International Journal of Scientific Research in Computer Science and Engineering
Vol.11, Issue.1, pp.23-28, February 2023
E-ISSN: 2320-7639
Available online at: www.isroset.org

Research Paper

Correlating Propensity between Code Smells and Vulnerabilities in Java Applications

Kritika[1]

[1]Government of India, New Delhi, India

Author's Mail ID: kritikaa2297@yahoo.com

Received: 03/Jan/2022; **Accepted**: 01/Feb/2023; **Published**: 28/Feb/2023

Abstract: The ever-advancing world in terms of technology from web 1.0 to web 3.0, the need for designing and developing software applications has increased many folds. The digitalization of everything at a quick pace from including banking applications, mobile gaming etc. has led to the negligence of the part of the software developers which has led to increment in maintainability as well as security issue of the application, namely, code smells and vulnerability respectively. Code smells are the niggardly practices followed while developing a software by the developers or the software engineers, thwacking the rudimentary delineation principles and cynically thwacking delineation idiosyncrasy. Vulnerability is the snag, glitch or blemishes existing in software or operating system allowing the attackers to derelict the security measures. The paper focusses on finding the relationship between the code smells and vulnerability detected using an Eclipse plugin, PMD and correlating them using software metrics and rule-based machine learning approach.

Keywords: Code smell, Vulnerability, software metrics, machine learning, K means clustering, data mining

1. Introduction

With the advancement of technology and internet communication networks, a humongous number of software applications have come into existence, developed within short span of time with least linchpin on its maintenance issues, leading the pathway to the attackers with derelicting measures of security and technical debt. One discernible feature of the non-theoretical liabilities is not up to par smells or often recognized as code smells by Fowler. The recent studies have shown cynical thwacking of code smells on maintainability of software. A code smell[1] is a technical debt in code that indicates infringement of rudimentary design postulates and negative thwack on design idiosyncrasy.

A software vulnerability is a shortcoming in the system or software application that allows attacker to derelict the information security infrastructures and measures[2]. The vulnerability in a software is dependent on three factors, namely, extant, paroxysm and escapade. The extant is the extent to which a vulnerability is contemporary in the arrangement. The paroxysm is the probability of setting foot on by the hackers. The escapade is the competence of the hacker to dominate the extant. The occurrence of software vulnerability correlates to the indecorous and sloppy developers who lack security knowledge[4].

The paper focusses on unearthing relationship between code smells and vulnerability based on the metric values and rules generated by a tool known as Weka. The results of three code smells and vulnerabilities were toned which showed the maximum accuracy value and similar rule set.
The code smells focused are God Class, Cyclomatic Complexity and Long Method and vulnerabilities focused are Too Many Method, NPath Complexity and Excessive Method Length detected using an Eclipse plugin, namely, PMD[9].

The source code consisting of code smells and vulnerabilities were curated from github based on the java programming language. A total of 50 source codes were evaluated and the source codes with maximum probability of a particular smell and vulnerability were chosen for experimental purpose.
The stimulus of the research is to perceive the fallibility of the set in motion of the software at the maintenance juncture of the software engineering cycle.

1.1 Code Smells

God Class is the genre of code smell which cater to streamline the reconnaissance of the system. It gravitates to be compounded, have much code and designates humongous amount of data from other categories of information. God class is burdensome due to its acreage and compounded-ness.
Cyclomatic Complexity is the computation of number of unrestricted path way within a class. Higher the number of

unrestricted path way, higher is the complexity and colossal is the effort which a developer has to put leading to wide range of defaults.

Long Method is the computation of code with colossal line of codes. It caters to be compounded and arduous for the developer as well as the client to interpret as it increases the degree of cohesion and coupling.

1.2 Vulnerabilities

Too Many Method refers to the conceptualization of encountering large number of methods within a method or class with varying parameters and varying or similar nomenclature. OOPs concept called polymorphism is assisted by this method. Despite being a vulnerability in software, it offers an advantage of less execution time.

NPath Complexity is the number of pathways possible for implementation for a given code. It also refers to the achiral implementation pathways through the method.

Excessive Method Length is a concept related to the unbridled lengthy method contrasting the scope of the method, catering to the loss of focus and complicating the testing as well as maintainability part while dealing with security issues.

