International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys. 16, No.1 (Aug-24)

http://dx.doi.org/10.12785/ijcds/160172

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] **and Dujan B. Taha**[2]

[1]*Department of Software, University of Mosul, Mosul, Iraq*
[2] *Department of Computer Science, University of Mosul, Mosul, Iraq*
Received 25 Feb. 2024, Revised 11 May 2024, Accepted 13 May 2024, Published 10 Aug. 2024

Abstract: Code smells are a signal 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 existence 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 dataset (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

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 denotes to the degree of system or a product will protect its data to ensure that different goods or people might access appropriate data according to their categories and authorizations. The eight quality attributes that make up the ISO/IEC 25010 defined product quality model are functional suitability, performance efficiency, compatibility, usability, reliability, security, maintainability, and portability [1].

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 existence of a security illusion and God class bad smell at the same time in a program, using deep machine learning to detect each of illusion of security and God class bad smell. 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.

Nowadays, the majority of software systems must meet security requirements [2]. Nevertheless, not all security problems can be resolved by traditional software metrics, resulting in the creation of several software systems that are hazardous [3]. In the initial stages of developing software, 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. For maximum efficiency and effectiveness, security early on in the development process, it should be taken into account [4]. To safeguard their systems, the majority of businesses invest a significant amount of money in purchasing firewall and antivirus software [5].

The degree to which a system or product protects information and data so that users or other systems or products can access it appropriately based on the types and degrees of authorization is known as security. It is made up of the subsequent sub-characteristics:

- Confidentiality: is the degree to which a system or product makes sure that information is only reachable to those who are permitted to do so.
- Integrity: is the level of protection provided by a

system, service, or product against illegal entree to or alteration of data or software.

- Non-repudiation: is the extent to which events or acts can be conclusively demonstrated to have occurred and, as a result, cannot be subsequently denied.
- Accountability: is the degree to which a unit may be held personally responsible for its actions.
- Authenticity: is the degree to which it is feasible to confirm the identity of a topic or resource.

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 [6]. 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 to find and fix problems [7].

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 (RNN), 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 [8].

Our paper follows this structure: Section two discusses the related works, section three provides background on software security metrics, code smell detections, and deep machine learning. The methodology of the research is obtainable in section four. Sections five and six present the results and its discussions and finally conclusions section and then future works.

2. RELATED WORK

After 1999, when Fowler et al. 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 [10], [11], [12].These investigations have demonstrated several methods and tactics for identifying poor code smell in current software systems through the utilization of machine learning methods [13].

Subedi [14] proposed a procedure for gathering, processing, and analyzing code smells from various open-source projects in order to use the LSTM machine learning model to detect code smells intelligently. Sharma et al. [15] used CNN and RNN as their major hidden layers along with the auto encoder model. They perform training and assessment on C# examples and Java code.

In [16] the purpose of this study is to compile and integrate the research on deep learning (DL) methods for odor identification. Considering the speed at which DL approaches are developing, we think that a study and analysis of the corpus of existing research would aid in the creation of new methods as well as aid in the identification of research gaps in this field. Methods: Up to October 2021, 67 studies on DL-based unpleasant scent identification were published. These were found using a methodical way.

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

Refactoring is a technique for reorganizing software’s internal architecture without affecting its functionality [6]. Numerous investigations were carried out to gauge the design’s early weaknesses. In [17] for the object-oriented layout, the researchers developed a set of safety measures that may help designers identify and address safety issues early in the design process. These measures might be helpful tools for assessing the security of various layout iterations. Specifically, it was suggested that seven protection measures be used to determine the degree of concord and encapsulation.

AL Mogahed et al. Software systems with high compatibility, low coupling, and minimal complexity are shown to be more secure, and vice versa [18].

In [19] the researchers use a dendrogram to illustrate the hierarchical grouping based on the correlations between software metrics and code smells to determine how comparable code smells are. This offers a new approach to classifying code smells.

Vulnerability identification is aided by the possibility of identifying bug patterns that may result in security vulnerabilities due to the broad availability of open-source code for analysis. Researchers in cybersecurity and software engineering have been motivated by recent advances in deep learning to use these methods to identify and analyze dangerous code patterns and semantics that point to vulnerable code characteristics. The authors of this paper assess and examine twelve research that use DL and Ml techniques to find software vulnerabilities. The purpose of this review is to further knowledge on how neural network-based techniques may be used to learn and understand code semantics, which will aid in vulnerability finding [20].

Previous studies in this field typically employ one of two approaches: either applying ML or DL techniques for identifying undesirable patterns in software code, or manually measuring a few specific security metrics to evaluate code susceptibility to attacks. My research introduces a distinctive methodology by creating an automated tool that conducts comprehensive scans of software programs and autonomously calculates seven different security metrics. This innovation bridges a critical gap in current research by merging automated techniques with security assessments, thereby increasing the precision and efficiency of security metric evaluations in software systems.

A major step forward in tackling the ubiquitous problems of system vulnerability and data breaches in modern software development is the integration of security measures with quality metrics in databases. In this study, A unique database combining satisfactory and security measures is provided, and system studying and deep getting to know techniques are used to evaluate the data. Security measurements ought to be protected since they offer critical insights into viable vulnerabilities that may not be visible with handiest first-rate measures. The machine's usual security posture can be stepped forward through extra effectively identifying and mitigating complicated protection dangers via the software of superior mastering algorithms to this extended dataset.

