Analyzing Software Quality and Maintainability in Object-Oriented Systems using Software Metrics

FARIA TASNIM
Computer Science and Engineering
Brac University
Dhaka, Bangladesh
faria.tasnim1@g.bracu.ac.bd

MAHDI ISLAM
Computer Science and Engineering
Brac University
Dhaka, Bangladesh
mahdi.islam@g.bracu.ac.bd

SALEQUZZAMAN KHAN
Computer Science and Engineering
Brac University
Dhaka, Bangladesh
salequzzaman.khan@g.bracu.ac.bd

FARIB MD.FERDOUSH
Computer Science and Engineering
Brac University
Dhaka, Bangladesh
farib.md.ferdoush@g.bracu.ac.bd

FATEMA HAQUE
Computer Science and Engineering
Brac University
Dhaka, Bangladesh
fatema.haque@g.bracu.ac.bd

Mr. MD. AQUIB AZMAIN
Lecturer
Computer Science and Engineering
Brac University
Dhaka, Bangladesh
aquib.azmain@bracu.ac.bd

***Abstract*—Effective evaluation of software quality and maintainability is compulsory for successful object-oriented system development, and the potential of software metrics in achieving these goals are investigated in this research. To evaluate the quality of software, this research employs software metrics to identify potential errors and weaknesses in object-oriented systems. This analysis has been conducted by us in the Python programming language. We have applied machine learning techniques to different software metrics to analyze the issues consistently, which has evaluated the effectiveness and long-term feasibility of the system. Lastly, this study lays a foundation for future advancements in software quality assurance, demonstrating the significant benefits of integrating machine learning with traditional quality measurements to enhance the predictability and reliability of object-oriented systems.**

Keywords: LOC (Lines of Code) Comment Percentage Cyclomatic Complexity (CC) Weighted Methods per Class (WMC) Access to Foreign Data (ATFD) Response For a Class (RFC)

I. Introduction

Evaluating the quality and the maintainability of software is immensely important to the overall well-being of object oriented systems. With increasing system size, automated approaches based on software measurements are for detecting more subtle deficiencies such as code smells with roots in deeper design problems. This research specifically targets code smell detection in Python programs based on static and particularly dynamic metrics with the help of machine learning models. The benefits of this approach are that it is going to assist the developers to code smart, clean, and sustainable code.

II. Research Objectives

Evaluating Software Quality with Metrics The goal is to assess software quality in object-oriented systems using metrics related to code complexity, cohesion, coupling, and other key indicators.

Assessing Maintainability Analyze metrics such as modularity, reusability, readability, and maintainability to evaluate and identify areas for improvement in long-term software maintainability.

Correlating Metrics with Quality Attributes Establish relationships between software metrics and quality attributes (e.g., dependability, efficiency) to understand how design decisions impact software quality and maintainability.

Benchmarking and Comparative Analysis Compare obtained metrics with industry benchmarks to evaluate software quality and maintainability against standard practices.

Recommendations for Improvement Provide actionable recommendations and best practices to enhance software quality and effectiveness based on the analysis.

Integration into Development Practices Develop guidelines to incorporate insights into software development processes, improving overall quality and maintainability.

III. Comparison of Related Studies

Metrics: Both static metrics (cyclomatic complexity, LOC) and object-oriented metrics (ATFD, WMC, RFC) are widely used to assess software quality and maintainability. However, metrics like LOC are mostly insufficient for identifying deeper issues in complex systems [1], [2]. On the other hand, object-oriented metrics offer better insights into modularity and class-level dependencies [3], [4].

Methodologies: Studies progressively implementing machine learning models (Random Forest, Gradient Boosting) for maintainability predictions [3], [5]. Although the ML techniques enhance predictive accuracy, most models are trained on Java datasets, limiting their universality to other languages.

Programming Languages: Most of the research focuses on Java-based projects while few studies are focused on other popular languages like Python [3]. This leaves a gap in understanding how object-oriented metrics apply to Python's growing ecosystem.

