Webology (ISSN: 1735-188X)
Volume 18, Number 6, 2021

# Multilevel Ensemble Model For Prediction Of Android Malware

**Ravneet Singh Bedi**[1] **and Prashant Singh Rana**[2]

[1]Computer Science and Engineering
Department, Mewar University, Gangarar,
Chittorgarh, Rajasthan, India.

[2]Computer Science and Engineering Department,
Thapar Institute of Engineering and Technology, Patiala, Punjab, India.

**Abstract.** Early identification of malicious applications can help the Android user register private data and device integrity. It is imperative to introduce a reliable system with high precision and effectiveness for predicting. In this study, features extracted from intermediatecode representations obtained using de compilation of APK file are used for providing requi- site input data to develop the models for predicting android malware applications. A new multi-tier set model is developed for the prediction of Android Malware. Under this overar- ching approach, seven different machine learning models are combined to predict Android malware. The proposed model reaches 97.56

**Keywords:** Machine learning models; Multilevel ensemble model; Regularized trees; An- droid Malware;

## 1 Introduction

According to India's digital project and Android operating system, the mobile phone has grown inpopularity and is becoming essential to the modern user. From January 2015 to December 2021,155 million new Internet users have been identified in India and 40 per cent of Indians use mobileInternet and smartphones. The digital economy will also grow as more and more private and public actors have opportunities to provide digital services. Android is now the most popular smartphone platform with 80% of global sales at the end of Q4 2021. By the end of December 2021, 6.5 millionAndroid apps on Google Play and smartphone users are using over 60 percent of these Androidapps. With the popularity of these apps, it also calls on cybercriminals to develop malicious appsto access important information from smartphones. According to the survey of G DATA securityexperts, 3 million new malicious apps in the fourth half of 2021 and 80% smartphone worldwidehad installed on them Android. These experts also observed that the new strain of malware every 5 seconds and cyber criminals find new ways and increasing strength to attack Android users [1]. It is challenging to predict malware apps during installation in smartphones. Also, it is non- trivial in some cases to replicate the problem that arises due to this malware apk [2]. Early identi-fication of such malware apps can help the android user to save private data and device integrity. Based on the recent study, researchers observed that the behaviors of Android apps could be ana-lyzed using static source code metrics extracted from intermediate code representations obtainedusing de compilation of APK file [2] [3]. However, there are three main technical challenges in developing an android malware prediction model using machine learning techniques.

- **Aggregation Measures**: To develop an android malware prediction model, the apk file is decompiled to get the intermediate code and source code metrics that may have a relationshipwith malware, are computed. In this work, features sets extracted from source code are used to measure the internal structure of the android applications. These features are computed atthe file level, and considerably easier for a developer to develop any prediction model at the file level. But the objective of this work is to predict malware and normal android applications. To achieve this, we need to apply aggregation measures on the file-level metrics to compute metrics at the system level. So, the aggregation measures are used to find the metrics at the system level, which will help develop android malware, prediction models.
- **High-Dimensional Data**: The performance of the classifiers also depends on features thatare considered as the input of the android malware prediction models. Selection of the relevant and suitable set of the significant uncorrelated feature is a technical challenge in the context of malware prediction.
- **Imbalanced Data**: The other technical challenge in building an android malware predictionmodel is that the data used for training the models are not balanced. A data is considered as balanced when the instances of the target class are approximately evenly distributed across various categories of the target class [4] [5]. An imbalance dataset is a dataset in which the instances of the target class are unevenly distributed across the target class categories. In thiswork, the considered dataset has not had an equal number of malware and normal apks.

The study investigates source code-based approaches for analyzing historical information databasesto uncover interesting patterns and knowledge that can support android users in identifying mal- ware applications. This study uses different machine learning algorithms to find interesting patterns by considering features extracted from source code metrics as input.

The paper is organized as follows: A hasty overview of the evaluated features, dataset, feature selection, machine learning models, and benchmark dataset are presented. The methodology and proposed model are explained with detailed implementation. The experiments, result analysis, comparison, and discussion are discussed with the conclusion and future work.

## 2 Literature Review and Background

The focus areas of this proposal are on the application of various sets of metrics such as source code metrics, word metrics, and network metrics to develop a model for predicting malware applications. Due to space limitations, I reviewed some closely related work to this research proposal.

