IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.102 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 3, March 2024

DOI: 10.17148/IJARCCE.2024.13313

Recommendation System for Code Validation and Optimal Refactoring

Koteswara Rao Velpula[1], Hema Pavuluri[2], Poojitha Neeluri[3], Anushka Pappala[4], Mounika Narra[5]

Assistant Professor, Computer Science & Engineering, VVIT, Guntur, India[1]

Undergraduate, Computer Science & Engineering, VVIT, Guntur, India[2-5]

Abstract: This article describes about the project implementation of "Recommendation System for Validating code and Optimal Refactoring" and its outcomes for the problems mentioned. This project enhances coding practices by suggesting clean code snippets and with enhanced scoring mechanism. Initially, it identifies code issues using static code analysis APIs in languages like Java, Python, HTML, CSS and JavaScript. And then uses natural language processing techniques and libraries like NumPy and scikit-learn to recommend context-specific code solutions. This innovative system integrates with popular IDEs, supports multiple languages, can be customizable and developed on huge dataset training, significantly improving code validation and refactoring processes.

Keywords: Static code analysis APIs, Programming language linters, NumPy, Pandas, Scikit-learn, Collaborative Filtering Algorithm, NLP techniques (Stemming), Vector embeddings, Cosine similarity, TF-IDF algorithm.

I. INTRODUCTION

In an era dominated by software, the demand for clear, comprehensible code is paramount. Clean code, with its simplicity, maintainability, and adherence to best practices, the bedrock of quality applications. However, mastering these standards can be daunting, especially for beginners. Our innovative Recommender Systems for Validation and Optimal Refactoring aims to address this challenge. Designed for developers of all levels, it revolutionizes code writing, promoting cleanliness, efficiency, and maintainability. Through advanced algorithms and expert insights, our system empowers users to navigate code complexities with confidence. This paper unveils our system's inner workings, showcasing its capabilities and seamless integration into workflows. From novices seeking guidance to seasoned developers striving for excellence, our solution offers invaluable support. It paves the way for a new era of software craftsmanship and innovation.

A. *The Need for Clean Code:*

Code validation ensures that software functions as intended, reducing errors and enhancing reliability. Maintaining clean code simplifies debugging and troubleshooting, saving time and resources in the long run. By adhering to coding standards, developers improve readability and collaboration, fostering a cohesive development environment.

B. *Optimal Refactoring:*

Optimal refactoring leverages techniques rooted in natural language processing (NLP), such as TF-IDF (Term Frequency-Inverse Document Frequency) analysis and cosine similarity. By employing TF-IDF to identify key words and phrases in code snippets, refactoring processes can prioritize areas for improvement based on their relevance and significance. Additionally, cosine similarity measures the similarity between code snippets, enabling the system to recommend refactoring strategies based on the resemblance between different segments of code.

II. RELATED WORK

The foundation of our research lies in a comprehensive literature review, delving into pioneering studies conducted by esteemed researchers in the field. This segment meticulously examines seminal articles, elucidating their pivotal discoveries and setting the stage for our forthcoming investigations.

A. Recommender Systems: An Overview, Research Trends, and Future Directions - Pradeep Singh et al (2021):
The approach used in this research paper is collaborative filtering and it also encountered the limitations in code smell detection and the code quality maintenance.

B. A systematic review and research perspective on recommender systems - Deepjyoti Roy and Mala Dutta (2022): This research paper surveys and evaluates various recommender systems, focusing on their methodologies and inherent limitations. While the proposed approach leverages a collaborative filtering scheme enhanced by a Genetic Algorithm (GA), scalability presents a major obstacle to its broader application.

C. An Empirical Study on the Impact of Android Code Smells on Resource Usage - Jonatan Oliveira et al (2018): This paper offers insights into static code analysis and its integration with Android apps for error detection and code suggestion, facilitating enhanced code quality and reliability in the Android development ecosystem. It explores the role of static code analysis techniques in optimizing coding practices and streamlining the development process for Android applications.