Limitations: It is widely observed that majority researches depends heavily on static metrics that fail to observe dynamic behaviors of software at runtime [4]. Furthermore, ML models trained on only one language datasets risk limited cross-platform applicability [6].

IV. Detailed Description of the Developed Tool or Python Script

A. Libraries used

Python libraries help to ease many crucial operations including data analysis and data visualization, web scraping, image detecting or processing, machine learning model creation, textual information processing, etc. [7]. Python libraries are an array of fundamental sets of necessary skills that make it easier for the user not to write new codes [8].

1. Requests: Responsible for HTTP request to download repositories from GitHub.

2. OS: Contributes to file system, and platform management.

3. Zipfile: Compresses and extracts repository files using specified options.

4. Openpyxl: Develops Excel reports and also modify them.

5. Re: Executes text using regular expression which is pattern matching.

6. Ast: A very well written tool for analyzing Python code through parsing and the use of abstract syntax trees.

7. Collections (Counter): Includes data count and data frequency.

8. Pandas: Offers data preprocessing and data analysis, as well as data visualization.

B. Breaking down the script to achieve the desired outputs for the research

These functions have been used to make the tool and analyze the results of the research.

1. Download and Unzip GitHub Repository: Checks HTTP responses, fetches a repository as a ZIP file, and unpacks its content.

2. Analyze and Create Excel Reports: Produces Excel tables, containing repository information such as commit history and contributors.

3. Analyze Python Files: Studies repository files to determine such values as:

- ATFD: Availability of Foreign Data for measuring class coherency.

- WMC: Weighted Methods per Class for determining the degree of complexity of the system.

- RFC: Response For a Class for evaluating overall class interface.

4. Extract Comments and Methods: Regex is used to scan Python files for comments, methods, class definitions giving information about documentation and structure of the code.

C. Script Execution Process

The script is initiated using two parameters: The URL of the GitHub repository and an authentication token to be used. It downloads and extract the repository, then scans the repository's metadata and Python files, and produce output in Excel tables. The commit details, contributors activities and some specific metrics related to the QA and maintenance of the code of the language Python is part of the analysis.

To summarize, the execution of the script consists of several interrelated steps: the compilation of all the feedback from the user, the download and extraction of the repository, the processing of the contents and metadata of the file, and the preparation and generation of a comprehensive report. The process is entirely automated using libraries for accurate identification in addition to repetitive tests of the GitHub repository using methodical approach. The result of this includes two excel docs that not only provide a summary of info for the activity and quality of the code in the repository but also provide more detailed description of the respective subject.

V. Comparison of Our Custom Python Tool with Open source Python libraries Online (Pylint and Radon)

This chapter also establishes a comparison between the custom Python tool developed here and two well known code analysis tools named Pylint and Radon. Actually, both Pylint and Radon are designed for error detection, predicting potential errors, measuring program complexity, and increasing the quality of the given code, whereas our tool is designed particularly for the given research objectives. However, this tool is enhanced with other features which this chapter describes and their distinction from the other tools.

A. Our Custom Python Tool

The offered solution stands out as a GitHub repository analysis tool capable of doing much more than Pylint or Radon. It lets you download and analyze the contents of Python files located in GitHub repositories and carries out the extraction of certain parameters in the form of a report in Excel sheets. In these regards, it estimates such indexes as ATFD, WMC, RFC, which is the most informative when it comes to the analysis of the structure of Python files, especially the methods, classes, comments, etc.

B. Comparative Analysis

This section highlights the similarities and differences of the solutions.

The primary function of all three tools is for the static analysis of Python code, finding flaws, and suggesting various improvements with respect to maintainability, complexity, and adherence to coding standards. These tools predict code quality metrics with the goal of helping developers write better code. They are extensible and easily integrated into other Python-based systems, from CI pipelines to research environments.