In addition, the integration of those security measures with first-class metrics ensures the system's resilience to attacks and upholds nice standards, which might be vital for machine preservation and dependability. This twin-cognizance method guarantees that the machine's performance and maintainability healthy excessive necessities whilst also addressing the twin necessity of defending towards security threats. Therefore, a holistic technique to improving software program maintenance and safety within a single framework is supplied by using the implementation of integrated safety and pleasant metrics. Software engineering research in the past has mostly concentrated on using quality metrics to find and identify code smells or security metrics to find vulnerabilities in software systems. The majority of these studies were conducted in siloed paradigms, with quality metrics trying to increase readability and maintainability of code and security metrics concentrating on strengthening the system's resistance to possible intrusions and assaults. But it's becoming increasingly clear that bad code smells, like feature envy or God classes, can lead to security flaws as well as lower software quality because they obscure and complicate the codebase, which leaves it more vulnerable to malicious exploits. This insight has inspired my research's creative methodology. This combines security and quality measures. It becomes feasible to fully solve security flaws that are ingrained in badly organized code by doing this. This integrated method guarantees a more thorough evaluation, resulting in a high-quality software system that is safe from possible risks and enhances maintainability and security inside a single framework.

3. BACKGROUND

Software development has seen an increase in research in latest years to improve code satisfactory, detect code smells, and improve safety protocols. Numerous research works have proposed a huge variety of methods and strategies for figuring out terrible code smells and comparing the safety and pleasant metrics of software program structures. With an emphasis on the usage of deep machine mastering, this segment gives a concise synopsis of the pertinent statistics at the methods used inside the identity of safety metrics and code smells.

Early inside the software improvement existence cycle, researchers and practitioners have realized how vital it's far to take proactive steps to locate and fix code-related problems. To hold software structures strong and reliable, it is crucial to investigate diverse techniques for detecting code smells and to assess security metrics. Furthermore, the mixing of deep device gaining knowledge of into those detection algorithms has surfaced as a viable method, providing the possibility of extra particular and powerful code anomaly identification. The dynamic field of software engineering emphasizes the need of ongoing upgrades to safety and code satisfactory approaches. Through an exploration of the nexus of deep machine getting to know, protection metrics, and code odor detection, this overview seeks to illuminate the today's techniques that aid the development of software program improvement processes. Further investigation reveals that the secret to resolving the complicated topics related to code first-class and security inside the ever-changing subject of software program engineering is the use of current technologies, along with deep gadget learning.

A. *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 offer insights that allow the creation of beneficial prediction fashions, directing the improvement of software program merchandise, through quantitatively measuring vital aspects of software systems. Prioritizing security features early inside the software development manner is the satisfactory route of movement, mainly at some point of the layout section [3]. It is impossible to overestimate the importance of protective software towards ability attacks in the contemporary virtual era, where cyber dangers are pervasive. Metrics play a role in assessing and forecasting the overall safety posture and first-rate of software program structures as they get more complicated. Developers can also proactively discover vulnerabilities and create sturdy, resilient software program solutions through methodically recording and studying metrics.

When properly applied, software metrics serve as a preventative degree that allows developers to peer any vulnerabilities early on and deal with them. This technique improves the software program's protection whilst also enhancing the general effectiveness and dependability of the completed result. Metrics-driven security practices are being integrated in keeping with the enterprise's fashionable circulate toward a proactive safety posture, which recognizes that predicting and stopping vulnerabilities is just as vital as responding to them.

Furthermore, software metrics' predictive quality is going past security problems to provide a radical draw close of the program's functionality, maintainability, and scalability.

Development groups might also make properly-informed selections, use sources wisely, and expedite the improvement technique with the help of this complete attitude. In preserving with the idea of ”secure by layout,” safety features are carried out at the layout degree, ensuring that security concerns are included within the software program's basic structure. The courting among metrics and protection will become increasingly essential as the software development panorama adjustments to offer software that is safe, reliable, and of high best. This paradigm alternate emphasizes how critical it's far to take a proactive, metrics-driven approach to reduce viable risks and make stronger software systems against usually changing threats. To sum up, to successfully navigate the troubles presented through software assaults and guarantee the creation of reliable and secure software program answers, it's far important to incorporate software program metrics as predictive gear and to adopt safety features early on.

As stated by the National Institute for Using Technology and Standards to eliminate vulnerabilities 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. 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, their findings indicate whether alternative designs may strengthen or weaken the security of a given class [21]. Here we give a definition for each security metric and its equation to find it [17]:

CIDA: The ratio of the quantity of public characteristics for a classed instance to the quantity of characteristics in a class that are categorized.

$$CIDA = \frac{|NCIA|}{|CA|} \tag{1}$$

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.

$$CCDA = \frac{|NCCA|}{|CA|} \tag{2}$$

COA: It is calculated by dividing all privately shared methods in a class by all publicly shared methods

$$COA = \frac{|NCM|}{|CM|} \tag{3}$$

CMAI: The relation between the total number of mutators that could potentially interact with characteristics that are categorized and the number of mutators that actually could

$$CMAI = \frac{\sum_{J=1}^{Ca} \alpha\left(CA_j\right)}{|MM| \times |CA|} \tag{4}$$

CAAI: Measuring the number of accessors that can interact with the classed characteristics yields the maximum number of accessors that can have access to those attributes.

$$CAAI = \frac{\sum_{J=1}^{Ca} \beta\left(CA_j\right)}{|AM| \times |CA|} \tag{5}$$

CAIW: The ratio of all possible interactions with part attributes and all possible pathways to all attributes.

