International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys. #, No.# (Mon-20..)

The Development of the Secure Quality Dataset (SQDS): Combining Security and Quality Measures Using Deep Machine Learning for Code Smell Detection

Hiba M. Yahya [1] , Dujan B. Taha[2]

[1] Software Department, University of Mosul
Mosul, Iraq
[2] Computer Department, University of Mosul
Mosul, Iraq

E-mail address: hibamoneer@uomosul.edu.iq , dujan_taha@uomosul.edu.iq

Received ## Mon. 20##, Revised ## Mon. 20##, Accepted ## Mon. 20##, Published ## Mon. 20##

Abstract: Code smells are an indication of deviation from design principles or implementation in the source code. Early detection of these code smells increases software quality by using refactoring techniques that will help the developers in software engineering maintain the process of software. Security is included as one of the requirements of software artifact quality in the ISO/IEC 25010 standard so we thought the security in the design phase is more efficient than after delivery of the software to the customer. A study aims to create a new dataset containing security metrics besides the quality metrics that will help software engineering researchers by detecting both the presence of a security illusion and god class bad smell at the same time in a program, we take Fonata's dataset of god class that have 61features of quality metrics, then calculate the security metrics on these 74 software written in java by programming a parser to analyze each software, finally used five machine learning algorithms on the proposed datasets (SQDS), after that, we used accuracy performance metric was employed for comparing the results. The experimental findings suggest that the proposed dataset demonstrates superior performance in identifying code smell security vulnerability and augmenting the training data can improve the accuracy of predictions. Finally, we applied three deep machine learning (RNN, LSTM, and GRU) on both the original Fonata's Dataset of God Class bad smell and our proposed SQDS dataset and made a comparison between them.

Keywords: ***Security Metrics, God Class bad smell , Quality metric , Machine Learning , Deep learning***

E-mail:author's email

1. INTRODUCTION

Security is included as one of the requirements for software product quality in the ISO/IEC 25010 standard. According to this standard, security refers to which the degree a product or system can protect its data to ensure that different goods or people may access appropriate data according to their categories and authorizations [1],[2].The eight quality attributes make up the ISO/IEC 25010-defined product quality model are : functional suitability , performance efficiency , compatibility ,usability , reliability , security , maintainability and portability .

Nowadays, the majority of software systems must meet security requirements [2],[3]. Nevertheless, not all security problems can be resolved by traditional software metrics [4], resulted in the creation of several software systems that are hazardous [5]. Early on in the software development process, security concerns should be given more importance. The majority of developers and organizations often believe that security should be included after a system is developed [6]. For maximum efficiency and effectiveness, security should be taken into consideration early in the development process [6, 7, 8]. To safeguard their systems, the majority of businesses invest a significant amount of money in purchasing firewall and antivirus software [2], [9].

Security mean the extent to which a system or product secures information and data so that users or other systems or products can access it to the right extent depending on the kinds and degrees of permission. The following sub-characteristics make up this characteristic:

• Confidentiality: The extent to which a system or product guarantees that data are only accessible by those who are permitted access.

Integrity refers to how well a system, product, or component guards against illegal access to or alteration of data or computer programs.

• Non-repudiation: The extent to which deeds or events can be demonstrated to have occurred and so cannot be subsequently denied.

• Accountability: The extent to which an entity's activities may be directly linked to it.

• Authenticity: The extent to which it is possible to verify that a resource or topic is who they say they are.

Code smells can arise from any modifications made to the source code that go against the principles of software design. Code smells are defects in design or changes made by developers that may have an impact on future system quality and cause challenges with maintenance. Code smells may lead to technical debt and the degradation of software projects if they are not addressed. Code smells can therefore be used as a sign to determine whether the source code needs refactoring [10]. The first step in the code refactoring process is to find bad smells in the code. Code scent detection methods often depend on object oriented metrics as inputs to identify code smells in software projects. Many different tools for static analysis and code reorganization techniques have been established that carefully examine the source code in order to find and fix problems [11].

Machine learning approaches involve the training of supervised models using data extracted from the same or a different software project. To model the source code components, metrics are used, similar to heuristic-based approaches. However, ML approaches differ in that they do not necessitate the specification of threshold values. Instead, they depend on data-driven learning to determine whether a particular code component is categorized as "smelly" or "non-smelly".

Supervised learning algorithms, such as recurrent neural networks (RNNs) , have been responsible for the remarkable progress in deep learning in recent years. RNNs are currently active in various practical applications like text generation, auto-translation, speech recognition, and code smell detection.[12]

The primary objective of this research is to introduce new dataset contain the security metrics of 74 software system in Qualitas Corpus[13] that Fonata is used it and calculate four types of bad smell to made a dataset contain quality metrics . Five machine learning algorithms used on our proposed SQDB dataset that take the God Class bad smell, evaluate its performance based on accuracy metric. After that we made a comparison between the original Fonta ‘s dataset and our proposed SQDS dataset depending on using performance metrics (accuracy , precession , recall and f1-score) by applying three deep machine learning (RNN , LSTM and GRU). Our paper is follows this structured: Section two discusses the related works, section three provides a background on detection strategies for software security metrics, code smells detections and deep machine learning. Methodology of the research is obtainable in section four. Sections five and six presents the experimental results and subsequent discussions and finally conclusions section.