While sharing some common features, the tools differ in scope, output formats, supported metrics, GitHub integration, and customization. The custom Python tool is very good at analyzing GitHub repositories, providing Excel reports for non-developers, and evaluating object-oriented code metrics

such as ATFD, WMC, and RFC. It also supports comment extraction for deeper analysis, which Pylint and Radon lack.

Pylint: It is mainly focused on bug detection, code styling, and also PEP 8 violations. Its main output is through CI pipelines, and it can be highly customized using configuration files. Radon provides metrics related to cyclomatic complexity, Halstead metrics, and the Maintainability Index. It has outputs in JSON and XML but only partial GitHub integration, and it doesn't analyze comments.

In the end, all three support code analysis, but they serve different purposes. The custom tool is uniquely purposed for GitHub repository analysis and reporting, making it ideal for research-oriented tasks. Meanwhile, Pylint and Radon offer broader customization, community support, and utility in wider Python development contexts.

VI. Description Of The Models

The Python tool designed for this work allows for the identification of patterns in object-oriented Python sources stored in the GitHub platform. The commits, developer details, the percentage of comment, and few statistical calculations like Lines of Code (LOC), Access to Foreign Data (ATFD), Weighted Methods per Class (WMC), and Response for a Class (RFC) are extracted by it. When scanning the Python files and analyzing the distributions of repositories and textual analyses of the files, the tool employs the AST module and regular expressions to transform the files into Excel reports.

The study employs the application of machine learning techniques RandomForestRegressor, DecisionTreeRegressor, KNeighborsRegressor, and Gradient BoostingRegressor to estimate the LOC depending on features such as ATFD, WMC, and RFC. It is a predictive approach that provides revelation into the interaction of code metrics that define the quality and structure of a project.

When it comes to interpreting the results and determining the quality of the code, the tool employs regression modeling in conjunction with state of the art code analysis techniques to provide developers and researchers a toolkit to work with their OOP-based Python projects.

VII. Description of the Data

Firstly, in figure 1 we prepared a summary file including the repository information of 20 projects by adding total developer count, number of classes and methods, averages, and totals of comment percentages and software metrics like RFC, WMC, ATFD, and LOC.

Secondly, we created the major dataset by combining 20 sheets of Python file analysis into a single dataset with 21,563 entries which is in figure 2. This dataset presents insights at the individual Python file level for each repository regarding the total lines of code, ATFD, WMC, RFC, LOC, and comment percentages.

RFC: The number of methods that may be invoked in response to a message sent to an object of a class, including methods in subclasses and superclasses of the class. The higher the RFC, the more complex a class is, hence harder