$$CAIW = \frac{\sum_{j=1}^{Ca} \gamma\left(CA_j\right)}{\sum_{i=1}^{a} \delta\left(A_i\right)} \tag{6}$$

CMW: Equivalent to dividing the number of methods in a class by the ratio of categorized methods.

$$CMW = \frac{|CM|}{|M|} \tag{7}$$

B. Code Smell Detection

A crucial component of software program development is code smell detection, which finds and fixes tough styles or structures in supply code. These ”smells” are signs and symptoms of viable inefficiencies or design defects that could decrease the overall exceptional of this system and make it harder to preserve. Developers may systematically find positive code smells, which include reproduction code, prolonged procedures, or inconsistent naming conventions, through utilizing a variety of static code analysis approaches. Enhancing code clarity, maintainability, and scalability is the fundamental objective of code scent detection, which facilitates to construct greater dependable and powerful software structures. As software tasks get more complex, it's far essential to discover and put off code smells early directly to ensure lengthy-term sustainability and facilitate development crew verbal exchange.

Code smell detection is typically created by predefined threshold values and grouping metrics of object-oriented, aimed at identifying the main indications that define the code smells [22]. A variety of detection approaches rely on heuristics and detection rules that compare metric values obtained from source code with empirically established thresholds, 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 smell that will be used in our research is God class we can define 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 category frequently includes excessive code, more than one strategy, and tightly coupled dependencies, main to excessive coupling and coffee concord.

A variety of superior equipment had been created to help builders pick out code smells, which might be signs of feasible problems with the software program's structure that would reason it to grow to be much less complex or of worse great. PMD, JDeodorant, and SonarQube are a few of this gear. SonarQube is a feature-wealthy device that provides insights on duplicate code and coding standards breaches similarly to detecting specific code smells. JDeodorant is specifically made to stumble on code smells. Checking and provide refactoring answers for them. Another famous tool that exams code for diverse horrific coding behavior, such as empty seize blocks, superfluous complex expressions, and unneeded variables, is known as PMD. By addressing and reworking problematic code structures, these tools help developers improve maintainability, lower error rates, and increase overall program performance all of which are crucial for sustaining high-quality codebases [23].

C. Deep Machine Learning

A department of artificial intelligence referred to as "deep gadget learning" uses sophisticated neural networks and algorithms to evaluate and study massive amounts of records. The period "deep" describes the use of numerous layers of neural networks to investigate input, mainly due to the introduction of increasingly complicated and correct models. With this period, system studying will undergo a revolution as computer systems can be able to find out patterns, categorize statistics, and generate predictions which might be greater correct than in advance. Deep system gaining knowledge of has numerous uses, which encompass photo identity, predictive analytics, and herbal language processing [24].

Deep studying's transformational electricity is tested by using its potential to routinely extract complicated traits from statistics, enabling more correct and nuanced decision-making. We assume a paradigm exchange in numerous regions as this era develops, along with self-sustaining structures, economic forecasts, and medical diagnostics. Because of its versatility and capacity to manipulate difficult records systems, the deep getting to know framework is a key component in fixing troubles inside the real international.

Furthermore, via incorporating deep studying into disciplines like pc vision, advances in object and image detection have been made, greatly augmenting the electricity of computerized structures. Another thing of deep learning is predictive analytics, which helps businesses make statistics-driven picks by predicting developments and seeing feasible possibilities and hazards.

Deep getting to know applications depend closely on herbal language processing, which has advanced to the point that robots can now understand, interpret, and bring language that is similar to that of humans. This will completely trade the way we engage with technology and feature a big effect on sentiment evaluation, language translation, and chatbots.

To sum up, the several uses of deep studying and its capacity to completely transform a number of sectors highlight how critical it is to the development of machine getting to know 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 is used to deal with sequential records by retaining contextual knowledge from advance inputs. However, RNNs have 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. Traditional feed ahead neural network system inputs one by one. To address this issue, some RNN modifications have been proposed, which include LSTM and GRU, which use greater strategies to manipulate the input waft via the community. Long Short-Term Memory, or LSTM, is an RNN architectural kind that is employed in deep studying. LSTM networks aims to deal with the standard RNN's vanishing gradient trouble [25]. 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. To determine whether or not statistics have to be eliminated from the reminiscence cellular, the forget about the gate is critical [26], [27].

A set of performance measurements that quantify these algorithms' efficacy across a range of tasks is essential for critically evaluating machine and deep learning algorithms [28].

Accuracy: The percentage of accurate outcomes (true positives and true negatives) among all instances investigated is known as accuracy.

$$Accracy = \frac{Tp + TN}{TP + FP + TN + FN} \tag{8}$$

Recall: The percentage of true positives that the model accurately detects.

$$Recall = \frac{Tp}{TP + FN} \tag{9}$$

Precision: The percentage of positively identified cases that were in fact accurate.

$$Precision = \frac{Tp}{TP + FP} \tag{10}$$

F1 Score: The harmonic means of recall and accuracy, which offers a single statistic that strikes a compromise between the two issues.

$$F1\ Score = 2 * \frac{Precision \times Recall}{Precision + Recall} \tag{11}$$

By measuring the proportion of true positives compared to all positive predictions, precision offers valuable information on the dependability of positive class predictions. Recall, which is another name for sensitivity, quantifies a model's capacity to locate each and every pertinent example in a dataset. When working with class-imbalanced data, the F1-score provides a balance between recall and accuracy.

