

**NALYZER: AI BASED COMMUNITY-DRIVEN SOURCE CODE
ANALYSIS TOOL**

Denuwan Himanga Hettiarachchi

(199327M)

Degree of Master of Science

Department of Computer Science and Engineering

University of Moratuwa

Sri Lanka

April 2021

NALYZER: AI BASED COMMUNITY-DRIVEN SOURCE CODE ANALYSIS TOOL

Denuwan Himanga Hettiarachchi

(199327M)

Thesis/Dissertation submitted in partial fulfillment of the requirements for the degree Master
Of Science

Department of Computer Science and Engineering

University of Moratuwa

Sri Lanka

April 2021

DECLARATION OF THE CANDIDATE & SUPERVISOR

I declare that this is my own work and this dissertation does not incorporate without acknowledgment any material previously submitted for a Degree or Diploma in any other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgment is made in the text. Also, I hereby grant to the University of Moratuwa the non-exclusive right to reproduce and distribute my dissertation, in whole or in part in print, electronic or other media.

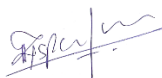
I retain the right to use this content in whole or part in future works (such as articles or books).

Signature

Date:

The above candidate has carried out research for the dissertation under my supervision.

Name of Supervisor: Dr. Indika Perera



15-11-2021

Signature

Date:

ABSTRACT

Identifying error-prone code snippets and potential vulnerabilities in the early stages of the development process allows reducing the considerable amount of time & the cost of the software project. But the process of ensuring the reliability of software projects has become a significant challenge due to the high complexity & the scalability of modern software projects. Also, the dynamic nature of modern frameworks & programming languages becomes a barrier to consistency. Manual code reviews/automated code analysis tools are obsolete due to time constraints & lack of adaptability for new programming languages & frameworks.

Nalyzer project aims to build a Machine Learning (ML) model to identify error-prone code snippets and potential vulnerabilities in the source code. And introduce a self-sustainable approach to adopt future programming languages & framework changes.

We used Convolutional Neural Network (CNN) deep learning algorithm to build an ML model for classifying buggy & non-buggy code snippets from source code. And introduce a maven customized build plugin to push source code to ML model & get prediction as a step in the Continuous Integration/Continuous Delivery (CI/CD) pipeline. Then the generated Nalyzer analysis result was published on the interactive dashboard inside the project directory. Interactive dashboard facilitated to get feedback from developers to improve ML model accuracy & future adaptations.

We evaluate the ML model in terms of F-measure. The evaluation results demonstrated the compatibility of ML techniques in the source code analysis paradigm with a significant score. And the interactive dashboard makes sure of the self-sustainability of the ML model through a Community-Driven approach.

Nalyzer project proves that the ML approach is an alternative for overcoming the limitations of manual code reviews and automated code analysis tools.

Key Words: Machine Learning (ML), Neural Network, Convolutional Neural Network (CNN), Source Code Analysis

ACKNOWLEDGEMENT

I would like to take this opportunity to express my gratitude to my supervisor Dr.Indika Perera for all guidance & advice provided through this journey.

And also I would like to thank industry experts, who shared their experiences and thoughts during the literature review and background studies. Their contribution adds value to this project to achieve the end goals in a more practical manner.

Also I would like to thank Kirill Eremenko, Hadelin de Ponteves for their great content on Udemy platform. Their amazing tutorials help me to clear my path of implementation.

Finally yet importantly, I want to mention the role of my parents in my journey. They add unconditional support to manage my career & studies in a free mindset.