2. RELATED WORK

After 1999, when Fowler et al. [14] published a book that outlined various bad code smells and the corresponding refactoring techniques, research into detecting these code smells began in the field. Numerous literature reviews and surveys have been carried out in the domain of code smell identification and refactoring [15-18]. These investigations have demonstrated several methods and tactics for identifying poor code smell in current software systems through the utilization of machine learning methods [19].

By rearranging internal design elements to remove system vulnerabilities, software refactoring may be utilized to increase software system security.

Refactoring is a technique for reorganizing software's internal architecture without affecting its functionality [14]. Numerous investigations were carried out to gauge the design's early weaknesses.

In [20] the researchers looked at whether complexity is detrimental to software security using statistical analysis. The results showed that the software's Complexity had a major influence on security.

In [21] the researchers advanced a fixed of safety metrics for the object-oriented layout that could enable designers to

stumble on and cope with safety flaws at some point of the design phase. These metrics can useful resource in evaluating the safety of diverse layout versions. In specific, seven protection metrics were proposed that may degree the concord and encapsulation.

Logistic regression changed into used to expect vulnerabilities using software metrics, zimmermann et al. Carried out experiments on Windows Vista and found that these metrics could be used to expect a number of the software program's vulnerabilities. They also analyzed the relationship between software program metrics, together with complexity and vulnerabilities. [22]

An empirical study by Abid et al. [23] to validate the relationships between several security dimensions (CIDA, CCDA, COA, CMAI, 2019).

CAAI, CAIW, CMW, VA, and Avg Security), and to explore correlations between security metrics and refactoring strategies.

Almogahed et al. discussed how software refactoring has been used to improve software security. They found that software systems with low coupling, low complexity and high compatibility are more secure and vice versa .[24]

Romeo L. [25] A prototype utilizing neural networks, machine learning, and deep learning for code smell detection was developed and implemented using the Python programming language. Subedi [26] suggested a method to collect, process and analyze code smells of different open-source projects and detect code smells in an intelligent way using the LSTM machine learning model. Sharma et al.[27] used CNN and RNN as their major hidden layers along with auto encoder model . They perform training and assessment on C# examples and Java code. Mhawish et al.[28] proposed was an approach for predicting code smells using machine learning techniques and software metrics, which incorporated the Local Interpretable Model-Agnostic Explanations (LIME) algorithm to improve comprehension of the machine learning model's decision-making process, and to identify the specific features that have an impact on the prediction model's decisions.

3. BACKGROUND

Software development has seen an increase in study in recent years with the goal of improving code quality, detecting code smells, and strengthening security protocols. Numerous research works have proposed a wide range of methods and strategies for identifying bad code smells and evaluating the security and quality metrics of software systems. With an emphasis on the use of deep machine learning, this section offers a concise synopsis of the pertinent data on the tactics used in the identification of security metrics and code smells. Early in the software development life cycle, researchers and practitioners have realized how important it is to take proactive steps to find and fix code-related problems. To keep software systems robust and reliable, it is essential to investigate various methods for detecting code smells and to evaluate security metrics. Furthermore, the integration of deep machine learning into these detection algorithms has surfaced as a viable approach, providing the possibility of more precise and effective code anomaly identification. The dynamic field of software engineering emphasizes the necessity of ongoing enhancements to security and code quality procedures. Through an exploration of the nexus of deep machine learning, security metrics, and code smell detection, this review seeks to illuminate the state-of-the-art strategies that support the progress of software development processes. Further investigation reveals that the secret to resolving the complex issues related to code quality and security in the ever-changing field of software engineering is the use of state-of-the-art technologies, such deep machine learning.

Software Security metrics

In the context of software system development, addressing attack capabilities has become increasingly crucial due to the growing threat of software assaults. There is growing agreement that software metrics are useful instruments for estimating and rating program quality. Metrics can provide insights that enable the creation of useful prediction models, directing the development of software products, by quantitatively measuring important aspects of software systems. The best course of action is to prioritize security measures early on in the software development process, especially during the design phase [3].

It is impossible to overestimate the importance of protecting software against potential assaults in the modern digital era, where cyber dangers are pervasive. Metrics play a crucial role in assessing and forecasting the overall security posture and quality of software systems as they get more complex. Developers may proactively uncover vulnerabilities and create robust, resilient software solutions by methodically recording and analyzing metrics.

When properly utilized, software metrics serve as a preventative measure that enables developers to see any vulnerabilities early on and address them. This method improves the software's security while also enhancing the overall effectiveness and dependability of the finished result. Metrics-driven security practices are being integrated in line with the industry's general move toward a proactive security posture, which recognizes that predicting and preventing vulnerabilities is just as important as responding to them.