Repo name	Number of devs	Project duration	Number of classes	Number of methods	Total comment %	Total LOC	Total ATFD	Total WMC	Total RFC	Avg comment %	Avg LOC	Avg ATFD	Avg WMC	Avg RFC
3b1b/manim	2	11.5 months	22	489	771.5642	24245	3484	1951	1705	4.0825	128.9692	18.4558	10.3227	9.0211
sparsy/DeepFaceLab	1	16 months	26	1534	1104.5315	59566	6932	2656	2445	3.4955	125.2088	21.9567	8.3544	7.7455
apache/airflow	14	1 day	308	5871	76536.4345	359965	31728	15077	13652	34.1375	158.77029	14.1516	6.7247	6.0892
openai/gpt-2	10	22 months	309	5905	76591.2234	356466	31846	15113	13663	34.053	158.3702	14.1663	6.7228	6.0778
s0md3v/XSStrike	10	33 months	301	5842	76491.1221	356182	31816	15049	13561	33.933	158.0923	14.1216	6.6995	6.019
hardikvasa/google-images-download	16	6.5374 Months	302	5870	76593.24515	357222	31894	15077	13612	33.89081644	158.0628319	14.11239	6.671238938	6.02300885
s0md3v/Photon	6	44.4812 Months	302	5899	76672.14514	358307	31975	15107	13612	33.67249968	157.3592446	14.0426	6.63460694	5.978041282
karpathy/neuraltalk	10	10.3482 Months	305	5919	76718.3418	358954	32023	15127	13638	33.63364393	157.3669	14.03902	6.63174	5.978957
xonsh/xonsh	12	4.2707 Months	383	10375	78041.43	420517	37205	19677	15327	31.22906	168.1941	14.88715	7.87395	6.135253
shobrook/rebound	9	43.0396 Months	384	10415	78059.73476	421077	37312	19728	15361	31.18647014	168.2285153	14.90691	7.8817419	6.137055557
atomicals/atomicals-electrums	4	0.56 Month	62	1253	551.57	22635	1513	862	937	10.407	427.075	28.54	18.15	17.67
pi-hole/pi-hole	5	3.02 months	14	82	276.66	4882	446	192	99	3.07	34.36	4.93	2.13	1.1
giuspen/cherrytree	3	0.82 month	4	177	280.004	4528	646	178	70	10	161.8	23.07	6.35	2.5
WikidPad/WikidPad	4	52.6284 Months	180	10650	3921.54	150853	15990	10590	6145	14.68	564.98	59.88	39.66	23.01
zim-desktop-wiki/zim-desktop-wiki	11	1.6415 Months	13	5371	2055.08	74473	9788	5375	4837	9.383963446	340.0501283	44.69406093	24.543379	22.08675799
ASKBOT/askbot-devel	1	6.2746 Months	76	3073	1770.981	59904	6560	3161	1796	4.878754	165.0248	18.07163	8.707989	4.947658
ankitects/anki	9	0.4399 Month	103	285	2044.554	40545	7014	3034	2257	9.046698	179.4027	31.0354	14.75221	9.986726
enthought/mayavi	3	7.0833 Months	35	3979	13942.41	75244	8845	3856	2978	24.24787	130.8591	15.38261	6.845217	5.17913
reviewboard/reviewboard	2	1 month	149	7489	4050.121	243178	15266	9444	9882	4.166791	250.1831	15.70576	9.716049	10.16667
kassoulet/soundconverter	8	19.5796 Months	5	78	495.1899	1197	216	79	22	35.3707	85.5	15.42857	5.642857	1.571429

Fig. 1: Summary of the repositories

```
df.head()
```

	repo_name	File_Name	Comments	Total_Comments	Class_Names	Method_Names	Comment_Percentage	Total_Lines_of_Code	atfd	total_wmc	rfc
0	3b1b/manim	copy_of_project_template.py	# -*- coding: utf-8 -*-\n# importing Matplotli...	28.0	NaN	predict_diabetes	17.177914	163.0	2.0	1.0	0.0
1	3b1b/manim	extract_two_excel.py	NaN	0.0	NaN	download_and_unzip_github_repository, analyze_...	0.000000	136.0	29.0	6.0	0.0
2	3b1b/manim	client.py	# 100 for 5 as keng is 5 for HELLO	1.0	NaN	send_message	5.000000	20.0	3.0	1.0	0.0
3	3b1b/manim	server.py	NaN	0.0	NaN	NaN	0.000000	24.0	0.0	0.0	0.0
4	3b1b/manim	client2.py	# 100 for 5 as keng is 5 for HELLO	1.0	NaN	send_message	4.000000	25.0	3.0	1.0	0.0

Fig. 2: First few lines of the dataset

to understand and test.

WMC (Weighted Methods per Class): Summarizes the complexity of all methods in a class, often based on decision points. Higher WMC indicates more functionality but also greater maintenance difficulty and reduced comprehensibility.

ATFD (Access to Foreign Data): Counts the number of different attributes from other classes accessed directly or by "getters" and "setters." High ATFD is indicative of poor encapsulation, high coupling, and low modularity, which may lead to more maintenance bugs.

LOC: This is a measure of the size of the codebase, by counting the lines of actual code. A higher LOC means increased project size and complexity, which can further lead to higher defect rates and problems in managing them.