Anshul Arora and Sateesh K Peddoju derived a malware detection model called NTP Droid using System Permissions and Network Traffic [6]. They have collected the features from System Permissions and Network Traffic of Android applications and used them as input to develop one hybrid model for predicting malware applications. They have considered the P-Growth algorithm to train this model and identify valuable patterns for predicting malware applications. They have also derived one model called Poster using the same features but trained using Unsupervised Learning and Supervised learning [1].

Arvind Mahindru and Paramvir Singh studied the dynamic permission of Android applications and proposed one model for detecting malware applications using Machine Learning Techniques [7]. They have experimented on 11000 Android applications and extracted 123 features based on the dynamic behavior of these applications. Arvind Mahindru and Paramvir Singh have considered these 123 features as input to train the android detection models using Decision Tree (J48), Naive Bayes (NB), Simple Logistic (SL), Random Forest (RF), and k-star for detecting malware appli- cations.

Shina Sheen et al. presented an approach for detecting malware applications using multi-feature collaborative decision fusion (MCDEF) [8]. They have analyzed different features based on API call and permission and considered these analyzed features as an input to train the malware prediction models using ensemble learning methods. They have validated these models on different categories of malware applications.

**Table 1**: Sample dataset of Android APPs

| **F1** | **F2** | **F3** | **F4 ——— F120** | **F121** | **F122** | **F123** | **Class** |
|---|---|---|---|---|---|---|---|
| 20.00 | 3.73 | 0.22 | 47.87 ——— -0.60 | -0.03 | 1176.35 | 9.44 | 1 |
| 118.00 | -0.82 | 0.35 | 8.00 ——— 0.44 | -0.59 | 533.64 | 3.85 | 1 |
| 29.50 | 2.61 | 0.40 | 53.96 ——— -0.33 | -0.08 | 2267.58 | 8.52 | 1 |
| 66.00 | 1.30 | 0.33 | 28.29 ——— 0.07 | -0.47 | 4212.79 | 8.24 | 0 |
| 137.50 | 0.32 | 0.31 | 11.07 ——— 0.19 | -0.39 | 1275.56 | 11.65 | 0 |
| 65.33 | 1.61 | 0.29 | 32.57 ——— -0.15 | -0.34 | 1630.90 | 8.23 | 0 |

## 3 Materials and Methods

### 3.1 Dataset and its features

To conduct an in-depth analysis of permission usage of Android applications, we collected 1600 of
.apk files, from Google's play store [6], app china [9], hiapk [10], Android [3], mumayi [11], gfan [12], pandaapp [4] and slideme [13]. These .apk files are collected after removing viruses, reported by Microsoft Windows Defender 10 and Virus Total 11. Virus Total 12 helps detect malware by antivirus engines and includes over 60 antivirus software. A number of 600 malware samples, from two different datasets [12] [14] are collected. In [12] Kadir et al. introduced an Android sample set of botnets, consisting of 14 different botnet families. Android Malware Genome project [14] containsa collection of different malware samples that cover the majority of existing Android malware families. Both malware and benign applications have been collected from the abovementioned sources until December 2018.

The dataset consists of 1600 rows and 123 features. Features consist of dynamic permission and behavior of Android applications. It consists of 800 normal apps and 800 abnormal (malware) apps. The glimpse of datasets is presented in Table 1. $F_1$ to $F_{123}$ are the features, and Class represent the type of the APP, i.e., normal (1) or malware(0).

### 3.2 Feature Measurement

This study obtained features extracted from intermediate code representations using the decom-pilation of APK files. CKJM tool [9] is used to calculate 18 different types of features. CKJM comprehensive tool is used for computing the object-oriented metrics at the class level. The byte code of the compiled Java files is processed to compute the object-oriented source code metrics. The definition of these considered metrics is given in [10].

### 3.3 Machine Learning Methods

To get better results, the models' parameters need to be tuned. The models used in the present study are described in Table 2 with required packages and their tuning parameters.