Furthermore, software metrics' predictive quality goes beyond security issues to provide a thorough grasp of the program's functionality, maintainability, and scalability. Development teams may make well-informed decisions, use resources wisely, and expedite the development process with the help of this comprehensive perspective. In keeping with the idea of "secure by design," security measures are applied at the design stage, guaranteeing that security considerations are included in the software's basic architecture.

The relationship between metrics and security becomes increasingly important as the software development landscape changes in order to provide software that is safe, reliable, and of high quality. This paradigm change emphasizes how crucial it is to take a proactive, metrics-driven strategy in order to reduce possible risks and

strengthen software systems against the always changing threats. To sum up, in order to effectively navigate the problems presented by software assaults and guarantee the creation of reliable and secure software solutions, it is essential to incorporate software metrics as predictive tools and to adopt security measures early on.

As stated by the National Institute for Using technology and standards to eliminate vulnerabilities during Up to thirty times as much can be saved during the design process as subsequently fixes , software security metrics are therefore required to measure the system's security straight from its layout. Determining the metrics for software security is essential to lowering risks and vulnerabilities related to system security [21]. A single object-oriented class's security level may be measured using the security design metrics, Lower values indicate a more secure program architecture. The measurements have all been scaled to fall within the range of 0 to 1. (Therefore, the measurements may be used to gauge a design's level of vulnerability.) According to a certain software security design concept (for example Least Privilege and Reduce Attack Surface), their findings indicate whether alternative designs may strengthen or weaken the security of a given class [29].

TABLE 1. Show the security metrics that will be calculated in our dataset [21]

Security Metric	Definition
CIDA	The ratio of the quantity of public characteristics for a classed instance to the quantity of characteristics in a class that are categorized.
CCDA	The relation of the quantity of public characteristics belonging to a categorized class to the quantity of characteristics in a class that are categorized.
COA	It is calculated by dividing all privately shared methods in a class by all publicly shared methods
CMAI	The relation between the total number of mutators that could potentially interact with classified attributes and the number of mutators that actually could
CAAI	The measurement of the number of accessors that can interact with the classified attributes is the maximum number of accessors that can have access to the classified attributes
CAIW	The ratio of all possible interactions with part attributes and all possible pathways to all attributes.
CMW	Equal to the ratio of classified methods divided by the number of methods in a class

Code Smell Detection

A crucial component of software development is code smell detection, which finds and fixes troublesome patterns or structures in source code. These "smells" are signs of possible inefficiencies or design defects that might lower the overall quality of the program and make it harder to maintain. Developers may systematically find certain code smells, such duplicate code, lengthy procedures, or inconsistent naming conventions, by utilizing a variety of static code analysis approaches. Enhancing code readability, maintainability, and scalability is the fundamental objective of code smell detection, which helps to build more reliable and effective software systems. As software projects get more complicated, it is critical to find and remove code smells early on to ensure long-term sustainability and facilitate development team communication.

Code smell detection is typically created on a grouping metrics of object oriented and predefined threshold value , aimed at identifying the main indications that define the code smells [30] . A variety of detection approaches rely on heuristics and detection rules that compare metric values obtained from source code with empirically established thresholds, in order to differentiate between code artifacts affected by a particular type of smell and those that are not. The choice of appropriate threshold values is crucial to the performance of detectors since it strongly influences their effectiveness. Hence, identifying suitable typical thresholds is a crucial factor in developing effective detection strategies. The code smells that will be used in our research is **God class** we can defined it as an anti-pattern in software design where a single class has too much responsibility and becomes overly complex. It tends to make the code difficult to maintain and modify. Such a class often contains excessive code, multiple methods, and tightly coupled dependencies, leading to high coupling and low cohesion.

Deep Machine Learning

A department of artificial intelligence referred to as "deep gadget learning" uses sophisticated neural networks and algorithms to evaluate and study from massive amounts of records. The time period "deep" describes the use of numerous layers of neural networks to investigate input, main to the introduction of increasingly complicated and correct models. With using this era, gadget learning will undergo a revolution as computer systems could be able to discover patterns, categorize information, and generate predictions which are extra correct than earlier than. Deep system getting to know has numerous uses, which includes photograph identification, predictive analytics, and herbal language processing [31].

Deep learning's transformational power is demonstrated by its capacity to automatically extract complex characteristics from data, enabling more accurate and nuanced decision-making. We expect a paradigm change in a number of areas as this technology develops, including autonomous systems, financial forecasts, and medical diagnostics. Because of its versatility and ability to manage intricate data structures, the deep learning framework is a key component in solving problems in the real world.

Furthermore, by incorporating deep learning into disciplines like computer vision, advances in object and image detection have been made, greatly augmenting the power of automated systems. Another aspect of deep learning is predictive analytics, which helps businesses make data-driven choices by predicting trends and seeing possible opportunities and hazards.