VIII. Preliminary Analysis

A. Relationships Between Number of Developers, Comment Percentage, and Project Duration

In this section, we aim at establishing correlation between Number of Developers, Comment Percentage and Project Duration. These metrics helps us understand relative size of teams and efforts towards documentation and the period that projects were completed.

The pair plot in figure 3, presents the relationships among Number of Developers, Comment Percentage, and the Project Duration. The correlation between Number of Developers and Comment Percentage is visible which suggests that larger teams tend to produce more detailed documentation. However, there is weak relationship between Project Duration and Number of Developers which indicates that larger teams do not necessarily lead to shorter or longer timelines of project.

In figure 4, the correlation heatmap also shows the relationships between the Number of Developers, Comment Percentage, and the Project Duration. A strong positive correlation between Number of Developers and Comment Percentage

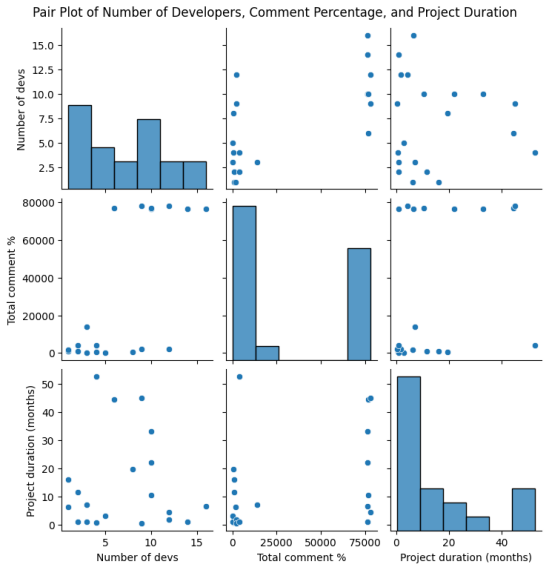

Fig. 3: Pair Plot of Number of Developers, Comment Percentage, and Project Duration

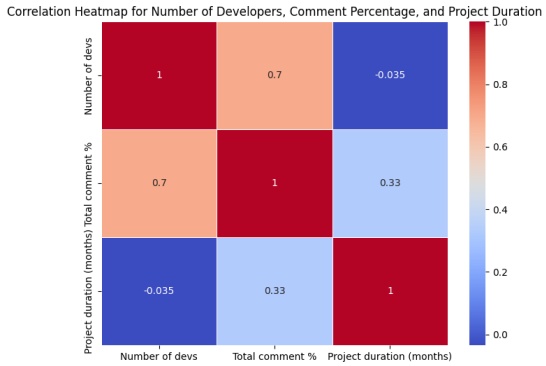

Fig. 4: Heatmap of Number of Developers,Project Duration, and Comment Percentage

highlights the impact of team size on documentation quality. However, the relationship between Project Duration and Number of Developers is weak, reinforcing that project length is not significantly influenced by the number of developers involved.

Discussion: Both visualizations show that team size has a notable effect on documentation quality but it does not significantly affect the length of the project.

B. Relationships Between Complexity Metrics and LOC

This section explores the relationships between the key complexity metrics: WMC (Weighted Methods per Class), RFC