III. EXISTING SYSTEM

Recommendation systems work based on the input given by the user, by selecting appropriate language and then writing some code. The focus is solely on generating code that gives more score when code is analyzed statically. These are following features of existing system.

A. *Language Support:*

The existing system offers limited language support, restricting developers to a narrow range of programming languages and limiting their ability to work with diverse technologies. Lack of support for multiple programming languages hampers productivity and versatility, forcing developers to rely on external tools or manual processes to work with unsupported languages.

B. *Static Code Analysis:*

The static code analysis techniques employed by the existing system are rudimentary and fail to provide comprehensive insights into code quality, leading to suboptimal results and increased risk of errors. Lack of advanced static code analysis capabilities limits the system's ability to detect subtle issues and vulnerabilities in the code, increasing the likelihood of software defects and security vulnerabilities.

C. *Recommendation Generation:*

The recommendation generation process in the existing system is rudimentary and lacks sophistication, resulting in generic and unhelpful code suggestions that fail to address developers' specific needs. The existing recommendation generation approach lacks transparency and explainability, making it difficult for developers to understand and trust the suggestions provided, thereby reducing adoption and usage.

D. *User Interface:*

Lack of intuitive design and user-centric features in the existing user interface contributes to frustration and confusion among developers, exacerbating usability issues and impeding workflow efficiency. The existing user interface fails to prioritize user feedback and incorporate user preferences, resulting in a disconnect between user needs and system capabilities, leading to dissatisfaction and disengagement.

E. *Integration:*

Limited enforcement of industry standards and best practices in the existing system results in code that is prone to errors, vulnerabilities, and suboptimal performance, undermining software reliability and stability. The absence of automated code quality checks and real-time feedback mechanisms in the existing system contributes to a reactive rather than proactive approach to code validation, increasing the risk of quality issues and technical debt accumulation.

IV. EXPERIMENT METHOD/PROCEDURE/DESIGN

Our system streamlines the coding process by offering a user-friendly interface where individuals can select their preferred programming language from a dropdown menu encompassing options such as Python, Java, HTML, CSS, and JavaScript. Once a language is chosen, users can input their code snippets, prompting the system to swiftly generate corresponding code suggestions accompanied by scores.

A. *Multilingual Support:*

Our system offers expanded language options, allowing users to work with a diverse range of programming languages, including Python, Java, HTML, CSS, and JavaScript. With support for multiple programming languages, users have the flexibility to work in their language of choice, enhancing productivity and versatility.

DOI: 10.17148/IJARCCE.2024.13313

B. ***Enhanced Accuracy:***

By leveraging advanced NLP, cosine similarity, and TF-IDF techniques, our system delivers more precise and reliable code suggestions tailored to individual user inputs. With a focus on precision and reliability, our system sets a new standard for code recommendation systems, optimizing the development process and minimizing errors.

C. ***Intuitive User Interface:***

With a streamlined interface, users can effortlessly navigate through the process of selecting and refining code snippets, enhancing overall usability and efficiency. By prioritizing simplicity and clarity, our interface fosters a positive user experience, encouraging adoption and usage among developers of all skill levels.

D. ***Comprehensive Feedback:***

Our system not only provides code snippets but also offers detailed scores and error messages, empowering users to make informed decisions for optimal code quality. The inclusion of detailed scores and error messages enhances transparency and accountability, allowing developers to track their progress and measure their success.

E. ***Continuous Improvement:***

Through ongoing updates and enhancements, our system remains at the forefront of code validation and refactoring, ensuring customisable for future works. With a focus on transparency and accountability, our system promotes a culture of continuous improvement and refinement, driving better outcomes and results.