Deep learning applications rely heavily on natural language processing, which has advanced to the point that robots can now understand, interpret, and produce language that is similar to that of humans. This will completely change the

way we engage with technology and have a significant impact on sentiment analysis, language translation, and chatbots.
To sum up, the numerous uses of deep learning and its ability to completely transform a variety of sectors highlight how important it is to the development of machine learning. Deep learning is expected to have a significant influence on artificial intelligence and redefine the potential for data-driven decision-making as we move further into this period of technological progress.
RNNs are a selected form of deep neural network that are used to deal with sequential records by using retaining contextual knowledge from in advance inputs. However, RNNs have a trouble referred to as vanishing gradients, while the gradients used to replace the network's parameters grow to be too tiny and cause the network to cease gaining knowledge of. Traditional feed ahead neural networks system inputs one by one. In order to address this issue, some of RNN modifications had been proposed, which include Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRUs), which use greater strategies to higher manipulate the input waft via the community. Long Short-Term Memory, or LSTM, is an RNN architectural kind that is employed in deep studying. The aim of LSTM networks is to deal with the standard RNN's vanishing gradient trouble.
With the addition of specialized additives known as reminiscence cells. Three gates make up each reminiscence mobile: the enter, output, and forget gates. These gates will control the records flow into and out of the reminiscence cell and decide what information should be remembered and what should be deleted. The input gate manages the waft of clean information into the reminiscence cellular, while the output gate regulates the memory mobile's output to the network's next layer. In order to determine whether or not statistics have to be eliminated from the reminiscence cellular, the forget about gate is critical [32][33].

4. METHOD

The proposed dataset, SQDS, is being developed through a methodical procedure that includes many crucial components. First, 74 open-source Java systems will be downloaded from the Qualitas Corpus software repository in order to obtain data. In order to enable thorough analysis, we will utilize a specially designed parser that is designed to methodically examine the classes included in every software. This parser, which focuses on measurable elements like the quantity of public and private properties and methods within each class, will be crucial in helping to tidy up the code by eliminating comments and blank lines.
The next step is to take the parsed code , extract and compute seven different security metrics. This methodical technique guarantees a comprehensive evaluation of the security features integrated into the software systems. We want to combine these security measures with the Fonata god class dataset, an extensive collection of 62 quality indicators, in order to further enhance the dataset. Combining these datasets which are shown in Table2 will yield a comprehensive picture that includes quality and security measures.
Our methodology aims to generate a more comprehensive and nuanced view of the software systems under examination by fusing security indicators with an established quality dataset. This combination allows for a more thorough analysis that takes into account factors of overall code quality as well as security. It is expected that the resultant SQDS dataset will be a useful tool for both practitioners and scholars, providing insights into the complex interactions that exist between security and quality measures in open-source Java systems. This project is in line with the overarching objective of improving dataset richness and enabling more reliable analyses in the fields of security research and software engineering.

TABLE 2. Show quality metrics in Fonata's Dataset

Size	Complexity	Cohesion	Coupling	Encapsulation	Inheritance
LOC	CYCLO	LCOM5	FANOUT	LAA	DIT
LOCNAMM*	WMC	TCC	ATFD	NOAM	NOI
NOM	WMCNAMM*		FDP	NOPA	NOC
NOPK	AMWNAMM*		RFC		NMO
NOCS	AMW		CBO		NIM
NOMNAMM*	MAXNESTING		CFNAMM*		NOII
NOA	WOC		CINT		
	CLNAMM		CDISP		
	NOP		MaMCL§		
	NOAV		MeMCL§		
	ATLD*		NMCS§		
	NOLV		CC		
			CM		

With 420 rows and 68 columns that indicate various attributes, the SQDS dataset has an extensive structure. The data then goes through an important preprocessing step that is designed to convert unprocessed data into a format that can be analyzed. To do this, the data must be carefully cleaned to remove any missing values and any modifications such as scaling or normalization. This kind of preprocessing is essential because it improves the general efficacy and quality of the data analysis that follows, producing results that are more precise and trustworthy.
To identify code smells and assess the software's security, we employ five different machine learning techniques: Decision Tree, Random Forest, SVM, KNN, and Logistic Regression. A careful division of the dataset into training and testing sets is made. Models are trained on the assigned training set during the training phase, and their performance is evaluated using the testing set. Strict assessment is essential to guaranteeing the model's effectiveness. This paper presents a technique that compares the effectiveness of two methods for identifying security flaws and code smells.
Three deep machine learning algorithms(RNN, LSTM, and GRU) are applied to the original Fonata's dataset as well as our suggested SQDS dataset in order to further deepen the scope of our research, as shown in Fig 1. The use of deep learning techniques holds the potential to reveal complex patterns and subtleties present in the datasets, hence

advancing a comprehensive comprehension of code quality and security in software systems. The thorough assessment of these models and approaches is essential to pushing software engineering and security research forward and offers insightful information to both scholars and practitioners.