TABLE OF CONTENTS

DECLARATION OF THE CANDIDATE & SUPERVISOR	i
ABSTRACT	ii
ACKNOWLEDGEMENT	iii
TABLE OF CONTENTS	iv
LIST OF FIGURES	vi
LIST OF TABLES	vi
1. INTRODUCTION	1
1.1. Background Context & Motivation	1
1.2. Research Problem Definition	2
1.3. Research Objectives	3
1.4. Scope of the Research	4
2. BACKGROUND AND LITERATURE REVIEW	5
2.1. A Survey of Unit Testing Practices	7
2.2. Junit: Unit Testing and Coding In Tandem	12
2.3. The Art of Unit Testing (2nd Edition)	17
2.4. Improving the Bug Tracking System	21
2.5. Building Useful Program Analysis Tool Using an Extensible Java compiler	26
2.6. Cute: A Concolic Unit Testing Engine for C	30
2.7. A Comparison Of Bug-finding Tools for Java	35
2.8. Pragmatic Unit Testing In Java 8 with JUnit	40
2.9. BP Neural Network-based Effective Fault Localization	45
3. METHODOLOGY	50
3.1. Data Gathering	52
3.1.1. Customized maven plugin	53
3.1.2. Customized PWD plugin	54
3.1.3. Java Parser	56
3.1.4. Syntax Tokenizer Mechanism	57
3.1.5. Summary of the Training Data	57
3.2. Build Neural Network Model	59

3.2.1.	Convolution Neural Networks (CNN)	59
3.2.2.	Implementation of CNN	61
3.2.3.	Hardware Implementation	62
3.2.4.	Access to Build Model	62
3.3.	The Nalyzer Analysis Approach	63
3.3.1.	Nalyzer Maven plugin	63
3.3.2.	Interactive Dashboard	65
4.	RESULTS & DISCUSSION	67
4.1.	Machine Learning Performance Measure Mechanisms	67
4.2.	Precision/Recall/F-Measure & Confusion Matrix	68
4.3.	Nalyzer Model Evaluation	69
4.3.1.	Build Confusion Matrix	72
4.3.2.	Precision Calculation	75
4.3.3.	Recall Calculation	75
4.3.4.	F1-Score Calculation	75
4.4.	Discussion	76
5.	SUMMARY AND CONCLUSIONS	79
	REFERENCES	82

LIST OF FIGURES

Figure 1: Cost per defect analysis in SDLC.....	1
Figure 2: Survey finding of unit testing practices.....	8
Figure 3: JUnit test setup & test cases	14
Figure 4: An example of a test-code-coverage report in TeamCity with NCover	20
Figure 5: The decision tree used to get the full picture of the situation.....	24
Figure 6: CUTE generates input against C code.....	32
Figure 7: The BP neural network used in the research method	47
Figure 9: High-Level Architecture Diagram.....	51
Figure 10: Difference between dense connectivity and sparse connectivity	60
Figure 11: Nalyzer user interactive dashboard.....	66
Figure 12: Nalyzer feedback window	66
Figure 13: Overall evaluation result.....	76

LIST OF TABLES

Table 1: Conclusions of the survey (Definitions of Unit Test paradigm).....	9
Table 2: Finding of analysis tools	37
Table 3: Each tool generated warnings	38
Table 4: Maven command for run NalyzerDK	53
Table 5: NalyzerDK options	53
Table 6: Syntax extraction summary (Process of AST to 1D array).....	56
Table 7: Identified violations in each repo.....	58
Table 8: Structure of neural network	61
Table 9: Command use for run Nalyzer Maven plugin	63
Table 10: JSON formatted Nalyzer result	64
Table 11: ML model evaluation metrics	67
Table 12: ML model evaluation matrix & terms	68
Table 13: Overall Evaluation Data Extraction Summary	71
Table 14: Apache/hbase - Confusion matrix.....	72
Table 15: Apache/incubator-pinot - Confusion matrix	72
Table 16: Apache/hudi - Confusion matrix.....	72
Table 17: Apache/ignite - Confusion matrix.....	73
Table 18: Apache/zookeeper - Confusion matrix	73
Table 19: Spring-projects/spring-data-rest - Confusion matrix	73
Table 20: Spring-projects/spring-ws - Confusion matrix.....	73
Table 21: Oracle/helidon - Confusion matrix	74
Table 22: Oracle/opengrok - Confusion matrix	74
Table 23: AWS/aws-sdk-java- Confusion matrix	74

LIST OF EQUATIONS

Equation 1: Process within the CNN filter system.....	61
Equation 2: Define F1-score	69
Equation 3: Precision Calculation.....	75
Equation 4: Recall Calculation	75
Equation 5: F1-Score Calculation	75