Replication and Extension of Schnappinger's Study on Human-level Ordinal Maintainability Prediction Based on Static Code Metrics

Sébastien Bertrand [1,2,4], Silvia Ciappelloni [4], Pierre-Alexandre Favier [1,2,3], Jean-Marc André [1,2,3]

[1] Université de Bordeaux
[2] IMS Laboratory
[3] ENSC, Bordeaux INP
[4] onepoint, Sud-Ouest

s.bertrand@groupeonepoint.com

Résumé

Dans le cadre d'un projet de recherche sur l'évaluation de la maintenabilité des logiciels en collaboration avec l'équipe de développement, nous avons voulu explorer les dissensions entre les développeurs et le facteur confondant de la taille. A cette fin, cette étude a reproduit et étendu une étude récente de Schnappinger et al. avec la partie publique de son jeu de données et les métriques extraites de l'outil basé sur les graphes Javanalyser. L'ensemble du processus de traitement a été automatisé, de l'extraction des métriques à l'entraînement des modèles d'apprentissage automatique. L'étude a été étendue en prédisant la maintenabilité continue pour prendre en compte les dissensions. Puis, tous les entraînements ont été dupliqués pour évaluer l'influence globale de la taille de la classe. Au final, l'étude originale a été reproduite avec succès. De plus, de bonnes performances ont été obtenues pour la prédiction de la maintenabilité continue. Enfin, la taille de la classe n'était pas suffisante pour une prédiction fine de la maintenabilité. Cette étude montre la nécessité d'explorer la nature de ce qui est mesuré par les métriques du code. Elle constitue également la première étape dans la construction d'un modèle de maintenabilité.

Mots-clés

Maintenabilité des logiciels, prédiction de la maintenabilité, classification ordinale, jugement d'expert, apprentissage automatique.

Abstract

As part of a research project concerning software maintainability assessment in collaboration with the development team, we wanted to explore dissensions between developers and the confounding effect of size. To this end, this study replicated and extended a recent study from Schnappinger et al. with the public part of its dataset and the metrics extracted from the graph-based tool Javanalyser. *The entire processing pipeline was automated, from metrics extraction to the training of machine learning models. The study was extended by predicting the continuous maintainability to take account of dissensions. Then, all experimental shots were duplicated to evaluate the overall influence of the class size. In the end, the original study was successfully replicated. Moreover, good performance was achieved on the continuous maintainability prediction. Finally, the class size was not sufficient for fine-grained maintainability prediction. This study shows the necessity to explore the nature of what is measured by code metrics, and is also the first step in the construction of a maintainability model.*

Keywords

Software Maintainability, Maintainability Prediction, Ordinal Classification, Expert Judgment, Machine Learning.

1 Introduction

According to the ISO 25010 [20], software maintainability is defined as the efficiency with which the development team can fix a defect within a software, or implement an evolution according to a change of the needs. Software maintenance represents an important part of software development costs in time [5, 3, 10, 33, 9]. Assessing the maintainability of a piece of software is therefore important to ensure the control of these costs.

Our research project concerns software maintainability assessment in collaboration with the development team. Our goal is to pinpoint maintainability problems within a *Java* program, while taking into account team preferences and potential disagreements between developers.

Most studies use static code metrics to predict software maintainability [1, 12]. Very often, they study the source code at the class-level, because it is both a convenient and sizeable way to talk about code [35]. One group of studies uses the number of changed code lines as a proxy for the maintenance effort [22, 25, 29, 31, 39, 40]. Another group of studies is based on the prediction of expert assessments [18, 19, 34, 38]. However, the relation between the number of changed code lines and maintainability is not established [35]. Kafura and Reddy even states that developers tend to avoid modification of complex part of the program during maintenance [21]. That is why, in this work, we focused on prediction of expert assessments.

A study from Schnappinger *et al.* [37] explore human-level maintainability prediction based on static code metrics, specifically the distinction between easy and hard-to-maintain code and a more fine-grained ordinal classification problem. This study is based on a recent high quality software maintainability dataset [35, 36], also built by Schnappinger *et al.* . This dataset targets *Java* classes and represents maintainability as an ordinal four-part scale percentage agreement. However, only the open-source part of the dataset is publicly available. Additionally, due to a very large number of existing tools [30, 2] and their lack of agreement [4, 28, 8], the extraction of metrics from the dataset is quite hazardous. In addition, the original study uses numerous tools, including unpublished ones [37]. So to rationalize the extraction of metrics, we chose to use *Javanalyser* [4] which we developed to extract metrics solely based on the structure of the source code. *Javanalyser* [1] is an open-source graph-based static code analysis tool that leverages declarative programming as formal definition of metrics. Moreover, the study only reports performance results, but the implementation to train the machine learning models is not available. Therefore, our first goal was to check we could independently replicate the findings of Schnappinger *et al.* with a full open-source setup available to the community, including *Javanalyser* for metrics extraction.

Then, the study from Schnappinger *et al.* models maintainability prediction as an ordinal classification problem. The maintainability category of a *Java* class is determined by taking the most probable among experts as the response variable. However, while building their dataset, Schnappinger *et al.* notes that disagreement between experts is frequent [35]. To be able to work in collaboration with development teams, we wanted to predict potential disagreement. This is why we chose to additionally study a continuous representation of software maintainability.