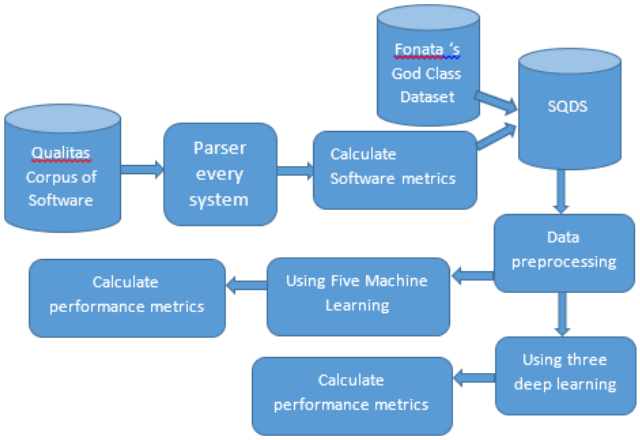

Figure 1. Proposed process for SQDS

5. THE RESULT

This study uses typical accuracy performance criteria generated from the confusion matrix to assess the efficacy of the SQDS. A fundamental technique for evaluating a model's performance in classification is the confusion matrix. It forms the foundation for a number of performance indicators by methodically classifying predictions into true positives, true negatives, false positives, and false negatives. A key indicator called accuracy evaluates how accurate the model's predictions are overall. The ratio of successfully predicted instances to all occurrences in the test dataset is used to compute it. Recall evaluates the model's capacity to catch every positive event, whereas precision examines the accuracy of positive predictions. An impartial assessment is given by the F1 score, which is the harmonic mean of recall and accuracy. The investigation also explores specificity and sensitivity, which center on accurately identifying negative and positive examples, respectively.

A thorough understanding of the accuracy performance of each of the five machine learning techniques (Logistic Regression, Decision Tree, Random Forest, SVM, and KNN) is provided by the performance analysis, which is displayed in Table 3. Through close examination of these metrics, researchers may obtain valuable insights into the advantages and disadvantages of each model, assisting in the identification of the best method for code smell detection and security assessment.

The study also looks at the consequences of false positives and false negatives because these occurrences are important in practical applications. While false negatives can provide serious security issues, false positives may result in needless actions or alarms. Practitioners obtain a deeper knowledge of the models' practical utility by comprehending the subtleties of these measures.

A key component of the whole assessment procedure is the accuracy performance analysis, which guarantees a thorough appraisal of the SQDS's ability to detect code smells and assess software security. The studies conducted yield valuable insights that aid in the improvement of machine learning models and promote ongoing progress in the fields of security research and software engineering. These performance measures provide useful benchmarks as the study progresses, pointing practitioners and academics in the direction of more dependable and efficient model deployment in practical situations.

TABLE 3. The accuracy performance metrics for five machine learning

Machine Learning	Accuracy
Logistic recognition	0.9365
Decision tree	0.9683
Random forest	0.9783
SVM	0.9683
KNN	0.9603

In this study, we aimed to use five machine learning algorithms: decision tree , random forest , logistic regression, KNN and SVM. The classification accuracy results of different algorithms are as follows: logistic regression (93.65%), decision tree (96.83%), random forest (96.83%), SVM (96.83%) and KNN (96.03%) these values represent the overall predictive ability of the models. High accuracy scores in all models indicate that the selected features, together with safety and quality considerations, provide sufficient information for effective classification Especially the decision tree, random forest, and SVM models consistently showed outstanding performance, and achieved an accuracy rate of 96.83% consistently This showed a strong ability to distinguish between classes in the data set.

Comparing the results, we find a small difference in accuracy between Decision Tree, Random Forest, and SVM, with KNN lagging slightly behind. The logistic regression, although slightly lower in accuracy, achieved a commendable 93.65%. This variation highlights the various strengths and weaknesses of each algorithm when applied to this particular data set.

After that we applied three deep machine learning (RNN , LSTM and GRU) on both the original Fonata's dataset of God Class bad smell then calculate the performance metrics (accuracy , precision , recall and f1-score). Table (4) show the performance analysis for three models when using God Class bad smell with quality metrics.

TABLE 4. The performance metrics for (RNN, LSTM and GRU) on Fonata's God Class dataset

The performance Metrics.	Precision	Recall	Accuracy	F1_score
RNN	0.90	0.87	0.85	0.88
LSTM	1	0.85	0.90	0.92
GRU	1	0.89	0.93	0.94

Then, we analyzed our suggested SQDS dataset which was intended to identify the God Class code smell using three deep machine learning models: Gated Recurrent Unit (GRU), Long Short-Term Memory (LSTM), and Recurrent Neural Network (RNN). To assess the efficacy of these models, performance indicators such as accuracy, precision, recall, and F1-score were calculated. Table 5 presents the findings of this performance investigation, which included God Class code scent detection along with quality metrics. The table gives a brief summary of each deep learning model's performance in identifying complex patterns linked to the God Class code smell while taking into account more general software quality considerations.