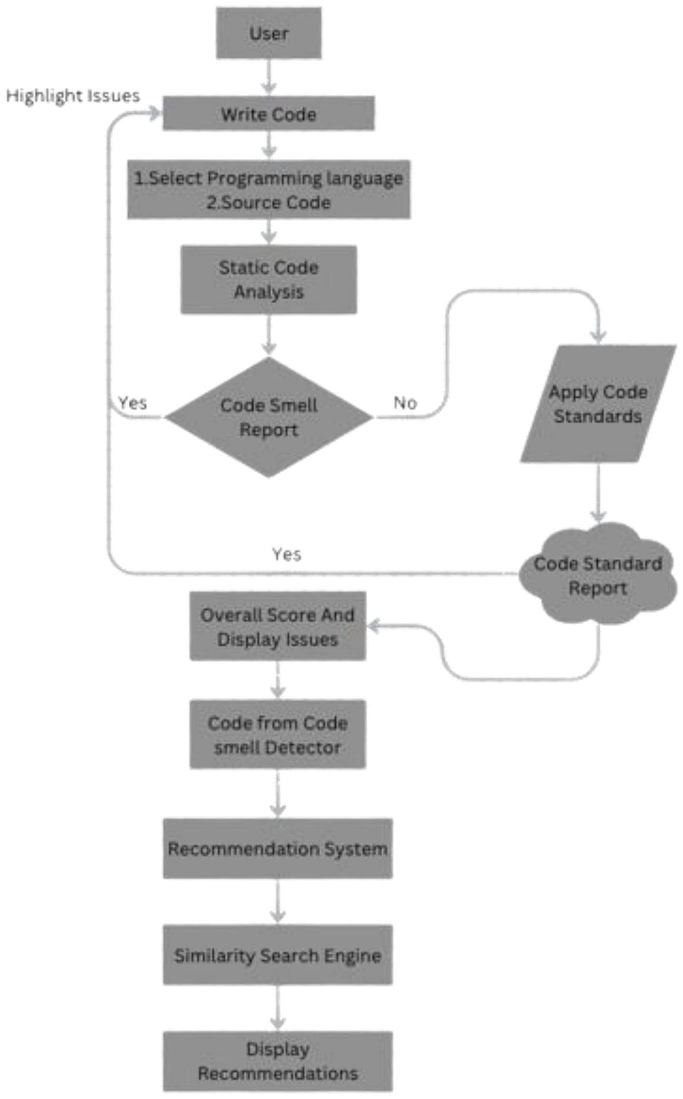


Fig.1: Proposed System with recommendation

DOI: 10.17148/IJARCCE.2024.13313

V. IMPLEMENTATION

The implementation of Recommendation system for code validation and optimized refactoring is as follows:

A. *Import Libraries:*
Essential Python libraries are imported, including pandas for data manipulation, NumPy for numerical computations, and scikit-learn for machine learning algorithms. Additionally, Selenium is imported for web scraping, and Streamlit is imported for building interactive web applications.

B. *Pre-Processing:*
The code pre-processes the input data, including cleaning and transforming it into a format suitable for further analysis. This step may involve handling missing values, encoding categorical variables, and scaling numerical features.

C. *Set-Up Linters:*
Linters such as pylint for Python, htmllint for HTML, csslint for CSS, and jshint for JavaScript are configured to analyse code quality and adherence to coding standards. These linters help identify and fix potential errors, inconsistencies, and style violations in the codebase.

D. *Loading pre-trained Models:*
Pretrained models and embeddings, such as word embeddings for NLP tasks or machine learning models for recommendation algorithms, are loaded into memory. These models are essential for performing various tasks like text similarity computation or generating code recommendations.

E. *Handling PKL Files:*
Pickle (PKL) files containing serialized objects, such as trained machine learning models or processed data, are loaded into memory. These files are used to store and retrieve complex Python objects efficiently, enabling faster execution and reduced memory usage during runtime.