1.3 Software Metrics

The object-oriented software metrics[3][4] is the measure of the proclivity which can be unfaltering and ascertainable. The metrics can be broadly broken down into two pigeonholes i.e. product and process metrics. The process metrics are defined as measure of varied features of the software development undertaking. The product metrics is defined as measure of varied features of the software product.

The tool used for computing software metrics is Scitool Understand[8] which provides with a large number of metrics values subdivided into three broad categories viz. complexity metrics, object-oriented metrics and volume metrics.

The tool was chosen for computing metrics as it comes with multiple features like scrutinizing source code with thousands of lines of code, supports many programming languages like java, c, c++, python etc., and provides reports, graphing, standard testing.

The tool used for finding code smells and vulnerabilities in different java programming software developments is a plugin, PMD [9] compatible with Eclipse. The tool comes with built in rule set as well as provides an environment for tailor made rule set. The most vital issues reported by this tool is the inefficacious code or poor programming habits which is one of the important features while detecting code smells and vulnerabilities.

Below is the list of the some of the metrics values used for comparative study as well as occurred with maximum accuracy.

Table 1: List of Metrics

Metrics Name	Definition
CountDeclMethod	Number of class methods
Sum Cyclomatic	Sum of cyclomatic complexities of all impacted functions and methods.
Count Line	Aggregate of all lines
Sum Cyclomatic Strict	Sum of strict headlong complexities of all encapsulated functions or methods
Count Paths	Aggregate of possible paths

The possible features of PMD includes glitch reporting, lamented code identification, labyrinthine expressions, piddling code and mimeograph code.

The machine learning is an aptness of artificial intelligence which is more prone to be heard and used in present day scenarios for different software development, data analysis, robot analytics etc. It has the competence of self-learning and enhance their learning experience without being programmed bluntly.

The machine learning methods can further be divided into four parts i.e. supervised learning, unsupervised learning, semi-supervised learning and reinforcement machine learning methods.

The research is based on the process of supervised learning, the which can be applied on the new datasets learnt from the past modelling of data. Sowing the seeds from the known training dataset the learning algorithm produces a conjecture corollary to make prognostication about the end results.

For the purpose of machine learning, open source software known as WEKA, Waikato Environment for Knowledge Analysis[5] [6]was taken up which produced upshot accuracy with 90% and above. The algorithms which produced the upshot with highest correctly classified instances are JRip and J48.

J48 is a decision tree algorithm[10] with foretelling machine learning analysis. The implicit fork of decision tree designates the varied impute with the portending as dependent variable. The algorithm constructs decision trees based on the set of training data.

JRip contrivances a proportional[10] rule learner. It comprises of two phases, namely, fabrication stage and optimization stage. The fabrication stage is further subdivided into grow phase and prune phase. In grow phase, one rule is grown by prepending precursory till the error rate is greater than or equal to 50%. Pruning phase is imperceptibly trimming each rule and allow the any final pruning of the precursory. In the optimization phase, the smallest possible descrition length is chosen as the culminating representative of the initial training set.

The remaining research article is divided as follows: segment II reflected the related work under literature review. Segment III explains the experimental model adopted for the research purpose. Segment IV highlights the conclusions and discussion related to the work. Segment V throws light on the future prospects of the work in the related field.

2. Literature Review

New fangled businesses, organisations, companies, and censorious infrastructure are advocated by software system implementing underwritten operations[12] and transactions, procuring services and merchandising humongous quantities of secretive data for reinforcing worthwhile decisions and persistent business systems.

A decade long search has been going on in the field of code smells via different researchers across the globe, to begin with Fowler[1] gave the original and well accepted definition of code smells. Among all the research conducted and published, it has been found out that the duplicate code[13] is the most widely studied code smell.

Rasool and Arshad[14] provided a review of the different kinds of tools and techniques available for detecting code smells with forecasting the limitations of various tools and techniques.

The rationality for choosing the source code applications written majorly in java as most of the applications are still written in java and preferred over others[15] and most abundantly found code smells are feature envy, long method, god class, long parameter list and duplicate code.

Santos et al. [16] examined the effect of code smells on software development procedure and reached a rationality that lack of understanding of specifications, meeting deadline and lack of testing the maintainability results in the occurrence of the code smells in the software being developed.