In actuality, the particular needs of the application determine which measure is best. For example, a high recall is required if preventing false negatives is important. However, accuracy becomes more crucial when preventing false positives is of the utmost importance. By combining these indicators, analysts and data scientists may enhance and get a deeper thoughtful of their prediction models, enabling them to function at their best in practical applications. The multifaceted method of evaluating the model aids in optimizing its parameters for increased prediction accuracy and dependability.

4. METHOD

The proposed dataset SQDS, is being developed through a methodical procedure that includes many crucial components as shown in Fig 1. First stage contained data gathering from the Qualitas Corpus software repository consist of 74 open-source Java systems will be download. The second stage we programming a parser examen every system in this dataset, then preprocess the software to enable the third stage thorough analysis, we will utilize a specially designed parser that is designed to methodically examine the classes included in every software.

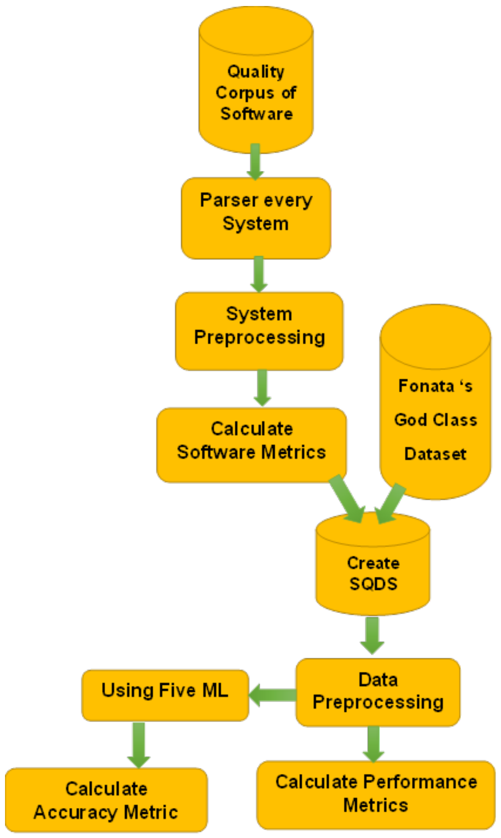

Quality Corpus of Software
Parser every System
System Preprocessing
Fonata 's God Class Dataset
Calculate Software Metrics
Create SQDS
Data Preprocessing
Using Five ML
Calculate Accuracy Metric
Calculate Performance Metrics

Figure 1. The process Stages for Method

The fourth stage contain system preprocessing focuses on measurable elements like the quantity of public and private properties and methods within each class and cleaning all comments and blank lines from the source code was being parsed. This required getting rid of everything extraneous that may have tainted the analysis. In particular, this is an important step since it guarantees that our analysis just looks at the code's functional elements, which improves the process's accuracy for detecting code smells.

The final step is to take the parsed code 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 quality metrics related to same open-source java program that will be found in Fonata God Class dataset, an extensive collection of 62 quality indicators, to further enhance the dataset. Combining the datasets which are shown in Table I that will be calculated by Fonata and three PhD students contains quality metrics [23].

TABLE I. Show quality metrics in Fonata's dataset [23]

Quality Attributes	Quality Metrics
Complexity	CYCLO, WMC, WMCNAMM, AMW, WOC, NOP NOAV, ATLD
size	LOC, NOM, NOPK, NOCS, NOMNAMM, AMW, NOA
Cohesion	TCC, LCOM
Coupling	ANOUT, ATFD, FDP, RFC, CINT, CC, CM
Encapsulation	LAA, NOAM, XOPA
Inheritance	DIT, NOI, NOC, NMO, NIM, NOII

In software development, a system's resilience and upkeep are greatly influenced by a number of factors. A summary of the main traits of high-quality software and their corresponding effects may be found below [1], [29].

A. *Complexity*

A measure of how intricately the program is structured, which can have an impact on mistakes' probability, readability, and maintainability.

Metrics including Lines of Code, Number of Methods, Number of Packages, Number of Classes, and Number of Attribute, measure the dimensions of the software. Size metrics are indicative of the code quantity and may be used to estimate undertaking attempt, predict protection fee, and gauge standard system scalability.

B. *Size*

Describes the size of the program, usually expressed as the number of functions and classes or lines of code [30].

Metrics including Lines of Code, Number of Methods, Number of Packages, Number of Classes, and Number of Attributes, measure the dimensions of the software. Size metrics are indicative of the code quantity and may be used to estimate undertaking attempt, predict protection fee, and gauge standard system scalability.

C. *Cohesion*

Is a measure of how well a module's components fit together; more cohesion is preferable for maintainability [31]. Tight Class Cohesion and Lack of Cohesion in Methods are concord metrics. Cohesion refers to the degree to which factors inner a module belongs together. High brotherly love inside a module usually indicates a better design which makes the software easier to preserve and understand.

D. *Coupling*

Indicates how dependent one module is on the others; low coupling is ideal to lessen the effect of changes to one module on the others [32].

Coupling is measured using the following metrics: Afferent Coupling, Coupling Complexity, Foreign Data Providers, Response For a Class, Coupling Intensity), and Coupling between Methods. A system with a lower coupling is better since it means that its parts are less interdependent, which improves modularity and lowers integration or change risks.

E. *Encapsulation*

Encapsulation: the procedure of limiting access to a class's internal operations in order to maintain integrity and avoid accidental interactions.

A system's usage of encapsulation is assessed by metrics like exchange of Private Access, Number of Accessor Methods , and Locality of Attribute Accesses. A well-designed encapsulation helps to defend against changes in external components and lessens system fragility by concealing the internal states and functions of one component from others.