F. *Implementing Algorithms:*
Algorithms for code validation, recommendation, and refactoring are implemented using appropriate techniques like NLP, cosine similarity, or TF-IDF. These algorithms analyse code snippets, identify patterns, and provide recommendations for improving code quality and readability.

$$\cos(a,b) = \frac{\|a\|\|b\|}{\sqrt{\sum_{i=1}^{n}(a_i)^2} \times \sqrt{\sum_{i=1}^{n}(b_i)^2}}$$

$$\text{tf-idf}(t,d,D) = \text{tf}(t,d) \times \text{idf}(t,D) \quad (1)$$

$$\text{tf}(t,d) = \log(1 + \text{freq}(t,d)) \quad (2)$$

$$\text{idf}(t,D) = \log \frac{N}{|\{d \in D : t \in d\}|} \quad (3)$$

where t is a term, d is a document, D is a corpus, $\text{tf}(t,d)$ is the term frequency of t in d, and $\text{idf}(t,D)$ is the inverse document frequency of t in D.

G. *Integrating Web Scraping:*
Web scraping techniques are employed to gather additional data or insights from online sources, such as code repositories or developer forums. Selenium WebDriver is utilized to automate web browser interactions and extract relevant information for analysis.

H. *Developing Streamlit Interface:*
A user-friendly interface is developed using Streamlit, allowing users to interact with the recommendation system effortlessly. The interface displays code snippets, recommendations, and analysis results in a visually appealing and intuitive manner.

DOI: 10.17148/IJARCCE.2024.13313

I. ***Testing and Validation:***
The implemented system undergoes rigorous testing and validation to ensure its functionality, accuracy, and robustness. Unit tests, integration tests, and user acceptance tests are conducted to identify and rectify any issues or bugs in the system.

J. ***Deployment and Maintenance:***
Once tested and validated, the system is deployed to a production environment, making it accessible to users. Continuous monitoring, updates, and maintenance are performed to keep the system operational, efficient, and up to date with evolving requirements and technologies.

VI. RESULTS AND DISCUSSION

After implementation, our proposed system excels in recommending code snippets tailored to the user's preferences across multiple programming languages. Below, you'll find a seamless display of these recommendations along with their corresponding scores, simplifying the process of selecting the most suitable snippets for your coding needs.

A. ***Multiple Language Support:***
Our system accommodates a diverse range of programming languages, ensuring that developers can receive recommendations and scores tailored to their specific language preferences.

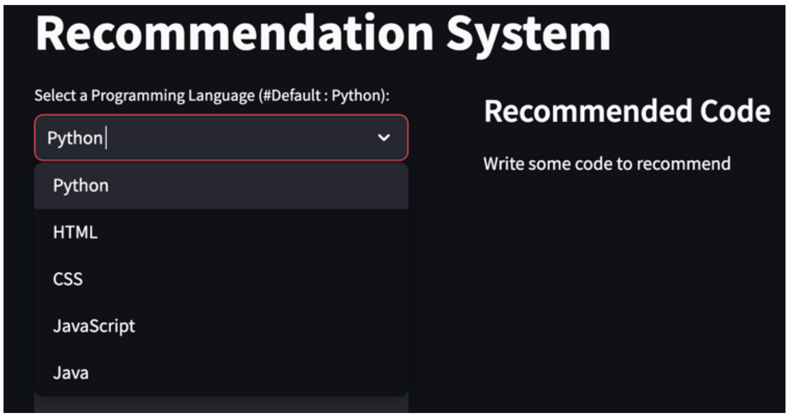
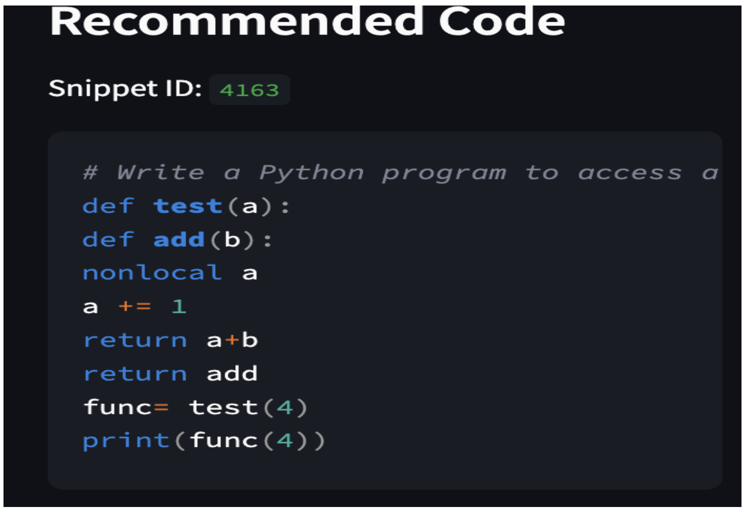