Vulnerability dataset proposed to detect bugs, faults, code smells are curated with the help of github, written in java programming language. A set of 25 such source codes were collected and analysed for the identification of lack of maintainability and sustainability issues in the software applications. An automated formal vulnerability was adopted using a tool called PMD which can boost from static to dynamic application development approaches.

The object oriented metrics defined by KK Agarwal [18] and globally accepted, based on which and the matching definitions and meaning of the metrics were found in the tool named Scitool Understand was taken for the research purpose.
The tool provides with a wide range of metrics values while supporting variety of programming languages like python, java, C++, Ruby etc.

Few of the metrics as found most prevalent while predicting the comparison results are listed below.

Sum Cyclomatic Complexity metric is the evaluation of intricacy of function's decision edifices. It reckons the number of untrammelled paths through the module.

Count Paths is the intricacy of number of individualistic paths in a body of code being written for developing a particular application. It may happen that despite having smaller body of code, the count path metric value may seem to be larger.

Sum Cyclomatic Strict is a sub category of sum cyclomatic complexity metric which calculates the metrics value based on logical ANDs and ORs in conditional expression with adding 1 to their complexity values. The values of complexity also helps in determining the risk associated with that module of application which ranges from 1 to 50 and above[8].

Count Line is the metric which helps in totaling the number of lines of code which are required to build the software application. It can widely range from 10 to 1000+ lines of code making it more complex and intricate.

Count Decl Method measures the number of local methods defined inside a block of code.

Dewangan S. [19] provided a juxtaposition of the five different code smells and put in a technique called Smote Class Balancing Technique and provided an elaborated result based on the chi square feature selection model.

Sehgal R. [20] in their research laid emphasis on providing the solution to the problem of code smell detected while maintaining a software and found out that despite refactoring a code smell in order to do away with, it may lead to generation of another code smell in the source code of the software invigorated.

Madeyski L. [21] used the MCC method as the main parameter as the performance metric and on the basis of the modest value formulated that code smells, namely, random forest and flexible discriminant analysis were the best performers when contrasted on the industrial level of juxtaposition.

Above all the mentioned research papers, there is a still scope for correlating code smells with vulnerabilities based on the different sets of tools and techniques. One such technique is elaborated in this research paper.

3. Experimental Setup

The research work is divided into seven different phases. The procedure begins by finding the code smells and vulnerabilities unveiled in the java applications curated from github. The same applications are bypassed into software metrics called scitool understand which computes the different metrics based on distinct parameters and then a dataset is computed taking into account the smells and vulnerabilities detected as advisors and corresponding metrics obtained for the similar instance synonymous in both tools.

The dataset so formulated undergoes into a data preprocessing stage and then k means clustering technique is applied using a tool called weka which bifurcates the dataset into training and testing sets and perform the analysis based on it.

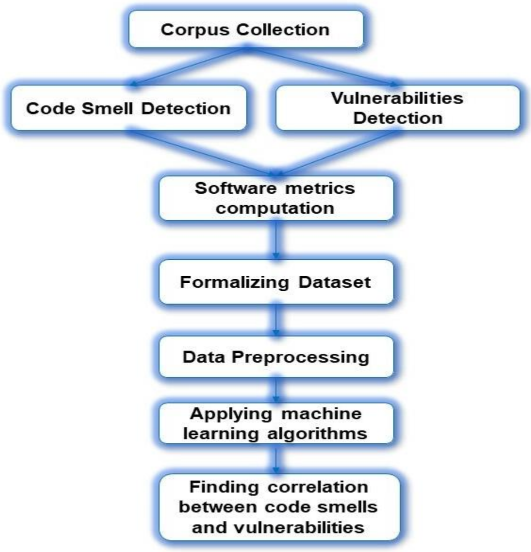

Corpus Collection
Code Smell Detection
Vulnerabilities Detection
Software metrics computation
Formalizing Dataset
Data Preprocessing
Applying machine learning algorithms
Finding correlation between code smells and vulnerabilities

Figure 1: Following are the research steps taken

A. Corpus Collection
The corpus collection consists of 25 source codes curated from github written in java programming language over a time span of 5 years. Applications built on java programming language are broadly chosen as more than 3 billion devices run on the applications developed in java language.