## 4 Methodology

The methodology is represented in Fig 1. In the first stage, android application packages (.apk) arecollected from different sources. In this work, we are using 1600 publicly available unique Android application packages (.apk) comprising of normal Android application packages and malicious

**Table 2:** Machine Learning Models

| Model | Method | Required Package | Tuning Parameter |
|---|---|---|---|
| Random Forest (RF) [15] | rf | random forest | mtry=2, ntree=500 |
| Support vector machine (SVM) [16] | ksvm | kernlab | kernel="rbfdot", type="C-svc" |
| Decision Tree [17] | rpart | None | Use surrogate=0, max surrogate=0 |
| Neural Network [18] | nnet | nnet | size=10 |
| Extreme Learning Machine (ELM) [19] | elmtrain | elmNN | nhid=10 |
| Avnnet [20] | avNNet | caret | size, linout, trace |
| Regularized Random Forest (RRF) [21] | RRF | RRF | None |

applications from different sources such as mumayi [22], gfan [7], Android Malware Genome Project [2], Droid Kin data set16 for this experiments.

The feature measurement is done in the second step and explained in Section 3.2. In thethird step, regularized random forest (RRF) [21] is used to get the subset of essential features. This process reduces the space complexity and time complexity and increases the accuracy of themodel. In the fourth step, the dataset is used to train the classifiers with their optimum tuning parameters. The used machine learning models are presented in Table 2. The models are combinedto get the proposed multilevel ensemble model as mentioned in Section 4.1. The proposed modelis described in Fig 2. Finally, the model's performance is evaluated on various parameters such asthe area under the curve (AUC), specificity, sensitivity, Gini, and accuracy with repeated k-fold cross-validation.

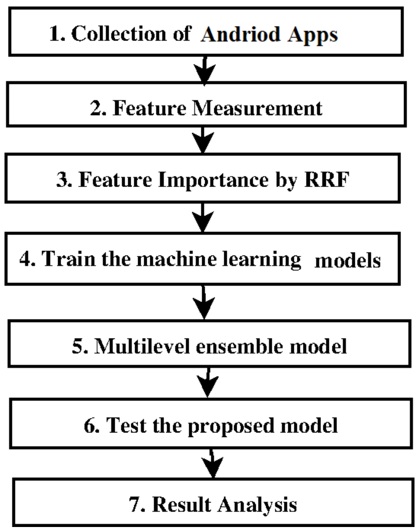
1. Collection of Andriod Apps
2. Feature Measurement
3. Feature Importance by RRF
4. Train the machine learning models
5. Multilevel ensemble model
6. Test the proposed model
7. Result Analysis

**Fig. 1:** Methodology of Proposed Model

### 4.1 Proposed Multilevel Ensemble Model

The ensemble is used to deal with the worst case of the model prediction. In the present work, the focus is on the false prediction and the true prediction of the model, and the multilevel ensemble model is used to deal with false and true predictions. Seven models i.e., Decision tree, RF, SVM,

ELM, Neural network, RRF, and av N Net are combined to get better accuracy as mentioned in Fig
2. All the models are trained on 70% of the dataset, and 30% is used for testing. The proposed model is divided into three phases, and all phases are explained below:

**Phase I:** The decision tree, ELM, neural network, SVM model are trained with 70% of the dataset and generate predictions from 30% of the dataset.

**Phase II:** The false predictions of two models (decision tree and ELM) from phase I are used to training the RF model. The false predictions of two models (neural network and SVM) from phase I are used to training the av N Net model.

**Phase III:** The false predictions from phase II and true predictions from Phase I are combined. This combined new dataset is used to train the RRF model that provides final predictions.

In this approach, true and false predictions, are refined to get an accurate proposed model. The purpose of using true prediction as the input of other models is to deal with false positive results. The data is traveled through seven models because these models perfectly learn the data to provide reliable and accurate results.