Recommendation System
Select a Programming Language (#Default : Python):
Python
Python
HTML
CSS
JavaScript
Java
Recommended Code
Write some code to recommend

Fig.2: Recommendation System Languages

B. ***Recommended Code and Score Depiction:***
Users are presented with a comprehensive display of recommended code snippets alongside their corresponding scores, facilitating informed decision-making and code selection.



```
# Write a Python program to access a
def test(a):
def add(b):
nonlocal a
a += 1
return a+b
return add
func= test(4)
print(func(4))
```

Fig.3: Recommended Code Output

C. ***Accuracy Results:***

The system delivers accurate and reliable recommendations, empowering developers with the confidence to choose high-quality code snippets for their projects.

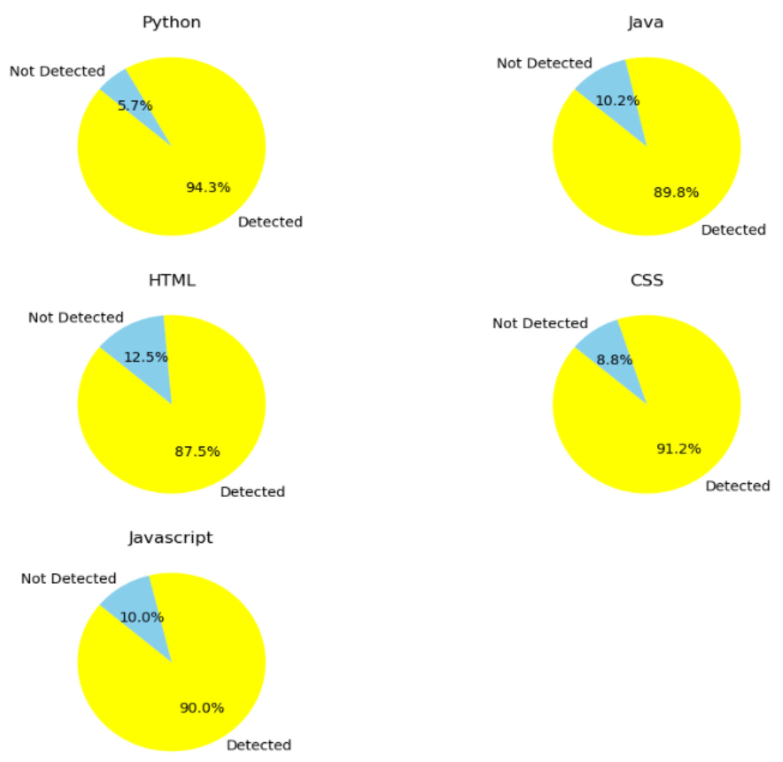


Fig.4: Individual Accuracy

D. ***Dataset Evaluation:***

In the dataset, each entry comprises a snippet ID and the corresponding code snippet. This structured format facilitates efficient storage and retrieval of code samples, enabling seamless integration with the recommendation system for code validation and optimal refactoring.

VII. CONCLUSION AND FUTURE SCOPE

In conclusion, the Recommendation System for Code Validation and Optimal Refactoring offers tailored solutions to enhance code quality and streamline the refactoring process. Leveraging advanced techniques such as NLP, cosine similarity, and TF-IDF word embeddings, the system empowers developers to make informed decisions when optimizing their codebases. While the system demonstrates impressive efficacy in identifying and rectifying common coding errors, continuous refinement and expansion are essential to keep pace with evolving industry standards and coding practices. Collaboration with industry stakeholders and ongoing research endeavours are pivotal in ensuring the system's relevance and effectiveness in real-world development scenarios.