TABLE 5. The performance metrics for (RNN, LSTM and GRU) on proposed SQDS dataset

The performance Metrics.	Precision	Recall	Accuracy	F1_score
RNN	1	0.85	0.90	0.95
LSTM	1	0.96	0.97	0.98
GRU	1	0.93	0.95	0.96

6. DISSCUSSION THE RESULT

The thorough performance metrics study for three different recurrent neural network model types (LSTM, RNN, and GRU) focusing on their effectiveness in identifying God Class code odors, is shown in Tables (4,5). The assessed metrics offer a comprehensive view of the models' performance and include precision, recall, accuracy, and F1-score.

Results for the RNN approach show a considerable improvement in accuracy, from 0.90 to one, as seen in Table (4). This significant increase demonstrates RNN's ability to correctly detect high-quality code while reducing false positives. Additionally, the RNN approach constantly performs well, demonstrating strong recall and F1 score, demonstrating its dependability in identifying God Class code smells.

Examining the LSTM approach, the study reveals some noteworthy advantages. The LSTM model predicts God Class code smells with great precision, suggesting a higher chance of correctness. Furthermore, the LSTM approach maintains high levels of accuracy, recall, and F1 score while exhibiting reliable and consistent performance across a variety of criteria.

Similar to the LSTM model, the GRU method's effectiveness in God Class code scent recognition is demonstrated by the analysis of accuracy and recall measures. With its competitive accuracy and F1 score, the GRU technique shows itself to be a dependable solution for this dataset, underscoring its general effectiveness and applicability for God Class code scent detection.

The results of these performance measures provide light on the unique advantages of every recurrent neural network model and offer insightful information to practitioners and researchers looking for efficient methods for God Class code smell identification. Based on particular project needs and objectives, the subtleties shown in the accuracy, recall, and overall performance metrics offer a nuanced knowledge of the models' capabilities and can direct the selection of the most appropriate method.

When we compare the previous results with our findings using the proposed SQDS dataset and three deep machine learning models, we find some interesting trends that provide useful information about how well each technique works to detect God Class code smells.

The SQDS dataset demonstrates perfect accuracy in the RNN approach, indicating the ability of RNN to accurately anticipate God Class odors. Moreover, the balanced metrics—memory and F1 scores, for example—highlight the SQDS dataset's ability to support a comprehensive workflow and demonstrate the dependability of the RNN model in this situation.

When we switch to the LSTM approach, our SQDS dataset shows superior memory and F1 scores, confirming that the model can accurately and precisely identify a sizable portion of real God Class occurrences. The LSTM approach performs admirably on all suggested datasets, extracting data with almost perfect accuracy.

Finally, the SQDS dataset highlights robust performance characteristics in the GRU technique, establishing GRU as a trustworthy model for God Class scent classification because of its exceptional accuracy and memory. The GRU model's applicability for the SQDS dataset is highlighted by the balanced metric method, which further guarantees a suitable equilibrium between model accuracy and F1 score.

In software engineering, identifying code smells is a crucial first step in achieving the best possible code quality and maintainability. The "God Class" is particularly notable among these code smells because of its tendency to take on too many responsibilities, which might have an effect on the entire program. The complex insights gained from these studies help to drive the ongoing search for better code smell detection techniques, which in turn improves the overall quality and maintainability of software systems as the field of software engineering develops.

In this study, the complex problem of identifying God Class code smells was tackled using three well-known deep learning models: Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM), and Gated Recurrent Unit (GRU). Two different datasets were used in the investigation:

the original dataset, which was taken from Fonata's extensive collection and contained the essence of God Class scents, and the proposed dataset, SQDS, which combined security and quality metrics for a more thorough analysis. The analysis of these models in comparison revealed important information about their consistency, the impact of the dataset, and factors to be taken into account when choosing a model. Notably, LSTM performed admirably in terms of precision, recall, accuracy, and F1 score, displaying notable consistency across the two datasets.

Because of its constancy, LSTM is positioned as a strong competitor for God Class fragrance identification, demonstrating its adaptability and dependability across a variety of dataset circumstances.

The SQDS dataset made it clear how dataset features affected model performance. The performance of every model was improved by adding security and quality metrics to SQDS. This emphasizes how important dataset properties are in determining how deep learning models are trained and evaluated. The capacity of the SQDS dataset to enhance the models' overall performance highlights the significance of a nuanced dataset design that takes into account the many facets of software security and quality.

When choosing a perfect model, the study results emphasize that the decision should be made in accordance with the project's particular requirements and goals. Regarding accuracy-focused applications, RNN and LSTM are also excellent choices. Nonetheless, LSTM turns out to be the best option if a high degree of overall performance and a harmony between recall and precision are required. This sophisticated knowledge offers researchers and practitioners insightful direction, empowering them to customize their model choices to the particular needs of their initiatives.