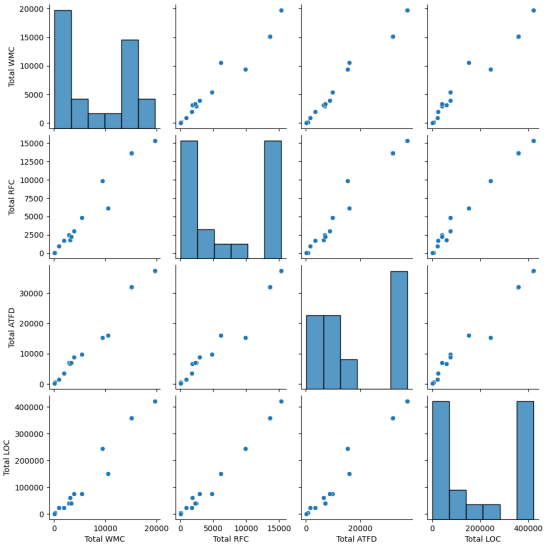

Fig. 5: Pair Plot of WMC, RFC, ATFD, and LOC

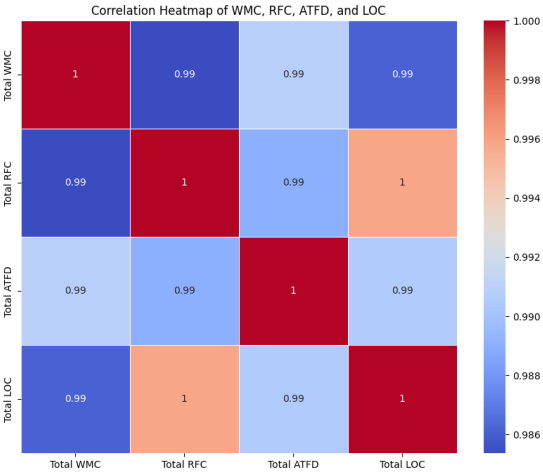

Fig. 6: Correlation Heatmap of WMC, RFC, ATFD, and LOC

(Response for a Class), and ATFD (Access to Foreign Data) and LOC (Lines of Code). It focusing on their interactions and how they affect code size and complexity.

The pair plot in figure 5, provides a detailed view of the pairwise relationships between WMC, RFC, ATFD, and LOC. Positive correlations are observed between WMC and RFC which indicates that if class complexity increases, method coupling also tends to increase. Also, the relationships between LOC and the complexity metrics (WMC and RFC) show that larger codebases tend to indicate higher complexity, which can make the system more challenging to maintain.

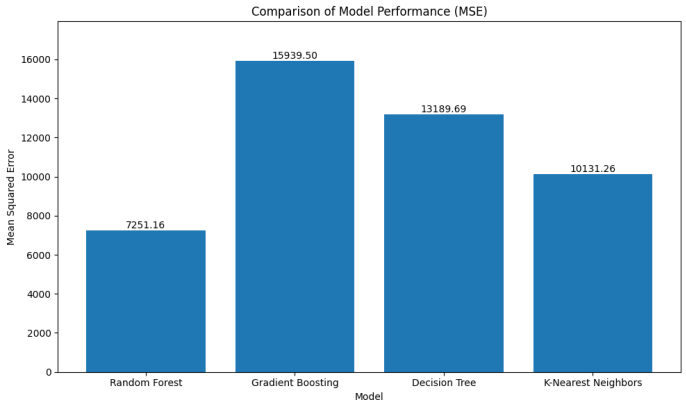

Fig. 7: Comparison of Model Performance (MSE)

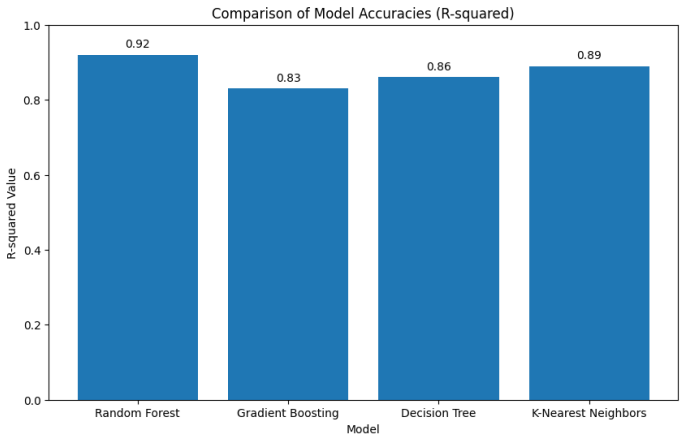

Fig. 8: Comparison of Model Accuracies (R-squared)

In figure 6, the correlation heatmap numerically presents the relationships between the four metrics using correlation coefficients. Strong correlations (near 0.99) are observed between WMC, RFC, ATFD, and LOC, confirming that these metrics are tightly coupled. As complexity increases, the codebase size (LOC) grows proportionally which highlights the importance of managing complexity early in the development process to maintain project scalability and prevent risks.