Looking forward, the project holds immense potential for growth and innovation. Firstly, expanding language support beyond the current repertoire presents an exciting opportunity to cater to a wider audience of developers and projects, enhancing the system's versatility and applicability. Additionally, integration with prominent software development platforms and projects can greatly amplify the impact of the system, fostering widespread adoption and facilitating seamless code validation and refactoring workflows on a larger scale. Furthermore, exploring the integration of generative AI algorithms opens new avenues for automating code generation and refinement, revolutionizing the coding process and empowering developers to tackle complex challenges with greater efficiency and precision. By embracing these avenues of expansion and innovation, the project can continue to lead the way in advancing software engineering practices and driving meaningful progress in the field.

A. ***Conflict of interest:***
The authors declare no conflicts of interest. They have no personal or financial affiliations with any groups or companies that could potentially influence the study's outcomes. There are no competing interests to report, and the research is entirely self-supported.

B. ***Funding source:***
The research was entirely self-funded by the author(s), with no external organizations or funding sources involved. Financial support for this work came solely from the authors' own resources, without any contributions from external entities or agencies.

C. ***Authors' Contribution:***
The success of the Recommendation System for Code Validation and Optimal Refactoring project was achieved through the collaborative efforts of multiple authors, each making significant contributions to different aspects of the research and development process.

Author 1:

- Benefited from mentorship and guidance throughout the project, leveraging the collective expertise of all authors to tackle complex challenges and refine the recommendation system.
- Ensured the project's alignment with industry standards and best practices, contributing to the credibility and relevance of the recommendation system in real-world software development scenarios.

Author 2:

- Developed the core recommendation algorithm leveraging advanced NLP techniques such as cosine similarity and TF-IDF word embeddings.
- Engineered algorithms for code analysis and refactoring suggestions across various programming languages.
- Utilized Python libraries like NLTK and scikit-learn to implement robust NLP models and optimize algorithm performance.

Author 3:

- Implemented the pre-processing pipeline for cleaning and tokenizing code snippets, ensuring compatibility with NLP algorithms.
- Designed and developed the user interface for the recommendation system, enhancing user interaction and feedback mechanisms.
- Integrated user feedback mechanisms to continuously improve the recommendation accuracy and user experience.

Author 4:

- Conducted extensive experimentation and evaluation of the recommendation system, analysing the effectiveness of different NLP techniques and parameters.
- Collaborated closely with Author 1 to fine-tune the recommendation algorithm based on performance metrics and user feedback.
- Contributed to optimizing system resources and scalability to handle large code repositories efficiently.

Author 5:

- Took charge of documenting the system architecture, algorithms, and implementation details, ensuring comprehensive documentation for future reference and maintenance.
- Led efforts to address scalability and performance bottlenecks, optimizing code execution speed and resource utilization.
- Played a key role in validating the system's recommendations through thorough testing and validation against diverse codebases and use cases.

VIII. ACKNOWLEDGEMENTS

We extend our heartfelt appreciation to all those who played pivotal roles in the successful completion of this project. Our profound gratitude goes to our college for fostering an environment conducive to creativity and scholarly exploration. We are deeply thankful to **Mr. V. Koteswara Rao sir** and **Dr. N. Sri Hari sir** for their invaluable advice, support, and extensive knowledge, which significantly influenced the development of this project.

Additionally, we wish to express our gratitude to the developers of the technologies and libraries used in our project, as well as the open-source community, whose contributions laid the groundwork for our work.

We extend our sincere thanks to everyone who contributed to the project's success, whether through direct involvement, feedback, or encouragement. Your collective efforts have been instrumental in achieving our goals, and we are truly grateful for your support.