As a result, this work highlights the importance of dataset design and selection in impacting model outputs in addition to exploring the performance of deep learning models in the context of God Class code smell detection. The results pave the way for improvements in software engineering techniques and the creation of more reliable and effective tools for code quality and maintainability. They also add to the continuing discussion on efficient methods for code smell detection.

7. CONCLUSION

The machine learning models utilized in this extensive investigation have exhibited strong precision in efficiently categorizing datasets according to conservation efficiency factors. Among these models, decision tree, random forest, and SVM models performed exceptionally well, demonstrating the effectiveness of machine learning in assessing and classifying systems that prioritize both safety and quality criteria. These results demonstrate the feasibility of machine learning technologies for thorough system evaluations and provide a detailed understanding of the interaction between quality and safety concerns.

Using both datasets, the deep machine learning models RNN, LSTM, and GRU performed well in detecting God Class code smells. But it was clear that LSTM performed better overall in God Class fragrance detection, routinely outperforming RNN and GRU. This comparison approach, when applied to the particular job of identifying complex code smells, provides insightful information about the strengths and capacities of various deep learning models.

The SQDS dataset was essential in improving the machine learning models' overall performance and highlighting the need of including safety and quality criteria in the training process. The integration of security and quality measures in the SQDS dataset served as a stimulant to improve the models' detection of God Class code smells. This emphasizes how important it is to take a comprehensive approach to software evaluation that takes into account all relevant factors in order to guarantee more precise and consistent model performance.

8. FUTURE WORKS

Adding more code scent categories to the analysis opens up an interesting new research direction. The integration of diverse foul smells with security metrics has the capacity to yield a more all-encompassing comprehension of the complexities associated with software assessment. Examining the similarities and differences across various kinds of code smells and how they affect system security and quality metrics might help improve and broaden the present model framework.

Furthermore, research efforts in the future could concentrate on creating hybrid models that fuse machine learning with other cutting-edge methods like anomaly detection algorithms or natural language processing. In the end, these integrative methods may improve software engineering techniques by producing more complex and precise outcomes in detecting and addressing code smells.

This work lays the groundwork for other research projects that will deepen our comprehension of software assessment and code smell detection. In order to ensure code quality, security, and maintainability, machine learning is expected to become more reliable and applicable as a result of the investigation of various foul odors and creative model combinations.

REFERENCES

[1] ISO/IEC. (2011). Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models. ISO/IEC 2011.

[2] Mohammed,N. M., Alshayeb, M., Mahmood, S., Mohammed,N. M., & Niazi, M. (2017). Exploring Software Security Approaches in Software Development Lifecycle : A Systematic Mapping Study. Computer Standards & Interfaces.50,107-115. https://doi.org/10.1016/j.csi.2016.10.001

[3] Shahriza, N., Karim, A., Albuolayan, A., Saba, T., & Rehman, A. (2016). The practice of secure software development in SDLC : an investigation through existing model and a case study. Security and Communication

Networks,9(18),5333–5345. https://doi.org/10.1002/sec.1700

[4] Kumar, S. R. T., Sumithra, A., & Alagarsamy, K. (2010). The Applicability of Existing Metrics for Software Security. International Journal of Computer Applications, 8(2), 29–33.

[5] Daley, J. (2017). Insecure Software is Eating the World: Promoting Cybersecurity in an Age of Ubiquitous Software-Embedded Systems. Stanford Technology Law Review, 19(3), 533–546.

[6] Siddiqui, Shams Tabres. (2017). Significance of Security Metrics in Secure Software Development. International Journal of Applied Information Systems (IJAIS), 12(6). https://doi.org/10.5120/ijais2017451710

[7] Firesmith, D. (2004). Specifying Reusable Security Requirements. Journal of Object Technology, 3(1), 61–75.

[8] Siddiqui, Shams Tabrez, Hamatta, H. S. A., & Bokhari, M. . (2013). Multilevel Security Spiral (MSS) Model : NOVEL Approach. International Journal of Computer Applications (0975, 65(20), 15–20.

[9] Howard, M., & Lipner, S. (2006). The Security Development Lifecycle. Redmond: Microsoft Press. https://doi.org/10.1007/s11623- 010-0021-7.

[10] M. Fowler *et al.*, "Refactoring Improving the Design of Existing Code Second Edition," 2019.

[11] A. Kaur and G. Dhiman, *A review on search-based tools and techniques to identify bad code smells in object-oriented systems*, vol. 741, no. September. Springer Singapore, 2019. doi: 10.1007/978-981-13-0761-4_86.

[12] M. A. Wani, F. A. Bhat, S. Afzal, and A. I. Khan, *Advances in Deep Learning*, vol. 57, no. January. 2019. doi: 10.1007/978-981-13-6794-6.

[13] E. Tempero *et al.*, "The Qualitas Corpus: A curated collection of Java code for empirical studies," *Proc. - Asia-Pacific Softw. Eng. Conf. APSEC*, pp. 336–345, 2010, doi: 10.1109/APSEC.2010.46.