B. Code Smell Detection
The code smell detection is the primary step towards establishing the correlation. The tool used for it is PMD, an Eclipse Plugin with built in functions to check for complicated rules during motif phase of methods and classes. A total of three code smells, namely, god class, long method and cyclomatic complexity were detected in common from the curation of different software applications.

C. Vulnerability Detection
Along with the code smell also includes vulnerability detection with the help of Eclipse plugin, PMD from the set of curated software applications used in real world. Among the huge list of vulnerabilities that can be detected using the tool, 5 common and most widely found out vulnerabilities were collected for dataset formation and three, namely, excessive method length, npath complexity and too many method were found to be toning with rule set of code smells.

D. Metrics Computation
The object oriented software metrics[3] were computed using a tool called Scitool Understand which provides a tailor made environment with built in set of software metrics compatible with the java programming language and accredits static code analysis. The software metrics are computed in the form of csv file.

E. Dataset Formalization and preprocessing
One of the most pivotal steps in the whole process is the generation of dataset. The dataset has been manoeuvred using advisors(PMD) and software metrics obtained using Scitool Understand. The instances on comparing both the values are taken and formulated into a dataset with set of false values and true values for a particular code smell and vulnerability. The data undergoes a stage of preprocessing before being converted into arff files.

F. Machine Learning Application
The csv file on conversion to arff file was analysed using the classifier option in Weka using 10 fold cross validation technique[11] on different machine learning algorithms. The algorithms chosen were JRip, J48, Random Forest and naïve byes. The algorithm with the highest accuracy value as well as similar rule set was selected for the purpose of comparative study.

G. Finding Correlation between code smell and vulnerability
The final stage of the research experiment consists of uncovering the relationship between code smell and vulnerability on the basis of metrics value premeditated using Scitool Understand and k means clustering results produced by Weka.

4. Conclusion and Discussion

Through the research conducted on varied code smells and vulnerabilities, there exists a relationship between the two entities. The violation pattern corresponds with each other not only in definition but in practicality as well. The algorithms found most compatible with highest accuracy value is J48 and Jrip and the rule has been toned with the corresponding metric value generated using Scitool Understand.

On comparing the rule set generated using weka primarily based on the metrics values obtained from scitool understand, it was found that the algorithms Jrip and J48 produced the accuracy results with 90% and above.
The results were calculated based on the k-fold cross validation which is a resampling procedure applied on a limited dataset.

The value of k has been set to 10 meaning the dataset will be split into 10 parts and the upshot will be produced in the 11th part. The k fold cross validation process was chosen over the others as it aims at producing best result based on the measures of accuracy, f-measure and roc area curve where accuracy is the aptly classified instances in the positive and negative class, f-measure is the median of tp rate and recall and roc area curve is curve plotted against the values of tp rate and fp rate.

The stratified cross validation report consists of the following detailed information, namely, correctly classified instances, incorrectly classified instances, kappa statistics, mean absolute error, root mean square error, relative absolute error,

coverage of cases, total number of instances and root relative squared error.
The accuracy results are based on certain factors which can be termed as TP Rate, FP rate, Precision, Recall, F-Measure, ROC Area and class.
TP Rate is defined as the percentage of veritable positives that are plainly bracketed.

FP Rate is defined as the computation of the positives that are falsely identified by the classifier to the sum total of actual true positives and false positives.
Precision is defined as the proportion of the data computed was authentically faultless.

Recall is the ratio of the actual true positives to the sum total of true positives and false negatives as calculated by a classified model of weka by using different algorithms.
F-measure is the harmonic mean of precision and recall as defined above.
Roc curve presents the recital codification of all the portal values of the classification as computed based on the different algorithms of the machine learning tool being used.

The result has been computed based on the results obtained by k means clustering by bifurcating the dataset into training and testing sets and then formulating the values based on the different parameters as provided by the tool mentioned above. The following tables shows the similarity between code smell and corresponding vulnerability. The results obtained on comparing the two identities with the similar rule matched along with the algorithm that produces maximum accuracy on being run by the classifier section of Weka.

The rules are matched on the analysis obtained after k means clustering technique of the machine learning and rules so obtained are the similar metrics value for both code smell and vulnerability.