F. *Inheritance*

a feature of object-oriented programming that encourages reuse by having a class inherit behaviors and attributes from a parent class.

Metrics that evaluate the application's usage of inheritance include Depth of Inheritance Tree, Number of Children, Number of Methods Overridden, Number of Inheritances Number of Inherited Methods, and Number of Immediate Subclasses. While polymorphism and code reuse can be enhanced by the appropriate use of inheritance, overuse of inheritance can result in complicated and difficult-to-maintain code.

Our method ambitions to generate a extra comprehensive and nuanced view of the software program structures underneath exam by using fusing security signs with an established great dataset.

This mixture allows for a extra thorough evaluation that takes into account factors of usual code pleasant in addition to safety. It is predicted that the ensuing SQDS dataset might be a useful device for each practitioners and pupils, presenting insights into the complicated interactions that exist between security and excellent measures in open-source Java structures. This venture is in line with the overarching objective of enhancing dataset richness and enabling more reliable analyses inside the fields of security research and software engineering.

With 420 rows and 68 columns that suggest numerous attributes, the SQDS dataset has an in-depth shape. The information then goes thru second level of preprocessing thorough training technique was used to keep the caliber of our system mastering fashions before the dataset became prepared for gadget getting to know analysis. The managing of missing records and the elimination of incomplete rows have been important steps in this procedure.

To preserve the accuracy, completeness, and representativeness of the records used to train our system mastering fashions, this method is crucial. To lessen biases and mistakes in the system gaining knowledge of manner and boom the validity of our conclusions, such cautious data curation is vital. A vital preprocessing step that is designed to convert unprocessed information into a layout that can be analyzed. To do that, the statistics need to be carefully wiped clean to get rid of any lacking values and any adjustments together with scaling or normalization. This kind of preprocessing is crucial because it improves the general efficacy and quality of the records analysis that follows, generating effects which might be more particular and straightforward. Preprocessing is essential to providing a strong basis for our study. Together, they strengthen our methodological foundation and guarantee that all ensuing analyses whether algorithmic or manual are based on the best possible and most consistent data.

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 unique Fonata dataset in addition to our recommended SQDS dataset to similarly deepen the scope of our studies. The use of deep learning strategies holds the capability to reveal complicated patterns and subtleties gift in the datasets, therefore advancing a comprehensive comprehension of code first-class and security in software program systems. The thorough evaluation of these models and procedures is important to pushing software engineering and protection research forward and offers insightful facts to both scholars and practitioners.

5. THE RESULT

This study uses of regular accuracy overall performance standards generated from the confusion matrix to assess the efficacy of the SQDS. An essential method for evaluating a version's performance in category is the confusion matrix.

Its paperwork the muse for several performance indicators through methodically classifying predictions into proper positives, actual negatives, false positives, and fake negatives. A key indicator referred to as accuracy evaluates how correct the model's predictions are universal. The ratio of efficaciously anticipated times to all occurrences in the test dataset is used to compute it. Recall evaluates the model's ability to trap each high-quality occasion, whereas precision examines the accuracy of high-quality predictions. An unbiased assessment is given by the F1 rating, that is the harmonic imply of consider and accuracy. The investigation also explores specificity and sensitivity, which center on as it should be figuring out negative and fine examples, respectively.

A thorough information of the accuracy performance of each of the 5 machine studying strategies (Logistic Regression, Decision Tree, Random Forest, SVM, and KNN) is provided by way of the performance analysis, which is displayed in Table II and Fig. 2., through near examination of these metrics, researchers may also gain valuable insights into the benefits and downsides of each model, helping inside the identity of the first-rate method for code smell detection and protection assessment.

TABLE II. 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

The examine additionally seems at the results of false positives and fake negatives due to the fact those occurrences are critical in practical applications. While fake negatives can offer critical protection troubles, fake positives may bring about useless moves or alarms.

Practitioners gain a deeper information of the fashions' practical software by using comprehending the subtleties of those measures.

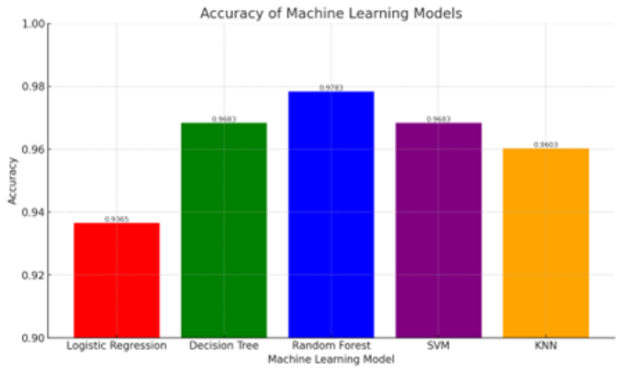


Figure 2. Accuracy of machine learning used

A key element of the entire evaluation procedure is the accuracy overall performance evaluation, which ensures a radical appraisal of the SQDS's capacity to hit upon code smells and verify software program security. The studies performed yield treasured insights that resource in the improvement of machine learning fashions and sell ongoing development inside the fields of protection research and software engineering. These performance measures offer useful benchmarks as the have a look at progresses, pointing practitioners and lecturers within the course of greater dependable and green model deployment in sensible conditions.

In this study, we aimed to use five ML algorithms: decision tree, logistic regression, random forest, 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 3 deep machine mastering (RNN, LSTM, and GRU) on both the original Fonata's dataset of God Class bad smell and then calculated the performance metrics (accuracy, precision, consider, and F1-rating). Table III indicates the performance analysis for 3 models while the use of God Class terrible odor with quality metrics.