[14] M. Fowler *et al.*, "Refactoring Improving the Design of Existing Code Second Edition," 2019.

[15] M. I. Azeem, F. Palomba, L. Shi, and Q. Wang, "Machine learning techniques for code smell detection: A systematic literature review and meta-analysis," *Inf. Softw. Technol.*, vol. 108, pp. 115–138, 2019, doi: 10.1016/j.infsof.2018.12.009.

[16] T. Sharma *et al.*, "A Survey on Machine Learning Techniques for Source Code Analysis," vol. 0, no. 0, 2021, [Online]. Available: http://arxiv.org/abs/2110.09610

[17] H. M. Yahya and D. B. Taha, "Software Code Refactoring : A Comprehensive Review," vol. 2023, pp. 71–80, 2023, doi: 10.33899/edusj.2023.137163.1298.

[18] G. Lacerda, F. Petrillo, M. Pimenta, and Y. G. Guéhéneuc, "Code smells and refactoring: A tertiary systematic review of challenges and observations," *J. Syst. Softw.*, vol. 167, no. April, 2020, doi: 10.1016/j.jss.2020.110610.

[19] F. Arcelli Fontana and M. Zanoni, "Code smell severity classification using machine learning techniques," *Knowledge-Based Syst.*, vol. 128, pp. 43–58, 2017, doi: 10.1016/j.knosys.2017.04.014.

[20] Shin, Y., & Williams, L. (2008). Is Complexity Really the Enemy of Software Security ? In Proceedings of the 4th ACM workshop on Quality quality of protection (pp. 47–50)

[21] Alshammari, B., Fidge, C., & Corney, D. (2010a) Assessing The Impact of Refactoring on Software Security-Critical Object-Oriented Designs. In 2010 Asia Pacific Software Engineering Conference Assessing. https://doi.org/10.1109/APSEC.2010.30

[22] Zimmermann, T., Nagappan, N., & Williams, L. (2010). Searching for a Needle in a Haystack : Predicting Security Vulnerabilities for Windows Vista. In 2010 Third International Conference on Software Testing, Verification and Validation. https://doi.org/10.1109/ICST.2010.32

[23] Abid, C., Kessentini, M., Alizadeh, V., Dhaouadi, M., & Kazman, R. (2020). How Does Refactoring Impact Security When Improving Quality ? A Security-Aware Refactoring Approach. IEEE Transactions on Software Engineering, 5589(c), 1–15. https://doi.org/10.1109/TSE.2020.3005995

[24] Almogahed, Abdullah & Omar, Mazni & Zakaria, Nur Haryani & Alawadhi, Abdulwadood. (2022). Software Security Measurements: A Survey. 1-6. 10.1109/ITSS-IoE56359.2022.9990968.

[25] RomeoLQuoiJr, "Deep Learning-Based Code Smell Detection Using CNN and RNN," Computer Science & Technology, 2020.

[26] S. Subedi, "INTELLIGENT CODE. SMELL. DETECTION SYSTEM USING DEEP LEARNING," 2021.

[27] T. Sharma, V. Efstathiou, P. Louridas, and D. Spinellis, "Code smell detection by deep direct-learning and transfer-learning," *J. Syst. Softw.*, vol. 176, 2021, doi: 10.1016/j.jss.2021.110936.

[28] M. Y. Mhawish and M. Gupta, "Predicting Code Smells and Analysis of Predictions: Using Machine Learning Techniques and Software Metrics," *J. Comput. Sci. Technol.*, vol. 35, no. 6, pp. 1428–1445, 2020, doi: 10.1007/s11390-020-0323-7.

[29] P. Manadhata and J. Wing, "An attack surface metric," IEEE Transactions on Software Engineering, vol. PP , no. 99, p. 1, 2010.

[30] U. Mansoor, M. Kessentini, B. R. Maxim, and K. Deb, "Multi-objective code-smells detection using good and bad design examples," *Softw. Qual. J.*, vol. 25, no. 2, pp. 529–552, 2017, doi: 10.1007/s11219-016-9309-7.

[31] M. A. Wani, F. A. Bhat, S. Afzal, and A. I. Khan, *Advances in Deep Learning*, vol. 57, no. January. 2019. doi: 10.1007/978-981-13-6794-6.

[32] and A. J. S. Aston Zhang, Zachary C. Lipton, Mu Li, "Dive into DeepLearning," p. 987, 2020.

[33] A. Aksoy, Y. E. Ertürk, S. Erdoğan, E. Eyduran, and M. M. Tariq, “Estimation of honey production in beekeeping enterprises from eastern part of Turkey through some data mining algorithms,” *Pak. J. Zool.*, vol. 50, no. 6, pp. 2199–2207, 2018, doi: 10.17582/journal.pjz/2018.50.6.2199.2207.

E-mail:author's email