REFERENCES

[1]. Fabio Palomba et al, “Do they really smell bad? A study on developer’s perception of bad code smells”, pp.1-10, doi: 10.1109/ICSME.2014.32, 2014.
[2]. Roberto Oliveira et al., “Identifying Code Smells with Collaborative Practices: A Controlled Experiment”, pp. 61–70, doi: 10.1109/SBCARS.2016.18, 2016.
[3]. Naoya Murakami and Hidehiko Masuhara, “Optimizing a Search-Based Code Recommendation System”, in: RSSE ’12, Zurich, Switzerland: IEEE Press, pp.68–72, Isbin: 9781467317597, 2012.
[4]. Pradeep Singh et al., “Recommender Systems: An Overview, Research Trends, and Future Directions”, in: International Journal of Business and Systems Research, Vol. 15, pp. 14–52, doi: 10.1504/IJBSR.2021.10033303, Apr. 2021.
[5]. Deepjyoti Roy and Mala Dutta, “A systematic review and research perspective on recommender systems”, in: Journal of Big Data, Vol. 9, Issue 1, p. 59, issn: 2196-1115, doi: 10.1186/s40537-022-00592-5, 2022.
[6]. Johnatan Oliveira et al. “An Empirical Study on the Impact of An-Droid Code Smells on Resource Usage”. doi: 10.18293/ SEKE2018-157, April 2018.
[7]. S.K. Pandey and A.K. Tripathi, “An Empirical Study towards dealing with Noise and Class Imbalance issues in Software Defect Prediction”, in: Soft Comput. Vol. 25, pp. 13465–13492, 2021.
[8]. M.M. Draz M.S. Farhan, S.N. Abdulkader, and M.G. Gafar, “Code smell detection using whale optimization algorithm”, in: Comput. Mater. Contin., Vol. 68, pp. 1919–1935, 2021.
[9]. Ali Ouni et al., “MORE: A multi-objective refactoring recommendation approach to introducing design patterns and fixing code smells”, in: Journal of Software: Evolution and Process, Vol. 29, Issue 5, pp.12-14, doi: https://doi.org/10.1002/smr.1843/, 2017.
[10]. Kaur and A. Kaur, “A Novel Four-Way Approach Designed with Ensemble Feature Selection for Code Smell Detection”, in: IEEE Access, Vol. 9, pp. 8695–8707, 2021.
[11]. Fontana, F.A.; Zanoni, M. Code smell severity classification using machine learning techniques. Knowl. Based Syst. 2017, 128, 43–58.
[12]. Draz, M.M.; Farhan, M.S.; Abdulkader, S.N.; Gafar, M.G. Code smell detection using whale optimization algorithm. Comput. Mater. Contin. 2021, 68, 1919–1935.
[13]. Gupta, H.; Kulkarni, T.G.; Kumar, L.; Neti, L.B.M.; Krishna, A. An Empirical Study on Predictability of Software Code Smell Using Deep Learning Models; Springer: Cham, Switzerland, 2021.
[14]. Di Nucci, D.; Palomba, F.; Tamburri, D.A.; Serebrenik, A.; de Lucia, A. Detecting Code Smells using Machine Learning Techniques: Are We There Yet? Proceedings of the 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER), Campobasso, Italy, 20–23 March 2018.
[15]. Yadav, P.S.; Dewangan, S.; Rao, R.S. Extraction of Prediction Rules of Code Smell using Decision Tree Algorithm. In Proceedings of the 2021 10th International Conference on Internet of Everything, Microwave Engineering, Communication and Networks (IEMECON), Jaipur, India, 1–2 December 2021; pp. 1–5.
[16]. Frederic Green. “The Book Review Column”. In: ACM SIGACT News 51 (3 2020).
[17]. Khalid Alkharabsheh et al. “Exploratory study of the impact of project domain and size category on the detection of the God class design smell”. In: Software Quality Journal 29 (2 2021).
[18]. “Mining Software Repositories for Automatic Interface Recommendation”. In: Scientific Programming 2016 (2016).
[19]. Girish Suryanarayana, Ganesh Samarthyam, and Tushar Sharma. Refactoring for Software Design Smells. 2014.
[20]. Khalid Alkharabsheh et al. “Exploratory study of the impact of project domain and size category on the detection of the God class design smell”. In: Software Quality Journal 29 (2 2021).