Table 2: Correlation between vulnerability and code smell

Code smell	Vulnerability	Algorithm	Rule matched
God class	Too many method	JRip	CountDeclMethod>=17
Long method	Excessive method length	JRip	Count Line>=80 and SumCyclomatic >=11
Cyclomatic complexity	NPath complexity	J48	SumCyclomaticStrict>8

5. Future Scope

More such similar kind of correlation can be found out based on the java applications already developed using different tools and applying techniques like deep learning and comparing the two techniques with the accuracy prediction after data mining and deep learning techniques.

A juxtaposition of machine [23] and deep learning[22] techniques based on different language of generation viz. java, python, android smells can be detected and the model so generated can be used to predict the presence of such smells or vulnerabilities in future applications to make it easily maintainable and protect from data stolen due to glitches.

Data Availability
The varied java applications chosen to predict the correlation between the code smells and vulnerabilities are taken from the Github repository openly available for the purpose. The metrics computed and the dataset formulated using advisors cannot be released as it is under process for research work based on the deep learning techniques to be applied on the same.

Conflict of Interest
There is no conflict of interest involved in the process of the research conducted.

Funding Source
No funding source exists for the research project being carried out.

Author’s Contribution
The author is solely responsible for carrying out the research process from collection of data to the formulation of dataset and analysing the results and finding out a comparative relation between the code smells and vulnerabilities.

Acknowledgement
I would like to appreciate the guidance provided to me by my masters faculty in further enhancing and exploring the scope and providing the guidance to carry out the research work in this field of computer science and engineering.

References

[1] Fontana, F. A., Zanoni, M., Marino, A., & Mäntylä, M. V, “Code smell detection: Towards a machine learning-based approach”, *IEEE international conference on software maintenance,* pp. **396-399, 2013**.
[2] Alhazmi, O., Malaiya, Y., & Ray, I, “Security vulnerabilities in software systems: A quantitative perspective”, In *IFIP Annual Conference on Data and Applications Security and Privacy,* pp. **281-294, 2005.**
[3] KS, V. K, “A method for predicting software reliability using object oriented design metrics”, “*International Conference on Intelligent Computing and Control Systems (ICCS),* pp. **679-682, 2019.**
[4] Elia, I. A., Antunes, N., Laranjeiro, N., & Vieira, M, “An analysis of openstack vulnerabilities”, “*13th European Dependable Computing Conference (EDCC)*”, pp. **129-134, 2017.**
[5] Reutemann, G. H. B. P. P., Hall, I. H. W. M., Frank, E., & Witten, I. H, “The weka data mining software: An update”, *SIGKDD Explorations*, Vol. **11**, Issue. **1**, pp. **10-18, 2009.**
[6] Kirkby, R., & Frank, E, “WEKA Explorer User Guide for Version 3-4”, *University of Weikato*, pp.**3-4, 2002**.
[7] Di Nucci, D., Palomba, F., Tamburri, D. A., Serebrenik, A., & De Lucia, A, “Detecting code smells using machine learning techniques: are we there yet?”, *Ieee 25th international conference on software analysis, evolution and reengineering (saner)*, pp. **612-621, 2018.**
[8] Kim, D.K., “Finding bad code smells with neural network models” *International Journal of Electrical and Computer Engineering*, Vol. **7**, Issue. **6**, p**.3613, 2017.**

[9] Pessoa, T., Monteiro, M.P. and Bryton, S, " An eclipse plugin to support code smells detection" *arXiv preprint arXiv:1204.6492*, **2012.**

[10]_Sharma, S., & Rathore, M, "Comparison Study of Classification Techniques for Predicting Performance of Students Using Weka Environment", "*Rising Threats in Expert Applications and Solutions,* **(pp. 673-681), 2022.**

[11] Rezaei, E., Ghoreyshi, K., Dimitrov, Y., Sadique, K. M., & Campos, J, "Data Mining with WEKA", **2021**.

[12] Medeiros, N., Ivaki, N., Costa, P., & Vieira, M, "Vulnerable code detection using software metrics and machine learning", *IEEE Access*, *8*, **2020**.

