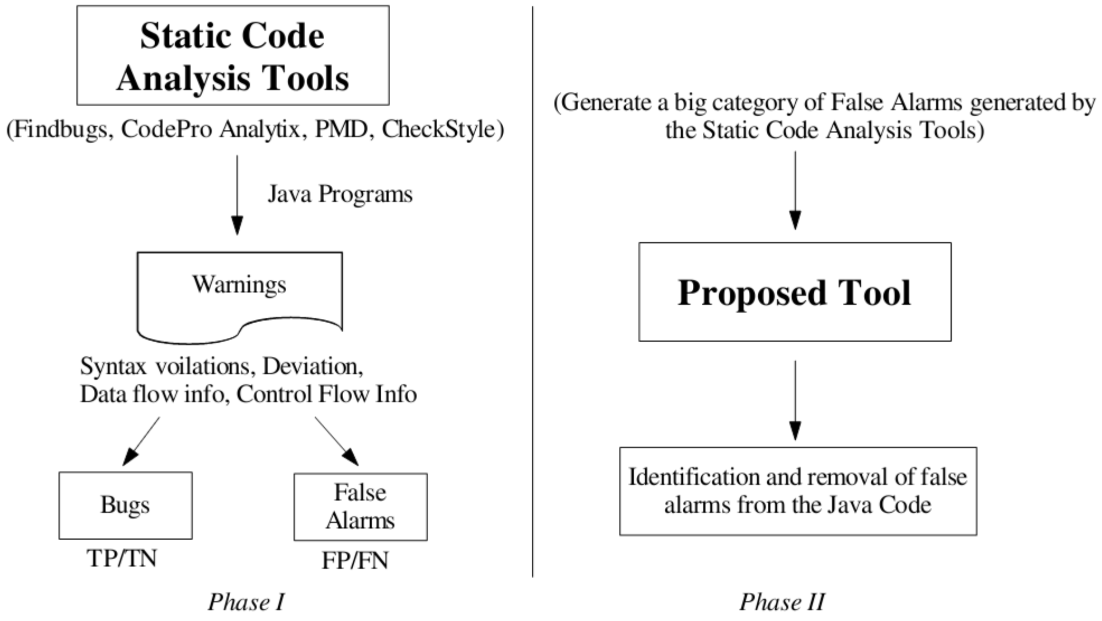
1. Membranesoft can recognize and filter the false alarms.
2. Membranesoft will generate false alarms data which can be later used to strengthen the database using Machine Learning techniques.
3. Membranesoft can run over any static analysis tool for analysis.

Road-map:

Since the analysis of different static analysis tools (e.g., Findbugs, PMD, CodePro, UCDetector, Checkstyle), and the types of errors and warning generated by the tools are recorded. We will first focus mostly on creating a database.

Week 1: Understanding the organization work
Week 2: Project Literature
Week 3 and 4: Familiarity with static analysis tools and the bug categorization
Week 5 and 6: Analysis of static analysis tools *(completed)*

Week 6, 7, 8 and 9: Development of the Membranesoft (including database updation)
Week 10 and 11: Writing test cases and testing the tool under development
Week 12: Preparation of Installers and preparation of Manuals (doc/video)


Static Code Analysis Tools
(Findbugs, CodePro Analytix, PMD, CheckStyle)
Java Programs
Warnings
Syntax voilations, Deviation, Data flow info, Control Flow Info
Bugs
TP/TN
False Alarms
FP/FN
Phase I
(Generate a big category of False Alarms generated by the Static Code Analysis Tools)
Proposed Tool
Identification and removal of false alarms from the Java Code
Phase II

Process of Identification of False Alarms *(Phase I is completed)*