TABLE III. 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 counseled SQDS dataset which become supposed to perceive the God Class code scent the usage of 3 deep device gaining knowledge of models: Gated Recurrent Unit (GRU), Long Short-Term Memory (LSTM), and Recurrent Neural Network (RNN). To determine the efficacy of those fashions, performance indicators consisting of accuracy, precision, do not forget, and F1-score had been calculated. Table IV gives the findings of this overall performance research, which included God Class code heady scent detection together with fine metrics. The desk offers a precis of each deep mastering version's performance in figuring out complex styles related to the God Class code odor whilst contemplating greater standard software nice considerations. In Fig. 3 we draw a chart contain the comparison between the deep machine learning with the performance metrics on our suggested SQDS dataset.

TABLE IV. 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

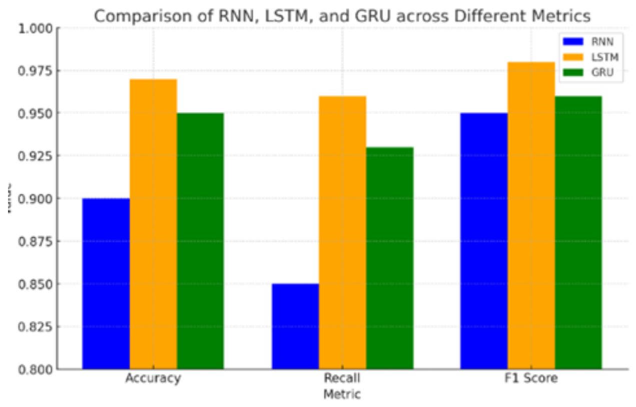


Figure 3. Comparison deep machine learning across different performance metrics

6. DISCUSSING THE RESULT

This work highlights the significance of dataset design and choice in impacting model outputs in addition to exploring the performance of deep getting to know models in the context of God Class code smell detection. The outcomes pave the way for upgrades in software engineering strategies and the creation of extra dependable and powerful tools for code best and maintainability. They additionally upload to the continuing dialogue on efficient methods for code smell detection.

Results for the RNN technique show a extensive improvement in accuracy, from 0.90 to one, as visible in Table III. This significant increase demonstrates RNN's capability to efficaciously stumble on first rate code even because it decreases fake positives. Additionally, the RNN method continually performs properly, demonstrating sturdy recollect and F1 rating, demonstrating its dependability in figuring out God Class code smells. The comprehensive performance metrics take a look at for 3 exceptional recurrent neural community version sorts (LSTM, RNN, and GRU) focuses on their effectiveness in identifying God Class code odors, and is shown in Table III and IV. The evaluated metrics offer a complete view of the fashions' usual performance. Analyzing the LSTM approach, the observer diagnosed many brilliant advantages. With incredible accuracy, the LSTM model predicts God Class code smells, indicating a higher chance of accuracy. Moreover, the LSTM technique continues high stages of don't forget, accuracy, and F1 rating while demonstrating regular and dependable performance across several parameters.

Like the LSTM version, accuracy and recall measurements are analyzed to show how powerful the GRU approach is in God Class code scent detection. The GRU approach shows that it's miles a reliable solution for this dataset based on its competitive accuracy and F1 score, highlighting its trendy efficacy and adaptableness for God Class code scent detection. The consequences of those performance metrics shed mild at the distinct advantages of each recurrent neural network version and permit researchers and practitioners attempting to find powerful strategies for God Class code odor popularity with valuable records. The nuances shown inside the accuracy, take into account, and normal performance signs offer a nuanced information of the models' talents and may guide the choice of the high-quality suitable approach, relying on precise task demands and targets. Using the proposed SQDS dataset and three deep machine studying models, we discover some fascinating dispositions that provide important insights into the relative effectiveness of each method for detecting God Class code smells when we examine our findings with the previous studies.

The SQDS dataset shows that the RNN technique is completely accurate, suggesting that RNN can reliably assume God Class scents. Furthermore, the F1 scores and balanced metrics reminiscence, for instance, display how reliable the RNN version is in this situation and showcase how properly the SQDS dataset can Finally, the SQDS dataset highlights strong overall performance characteristics in the GRU technique, establishing GRU as a straightforward model for God Class heady scent category because of its extremely good accuracy and memory. The GRU model's applicability for the SQDS dataset is highlighted by using the balanced metric technique, which in addition guarantees a appropriate equilibrium between version accuracy and F1 score. In software engineering, figuring out code smells is a crucial first step in achieving the great viable code pleasant and maintainability. The "God Class" is particularly top notch amongst those code smells because it has a tendency to tackle too many obligations, which would possibly have an effect on the complete program. The complex insights received from these studies help to pressure the continuing search for better code smell detection strategies, which in turn improves the general satisfactory and maintainability of software program systems as the sector of software program engineering develops.

In this have a look at, the complex problem of identifying God Class code smells became tackled using 3 famous deep gaining knowledge of models: Recurrent Neural Network, Long Short-Term Memory, and Gated Recurrent Unit. Two specific datasets were used in the research: the unique dataset, which was taken from Fonata's huge series and contained the essence of God Class scents, and the proposed dataset, SQDS, which combined protection and satisfactory metrics for a more thorough evaluation.