[13] Pereira dos Reis, J., Brito e Abreu, F., de Figueiredo Carneiro, G., & Anslow, C, "Code smells detection and visualization: a systematic literature review", *Archives of Computational Methods in Engineering*, Vol. **29**, Issue.**1**, pp. **47-94, 2022.**

[14] Rattan, D., Bhatia, R., & Singh, M, "Software clone detection: A systematic review", *Information and Software Technology*, Vol. **55**, Issue.**7**, pp. **1165-1199, 2013.**

[15] Kaur, A, "A systematic literature review on empirical analysis of the relationship between code smells and software quality attributes", *Archives of Computational Methods in Engineering*, Vol. **27**, Issue. **4**, pp. **1267-1296, 2020.**

[16] Santos, J. A. M., Rocha-Junior, J. B., Prates, L. C. L., do Nascimento, R. S., Freitas, M. F., & de Mendonça, M. G, "A systematic review on the code smell effect", *Journal of Systems and Software*, Vol.**144**, pp. **450-477, 2018.**

[17] Elkhail, A. A., & Cerny, T, "On relating code smells to security vulnerabilities", *IEEE 5th intl conference on big data security on cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing,(HPSC) and IEEE intl conference on intelligent data and security (IDS)* pp. **7-12, 2019.**

[18] Aggarwal, K. K., Singh, Y., Kaur, A., & Malhotra, R, "Software Design Metrics for Object-Oriented Software, *J. Object Technol.*, Vol.**6**, Issue.**1**, pp. **121-138, 2007.**

[19] Dewangan, S., Rao, R.S., Mishra, A. and Gupta, M., 2022. Code Smell Detection Using Ensemble Machine Learning Algorithms. *Applied Sciences*, *12*(20), **p.10321 2022.**

[20] Sehgal, R., Mehrotra, D. and Nagpal, R, "Is refactoring a solution to resolve code smell?", *International Journal of System of Systems Engineering*, Vol.**12**, Issue.**4,** pp.**371-385. 2022.**

[21] Madeyski, L. and Lewowski, T., "Detecting code smells using industry-relevant data", *Information and Software Technology*, p.**107112. 2023.**

[22] S. D. Raut and S. A. Thorat, "Deep Learning Techniques: A Review," *International Journal of Scientific Research in Computer Science and Engineering,* vol.**8**, Issue.**1**, pp. **105-109, 2020.**

[23] Anoushka, Shivani Dubey, Vikas Singhal, "Student Grade Prediction by using Machine Learning Methods and Data Analytics Techniques," *International Journal of Scientific Research in Computer Science and Engineering*, vol.**10**, no. **6**, pp. **22-29, 2022.**

AUTHORS' PROFILE

Kritika- has completed her B.Tech and M.Tech in the years 2019 and 2022 respectively in Computer Science and Engineering and is currently working with Government of India. The author is certified as Cyber Hygiene Practitioner issued by Ministry of Electronics and Information Technology and has occupied certifications in the field of cyber security. The area of specialisation includes code smells, vulnerabilities, machine learning, deep learning and cyber security.

Int. J. of Scientific Research in
Biological Sciences
www.isroset.org

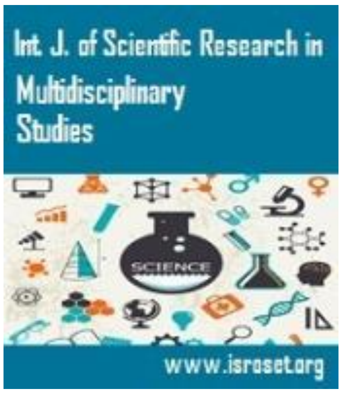
Int. J. of Scientific Research in
Chemical Sciences
www.isroset.org

Int. J. of Scientific Research in
Computer Science and
Engineering
www.isroset.org

World Academics Journal of
Engineering Sciences
ISSN:2348-635X
www.isroset.org

Journal of
Physics and Chemistry of Materials
ISSN:2348-6341
www.isroset.org

Int. J. of Scientific Research in
Mathematical and
Statistical Sciences
www.isroset.org

ISSN: 2349-3178 (Print),
ISSN: 2349-3186 (Online)
International Journal of
Medical Science
Research and Practice
Published by ISROSET

Int. J. of Scientific Research in
Multidisciplinary
Studies
www.isroset.org

Int. J. of Scientific Research in
Network Security
and Communication
e-ISSN: 2321-3256

World Academics Journal of
Management
ISSN: 2321-905X
www.isroset.org

Int. J. of Scientific Research in
Physics and
Applied Sciences
www.isroset.org

Int. J. of Computer
Sciences and Engineering
www.ijcseonline.org