IX. Applications of Regression Models

The second analysis is to find relations between metrics. We tried to find some correlations between the software metrics. But in this case, we used the big dataset and found a positive result here.

In figure 7, the analysis of regression models for software quality prediction highlights the strengths and weaknesses of four algorithms: Random Forest Regressor, Gradient boosting regressor, Decision Tree Regressor K-Neighbors Regressor. However, Random forest regressor showed the highest accuracy with an R-squared of 0.92 and the best MSE of 7251.16 [9].. Therefore, it manages large data, illustrating why and how a function is non-linear, and shows which attribute of a given set influences the prediction.

In figure 8, the Gradient Boosting Regressor yields an R-squared of 0.83 and MSE of 15939.50: it generates models step by step and for each step it adjusts for the mistakes produced by the previous steps which makes it ideal to work with datasets that have both bias and variance. For a beginner analyst or decision support systems the Decision Tree Regressor provided an accuracy, or R-squared value of 0.86, but had an MSE of 13189.69. The algorithm of choice for this dataset was the K-Neighbors Regresser, which scored 0.89 of R-squared and Mean Squared Error of 10131.26, making it the best suited for capturing local distancing and dealing with results of scattered training data.

X. Conclusion

When this is done manually, the methods are expensive and time consuming, but serve to guarantee the quality of the product. This paper also focuses on the use of software metrics as a form of scientific and constructive approach to qualities such as correctness, scalability and bug tolerance. Using the common metric tools allows for consistent measurement; enhancing product quality, as well as cutting costs of creating and implementing the metric for organizations.

References

[1] M. S. Rawat, A. Mittal, and S. K. Dubey, "Survey on impact of software metrics on software quality," *IJACSA) International Journal of Advanced Computer Science and Applications*, vol. 3, no. 1, 2012.

[2] S. Chawla and G. Kaur, "Comparative study of the software metrics for the complexity and maintainability of software development," *International Journal of Advanced Computer Science and Applications*, vol. 4, no. 9, 2013.

[3] M. Agnihotri and A. Chug, "Application of machine learning algorithms for code smell prediction using object-oriented software metrics," *Journal of Statistics and Management Systems*, vol. 23, no. 7, pp. 1159–1171, 2020.

[4] Y. Suresh, J. Pati, and S. K. Rath, "Effectiveness of software metrics for object-oriented system," *Procedia technology*, vol. 6, pp. 420–427, 2012.

[5] M. Y. Mhawish and M. Gupta, "Predicting code smells and analysis of predictions: Using machine learning techniques and software metrics," *Journal of Computer Science and Technology*, vol. 35, pp. 1428–1445, 2020.

[6] Z. Jiang, P. Naudé, and B. Jiang, "The effects of software size on development effort and software quality," *International Journal of Computer and Information Science and Engineering*, vol. 1, no. 4, pp. 230–234, 2007.

[7] S. Gholizadeh, "Top popular python libraries in research," *Journal of Robotics and Automation Research*, vol. 3, no. 2, pp. 142–145, 2022.

[8] A. Khandare, N. Agarwal, A. Bodhankar, A. Kulkarni, and I. Mane, "Analysis of python libraries for artificial intelligence," *Journal Name*, 2023.

[9] G. Shanmugasundar, M. Vanitha, R. Čep, V. Kumar, K. Kalita, and M. Ramachandran, "A comparative study of linear, random forest and adaboost regressions for modeling non-traditional machining," *Processes*, vol. 9, no. 11, p. 2015, 2021.