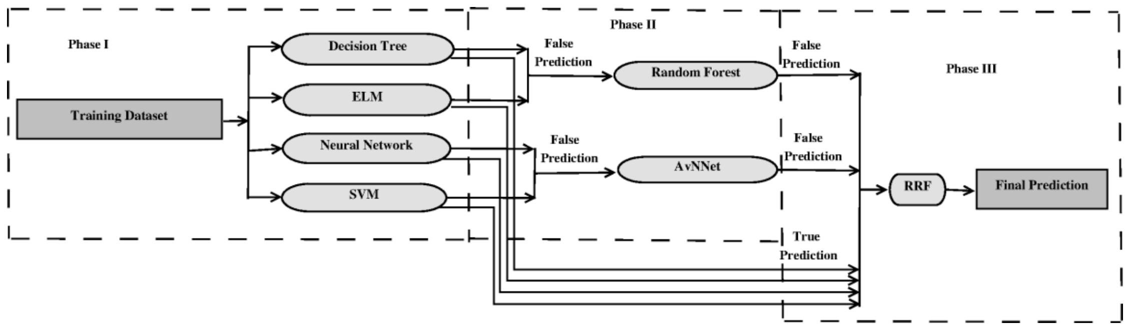

**Fig. 2:** Multilevel Ensemble Model

## 5 Model Evaluation Parameters

Various parameters such as Gini, accuracy, AUC, specificity, and sensitivity are calculated to evaluate the performance of the model. Repeated k-fold cross-validation is performed to test therobustness of the proposed model.

$$\text{Class} \sim f(F_1, F_2, F_3, F_4, ...., F_{121}, F_{122}, F_{123}) \quad (1)$$

### 5.1 Performance Evaluation

There are various parameters like Gini, accuracy, AUC, specificity, and sensitivity to measure theperformance of models. In the present study, all these parameters are used for evaluation.

**Gini Coefficient** Inequality in the distribution is measured through Gini coefficient. The rangeof Gini value is between 0 and 1. Like, model M has Gini value of 60%, and model D has Gini value of 45% then. Model M is considered an efficient model as compared to model D.

**AUC** The area under the curve (AUC) is calculated to measure the quality of the classifier. The amount of area under the receiver operating characteristics (ROC) curve is AUC. The model scoring high AUC compared to other models is considered an an efficient model. Its value is between 0 and 1. The quality of the model is good if it has an AUC value near to 1.

**Accuracy** Accuracy is calculated to measure the correctness of the classifier. The accuracy can be calculated as:

$$\text{Accuracy} = \frac{TP + TN}{\text{Total Data}} * 100 \quad (2)$$

**Sensitivity** Sensitivity(Sens) is also known as recall or true positive rate. It is the proportion ofactual positives which are correctly identified as positives by the classifier and is computed as:

$$\text{Sensitivity} = \frac{TP}{TP + FN} \quad (3)$$

**Specificity** Specificity (Spec) is also known as true negative Rate. It relates to the classifier'sability to identify negative results and is computed as:

$$\text{Specificity} = \frac{TN}{TN + FP} \quad (4)$$

TN: True negative, FP: False positive, TP: True positive, and FN: False negative

## 5.2 Repeated K-Fold Cross Validation

A large number of comparisons is always preferred to compare the performance of the model. To run K-fold cross-validation multiple times or increase the number of comparisons, repeated K-fold cross-validation is useful. In K-fold cross-validation, only k comparisons are acquired. In cross-validation, ,random data is provided to do the comparisons in each fold. Here, 10-fold cross- validation is repeated 3 times.

## 6 Result Analysis, Comparison and Discussion

The machine learning models are trained and evaluated on various parameters as mentioned in Table 3. Another problem is overfitting/underfitting. To deal with the overfitting/underfitting issue, the model should be cross-validated and tested on an independent dataset. If performance is foundto be consistent, then models are free from overfitting/underfitting. Overfitting is when the models learns too much, and underfitting is when the models learns too less. In cross-validation, modelsare executed n times, and accuracy is recorded if accuracy is highly fluctuating, then that model is overfitted/under fitted/biased. In the present work, repeated K fold cross-validation is used todescribe the consistency in the accuracy, which means the proposed model is not affected by these problems. For validation of the proposed model, a benchmark dataset is used and compared with the existing model by using various parameters such as Gini, AUC, accuracy, MCC, specificity, andsensitivity. The result concludes two things about the proposed model. First, the proposed model is free from overfitted/under fitted/biased issues. Second, the outcome of the proposed model is improved as compared to the existing technique.