The evaluation of these fashions in contrast discovered vital records approximately their consistency, the impact of the dataset, and elements to be taken into account when selecting a version. Notably, LSTM finished admirably in terms of precision, take into account, accuracy, and F1 rating, displaying extraordinary consistency at some stage in the 2 datasets.

Because of its constancy, LSTM is placed as a robust competitor for God Class fragrance identification, demonstrating its adaptability and dependability across hundreds of dataset instances. The SQDS dataset made it clear how dataset competencies affected model performance. The overall performance of each model changed into progressed via including protection and great metrics to SQDS. This emphasizes how crucial dataset houses are in identifying how deep studying models are educated and evaluated. The capability of the SQDS dataset to decorate the fashions' fundamental general overall performance highlights the importance of a nuanced dataset layout that takes into consideration the diverse aspects of software program protection and fine.

When deciding on a really perfect version, the look at results emphasize that the choice must be made on the mission's unique necessities and desires. Regarding accuracy-focused applications, RNN and LSTM are also outstanding alternatives. Nonetheless, LSTM seems to be the nice alternative if a excessive degree of regular average overall performance and harmony among keep in mind and precision are required. This brand-new know-how offers researchers and practitioners insightful course, empowering them to personalize their version picks to the ideal wishes in their tasks.

In our paintings, version interpretability is vital as it offers transparency to our gadget learning techniques, which in flip builds self-perception and comprehensibility.

We have placed strategies into vicinity to make our models' selection-making system more obvious. As a part of this, feature importance is analyzed to decide which code factors have the most consequences on code smell prediction. Furthermore, techniques like layer weight evaluation shed mild on how neural networks behave in our deep getting to know models. In software program engineering, interpretability is critical for both sensible software and educational rigor. In this field, understanding the ”why” at the back of a model’s forecast is simply as critical because the prediction itself. Our models are not simplest opaque systems; as an alternative, they are devices that offer treasured and explainable outcomes.

7. CONCLUSION

1) The SQDS dataset’s inclusion of security and quality metrics acted as a catalyst to enhance the models’ ability to identify God Class code smells. This highlights how crucial it is to evaluate software holistically, accounting for all pertinent aspects, to ensure more accurate and consistent model performance.
2) The machine learning models utilized in this extensive investigation have proven to be very accurate in their ability to categorize datasets according to conservation efficiency metrics. SVM, random forests, and decision trees fared better than the others, demonstrating the value of machine learning as a tool for classifying and assessing systems that prioritize quality and safety requirements. These findings provide a thorough understanding of the interaction between quality and safety concerns and demonstrate the viability of using machine learning techniques for thorough system assessments.
3) The deep machine learning models RNN, LSTM, and GRU did a good job at identifying God Class code smells using both datasets. Nonetheless, it was evident that LSTM consistently outperformed RNN and GRU in God Class scent detection, showing superior overall performance.
4) When this comparison method is used for the specific task of locating complicated code smells, it offers valuable insights into the capabilities and advantages of different deep-learning models. The SQDS dataset was crucial in enhancing the overall performance of the machine learning models and emphasizing the necessity of including safety and quality requirements during the training phase.

8. Future Works

1) Create other datasets related to different bad smells such as Feature Envy, long methods and Data Class, with Quality metrics.
2) Using other machine learning algorithms on the SQDS, or hybrid between ML and DL to enhance the performance.
3) Combine these Datasets to detect more than bad smells at the same time in the code of software.
4) Using techniques of refactoring after calculating the security metrics to enhance the security and quality of software. So, this will help the developers in software engineering.

This work lays the groundwork for further research projects to deepen our understanding of software evaluation and code scent detection. To ensure that the code is beautiful, preservable, and maintainable, the gadget mastering is expected to be more reliable and appropriate due to a combination of research on different bad smells and creative models.

References

[1] S. Wagner, *Software product quality control*. Berlin, Heidelberg: Springer, 2013.

[2] S. Lipner, “Security development lifecycle: Security considerations for client and cloud applications,” *Datenschutz und Datensicherheit-DuD*, vol. 34, no. 3, pp. 135–137, 2010.

[3] J. Daley, “Insecure software is eating the world: Promoting cybersecurity in an age of ubiquitous software-embedded systems,” *Stanford Technology Law Review*, vol. 19, no. 3, 2017.

[4] W. Jansen, *Directions in security metrics research*. Darby, USA: Diane Publishing, 2010.

[5] N. M. Mohammed, M. Niazi, M. Alshayeb, and S. Mahmood, “Exploring software security approaches in software development lifecycle: A systematic mapping study,” *Computer Standards & Interfaces*, vol. 50, pp. 107–115, 2017.

[6] M. Fowler, *Refactoring: improving the design of existing code*. Boston, USA: Addison-Wesley Professional, 2018.

[7] A. Kaur and G. Dhiman, “A review on search-based tools and techniques to identify bad code smells in object-oriented systems,” in *Harmony Search and Nature Inspired Optimization Algorithms: Theory and Applications, ICHSA 2018*. Springer, 2019, pp. 909–921.

[8] M. A. Wani, F. A. Bhat, S. Afzal, and A. I. Khan, *Advances in deep learning*. Singapore: Springer, 2020.

[9] E. Tempero, C. Anslow, J. Dietrich, T. Han, J. Li, M. Lumpe, H. Melton, and J. Noble, “The qualitas corpus: A curated collection of java code for empirical studies,” in *2010 Asia pacific software engineering conference*. Sydney, NSW, Australia: IEEE, 2010, pp. 336–345.