Table 2 describes the machine learning models that are trained on the dataset with optimum tuning parameters. The dataset is partitioned into two parts, 70% and 30%. The prepared models are unknown to the 30% of the dataset. The proposed model is a combination of seven models that makes it a multilevel ensemble model, as discussed in Section 4.1. The models are evaluated

**Table 3:** Performance evaluation of machine learning models for Prediction of Android Malware

| SN | Model Name | Gini | Accuracy | AUC | Specificity | Sensitivity |
|---|---|---|---|---|---|---|
| 1 | **Random Forest** | 0.32 | 63.35 | 0.66 | 0.68 | 0.43 |
| 2 | **Av N Net** | 0.34 | 44.1 | 0.67 | 0.71 | 0.40 |
| 3 | **Decision Tree** | 0.23 | 60.87 | 0.61 | 0.61 | 0.45 |
| 4 | **RRF** | 0.30 | 62.73 | 0.65 | 0.66 | 0.44 |
| 5 | **Neural Network** | 0.14 | 57.14 | 0.57 | 0.56 | 0.44 |
| 6 | **ELM** | 0.05 | 44.1 | 0.52 | 0.44 | 0.50 |
| 7 | **SVM** | 0.33 | 61.49 | 0.66 | 0.70 | 0.41 |
| **8** | **Proposed model** | **0.34** | **61.40** | **0.67** | **0.57** | **0.41** |

on various parameters as mentioned in Table 3. From the results, it is concluded that the accuracyof the proposed model is increased as compared to the single model accuracy.

Table 4 describes the accuracy of the proposed model. The accuracy has been recorded by applying 10-fold cross-validation three times. For cross-validation, 70% of the dataset is used fortraining and 30% used for testing. Fig. 3 describes the accuracy of the proposed model 3 times in10 runs and shows the consistency in the accuracy of the proposed model.

**Table 4:** Repeated 10-Fold Cross Validation of Proposed Model

| Folds | Run 1 | Run 2 | Run 3 |
|---|---|---|---|
| 1 | 93.17 | 95.61 | 96.10 |
| 2 | 94.15 | 96.10 | 96.10 |
| 3 | 92.20 | 97.07 | 95.12 |
| 4 | 95.12 | 93.66 | 96.10 |
| 5 | 94.15 | 95.12 | 94.63 |
| 6 | 93.66 | 96.10 | 97.07 |
| 7 | 92.68 | 97.56 | 92.20 |
| 8 | 94.63 | 90.73 | 94.63 |
| 9 | 95.12 | 97.07 | 93.66 |
| 10 | 92.68 | 93.66 | 94.63 |

## 7 Conclusion

The proposed model increases the prediction accuracy of Android Malware prediction as compared to the existing technique. This work aims to find the impact of features extracted from the sourcecode of the .apk file, which denotes the internal structure of the software on the prediction of malware and normal android applications. In the present study, seven models i.e., decision tree, ELM, RF, neural network, SVM, avnnet, and RRF, are used to create a multilevel ensemble model. A novel multilevel ensemble model is developed for prediction, and it produces high accuracy, Gini, AUC, specificity, and sensitivity. The multilevel ensemble model is divided into 3 phases. In this approach, true and false predictions are used to get an an accurate proposed model. The benefit of using true prediction as the input of other models is to deal with false-positive results. The data is traveled through seven models because these models perfectly learn the data to provide reliable and

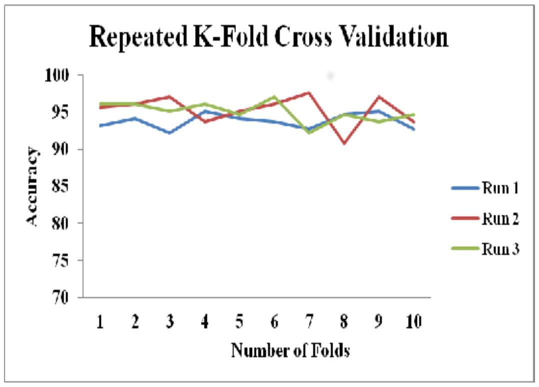

**Fig. 3:** Repeated K-Fold Cross Validation of Proposed Model

accurate results. To check the robustness of the proposed model, repeated k-fold cross-validation is used. We believe that using more properties and machine learning models with their optimized parameters produces even better outcomes.