[10] M. I. Azeem, F. Palomba, L. Shi, and Q. Wang, “Machine learning techniques for code smell detection: A systematic literature review and meta-analysis,” *Information and Software Technology*, vol. 108, pp. 115–138, 2019.

[11] T. Sharma, M. Kechagia, S. Georgiou, R. Tiwari, I. Vats, H. Moazen, and F. Sarro, “A survey on machine learning techniques for source code analysis,” *arXiv preprint arXiv:2110.09610*, 2021.

[12] G. Lacerda, F. Petrillo, M. Pimenta, and Y. G. Guéhéneuc, “Code smells and refactoring: A tertiary systematic review of challenges and observations,” *Journal of Systems and Software*, vol. 167, p. 110610, 2020.

[13] F. A. Fontana and M. Zanoni, “Code smell severity classification using machine learning techniques,” *Knowledge-Based Systems*, vol. 128, pp. 43–58, 2017.

[14] S. Subedi, “Intelligent code smell detection system using deep learning,” Master’s thesis, Institute of Engineering, Pulchowk Campus, Nepal, 2021.

[15] T. Sharma, V. Efstathiou, P. Louridas, and D. Spinellis, "On the feasibility of transfer-learning code smells using deep learning," *Journal of Systems and Software*, vol. 176, p. 11093, 2019.

[16] A. Alazba, H. Aljamaan, and M. Alshayeb, "Deep learning approaches for bad smell detection: a systematic literature review," *Empirical Software Engineering*, vol. 28, no. 3, p. 77, 2023.

[17] B. Alshammari, C. Fidge, and D. Corney, "Assessing the impact of refactoring on security-critical object-oriented designs," in *2010 Asia Pacific Software Engineering Conference*. Sydney, NSW, Australia: IEEE, 2010, pp. 186–195.

[18] A. Almogahed, M. Omar, N. H. Zakaria, and A. Alawadhi, "Software security measurements: A survey," in *2022 International Conference on Intelligent Technology, System and Service for Internet of Everything (ITSS-IoE)*. Hadhramaut, Yemen: IEEE, 2022, pp. 1–6.

[19] L. Bamizadeh, B. Kumar, A. Kumar, and S. Shirwaikar, "An analytical study of code smells," *Tehnički glasnik*, vol. 15, no. 1, pp. 121–126, 2021.

[20] A. Bahaa, A. E.-R. Kamal, and A. S. Ghoneim, "A systematic literature review on software vulnerability detection using machine learning approaches," *FCI-H Informatics Bulletin*, vol. 4, no. 1, pp. 1–9, 2022.

[21] P. K. Manadhata and J. M. Wing, "An attack surface metric," *IEEE Transactions on Software Engineering*, vol. 37, no. 3, pp. 371–386, 2010.

[22] U. Mansoor, M. Kessentini, B. R. Maxim, and K. Deb, "Multi-objective code-smells detection using good and bad design examples," *Software Quality Journal*, vol. 25, pp. 529–552, 2017.

[23] F. Arcelli Fontana, M. V. Mäntylä, M. Zanoni, and A. Marino, "Comparing and experimenting machine learning techniques for code smell detection," *Empirical Software Engineering*, vol. 21, pp. 1143–1191, 2016.

[24] M. A. Wani, F. A. Bhat, S. Afzal, and A. I. Khan, *Advances in deep learning*. Singapore: Springer, 2020.

[25] F. Mortezapour Shiri, T. Perumal, N. Mustapha, and R. Mohamed, "A comprehensive overview and comparative analysis on deep learning models: Cnn, rnn, lstm, gru," *arXiv e-prints*, p. 2305, 2023.

[26] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, "Dive into deep learning," *arXiv preprint arXiv:2106.11342*, 2021.

[27] A. Aksoy, Y. E. Ertürk, S. Erdogan, E. Eyduran, and M. M. Tariq, "Estimation of honey production in beekeeping enterprises from eastern part of turkey through some data mining algorithms," *Pakistan Journal of Zoology*, vol. 50, no. 6, 2018.

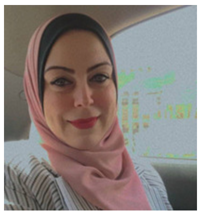
[28] V. M. Patro and M. R. Patra, "Augmenting weighted average with confusion matrix to enhance classification accuracy," *Transactions on Machine Learning and Artificial Intelligence*, vol. 2, no. 4, pp. 77–91, 2014.

[29] A. Almogahed, H. Mahdin, M. Omar, N. H. Zakaria, Y. H. Gu, M. A. Al-Masni, and Y. Saif, "A refactoring categorization model for software quality improvement," *Plos one*, vol. 18, no. 11, p. e0293742, 2023.

[30] M. Lorenz and J. Kidd, *Object-oriented software metrics: a practical guide*. New Jersey, USA: Prentice-Hall, Inc., 1994.

[31] W. J. Brown, R. C. Malveau, H. W. McCormick III, and T. J. Mowbray, "Refactoring software, architectures, and projects in crisis," *Google Scholar Google Scholar Digital Library Digital Library*, 1998.

[32] R. Marinescu, "Measurement and quality in object-oriented design," Ph.D. dissertation, University of Timisoara, Timisoara, Romania, 2002.



Hiba Muneer Yahya Assistant Lecture in software Department and PhD Student in Computer Sciences. Computer science and Mathematics Collage, University of Mosul, Mosul, Iraq.

Dujan Basheer Taha Prof in Department of Computer Science, Computer Sciences and Mathematics Collage, University of Mosul, Mosul, Iraq.