## References

1. Anshul Arora and Sateesh K Peddoju. Minimizing network traffic features for android mobile mal- ware detection. In Proceedings of the 18th International Conference on Distributed Computing and Networking, pages 1–10, 2017.
2. Niall McLaughlin, Jesus Martinez del Rincon, Boo Joong Kang, Suleiman Yerima, Paul Miller, Sakir Sezer, Yeganeh Safaei, Erik Trickel, Ziming Zhao, Adam Doupé, et al. Deep android malware detection. In Proceedings of the seventh ACM on conference on data and application security and privacy, pages 301–308, 2017.
3. Ambra Demontis, Marco Melis, Battista Biggio, Davide Maiorca, Daniel Arp, Konrad Rieck, Igino Corona, Giorgio Giacinto, and Fabio Roli. Yes, machine learning can be

more secure! a case study on android malware detection. IEEE Transactions on Dependable and Secure Computing, 16(4):711–724, 2017.

4. Fabio Palomba, Dario Di Nucci, Michele Tufano, Gabriele Bavota, Rocco Oliveto, Denys Poshyvanyk, and Andrea De Lucia. Landfill: An open dataset of code smells with public evaluation. In 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories, pages 482–485. IEEE, 2015.
5. Julian A Ramos Rojas, Mary Beth Kery, Stephanie Rosenthal, and Anind Dey. Sampling techniques to improve big data exploration. In 2017 IEEE 7th symposium on large data analysis and visualization (LDAV), pages 26–35. IEEE, 2017.
6. Anshul Arora and Sateesh K Peddoju. Ntpdroid: a hybrid android malware detector using network traffic and system permissions. In 2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Sci- ence And Engineering (Trust Com/Big Data SE), pages 808–813. IEEE, 2018.
7. Arvind Mahindru and Paramvir Singh. Dynamic permissions based android malware detection using machine learning techniques. In Proceedings of the 10th innovations in software engineering conference, pages 202–210, 2017.
8. Shina Sheen, R Anitha, and V Natarajan. Android based malware detection using a multi feature collaborative decision fusion approach. Neurocomputing, 151:905–912, 2015.
9. Shyam R Chidamber and Chris F Kemerer. Towards a metrics suite for object oriented design. In Conference proceedings on Object-oriented programming systems, languages, and applications, pages 197–211, 1991.
10. Neville I. Churcher, Martin J. Shepperd, S Chidamber, and CF Kemerer. Comments on" a metrics suite for object oriented design. IEEE Transactions on software Engineering, 21(3):263–265, 1995.
11. Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. Extreme learning machine: theory and ap- plications. Neurocomputing, 70(1-3):489–501, 2006.
12. Andi Fitriah Abdul Kadir, Natalia Stakhanova, and Ali Akbar Ghorbani. Android botnets: What urls are telling us. In International Conference on Network and System Security, pages 78–91. Springer, 2015.
13. Lov Kumar, Shashank Mouli Satapathy, and Aneesh Krishna. Application of smote and lssvm with various kernels for predicting refactoring at method level. In International Conference on Neural Information Processing, pages 150–161. Springer, 2018.
14. Yajin Zhou and Xuxian Jiang. Dissecting android malware: Characterization and evolution. In 2012 IEEE symposium on security and privacy, pages 95–109. IEEE, 2012.
15. Andy Liaw and Matthew Wiener. Classification and regression by random forest. R news, 2(3):18–22, 2002.
16. S. Sathiya Keerthi and Elmer G Gilbert. Convergence of a generalized smo algorithm for svm classifier design. Machine Learning, 46(1-3):351–360, 2002.

17. Paul D Berger, Arthur Gerstenfeld, and Amy Z Zeng. How many suppliers are best? a decision-analysis approach. Omega, 32(1):9–15, 2004.

18. Brian Ripley, William Venables, and Maintainer Brian Ripley. Package ‘nnet’. 2016.

19. Torsten Hothorn, Peter Buehlmann, Thomas Kneib, Matthias Schmid, Benjamin Hofner, Fabian Sobotka, Fabian Scheipl, and Maintainer Benjamin Hofner. Package ‘mboost’. 2016.

20. Chris Keefer Williams, Allan Engelhardt, Tony Cooper, Zachary Mayer, Andrew Ziem, Luca Scrucca, Yuan Tang, Can Candan, and Maintainer Max Kuhn. Package ‘caret’. 2016.

21. Suggests R Color Brewer, Houtao Deng, and Maintainer Houtao Deng. Package ‘rrf’. 2013.

22. Wei Li and Sallie Henry. Maintenance metrics for the object oriented paradigm. In [1993] Proceedings First International Software Metrics Symposium, pages 52–60. IEEE, 1993.