Finally, the class size in lines of code is highly correlated to many metrics [11, 13, 23, 26, 41] and is one of the most used metric for maintainability prediction [37]. From an engineering point of view, this is very surprising because poor maintainability seems intuitively more connected to a poorly built control flow or data flow. Moreover, modern integrated development environments make it easy to extract code from a class to a new one. Therefore, class size may be an unreliable measure to detect complex maintainability problems. So, we tried to assess its overall influence when predicting maintainability with this dataset.

In summary, the research questions replicated from the original study are :

- **RQ1** : How well can classifiers distinguish between easy and hard-to-maintain code ?
- **RQ2** : How good is the performance of the machine learning models considering an ordinal label ?

The extension part of our study is related to these two research questions :

- **RQ3** : How good is the performance of the machine learning models considering continuous maintainability ?
- **RQ4** : What is the overall influence of class size on maintainability prediction ?

In this study, we extracted a total of 33 metrics from the dataset with *Javanalyser*, and we replicated the study of Schnappinger *et al.* following the same overall method. Then we implemented an additional setup using regressors instead of classifiers to test performance on continuous maintainability. Finally, we duplicated the experiments with only the class size as input, to assess its overall influence. Lastly, we launched a total of almost 20 000 experimental shots, each consisting of 150 individual machine learning trainings. Along this study, we logged every training results to help potential further research. To allow for the definition and evaluation of new metrics and maintainability models, we open-sourced the implementation and the results of our experiments under the MIT licence (see section 3).

2 Method

This section begins by presenting the core replication of Schnappinger *et al.* 's study, that is the dataset and the base experimental setup. Then the extension is presented with the prediction of a continuous maintainability and the size experiments.

2.1 Replication

2.1.1 Dataset

The present study uses the public part of Schanappinger's dataset [35, 36]. Basically, it consists of a listing of 304 classes with their source code, drawn from 5 open-source projects : *ArgoUML* [2], *Art of Illusion* [3], *Diary Management* [4], *JUnit 4* [5], and *JSweet* [6]. Each *Java* class is labelled along 5 axis : readability, understandability, complexity, modularity and overall maintainability. Each *Java* class was assessed by several experts independently of its relation to other classes.

As the original study, this work focused on the overall maintainability. It is represented as a set of 4 probabilities corresponding to the evaluation of the experts on a 4-class Likert-scale. Each probability corresponds to the ratio over the total for this *Java* class. The original study refers to these probabilities with the labels *Class-A*, *Class-B*, *Class-C*, and *Class-D*, where *Class-A* is highly maintainable and *Class-D* is poorly maintainable. The table 1 shows that this dataset is heavily unbalanced, both in terms of maintainability labels distribution and project data points. For instance, *Diary Management* has only 11 data points, none with a *Class-D* maintainability assessment.

The extraction of metrics was done with *Javanalyser* [4]. Some minor modifications of the source code of *ArgoUML*

1. https://gitlab.com/onepoint/research/javanalyser

2. https://github.com/argouml-tigris-org/argouml
3. http://www.artofillusion.org/
4. https://sourceforge.net/projects/diarymanagement/
5. https://junit.org/junit4/
6. http://www.jsweet.org/

TABLE 1 – Dataset distribution

Project	Class-A	Class-B	Class-C	Class-D	Total
ArgoUML	34	25	11	4	74
Art of Illusion	10	20	25	18	73
Diary Management	7	2	2	0	11
JUnit 4	60	12	1	0	73
JSweet	63	5	2	3	73
Total	174	64	41	25	**304**

and *Art of Illusion* were necessary to make them parsable. Every modification is provided on the public repository as *diff* files. *Javanalyser* builds the graph of the code by folding and simplifying the abstract syntax tree, and then queries that graph to extract metrics. As a result, it computes a slightly different version of class size in lines of code, which is the number of statements (NOS), that is a formal count of statement nodes within the code graph. It is worth noting that even such a simple metric has many variants among metric tools [4], whereas *Javanalyser* implements formal definition of metrics.

2.1.2 Experimental setup from Schnappinger *et al.*

Here is presented a short version of the experimental setup of the study of Schnappinger *et al.* [37]. Their study presents two classification problems :

- A binary separation problem to distinguish between easy and hard-to-maintain code, where *Class-A* and *Class-B* are considered to be maintainable, whereas *Class-C* and *Class-D* are not ;
- An ordinal classification problem to predict the majority class assigned by the experts.

The original study preprocesses metrics (features) in three ways. First, oversampling is used to account for the unbalanced dataset. As Schnappinger *et al.* , k-means-SMOTE was used with the implementation described in [24] and set *k=2* due to the small dataset. Second, normalisation or standardisation of features is applied to potentially improve performance. Third, feature selection based on the mutual information between the metrics and the target is leveraged to select only a subset of available features. Each of these preprocessing techniques was implemented as a conditional setting and can be combined.

In their pre-study, Schnappinger *et al.* identify the six most promising algorithms for their setup [37] : Gradient Boosting [15], Ada-Boost [17], Extremely Randomized Trees [16], Logistic Regression, Random Forests [7] and the K-Nearest Neighbor classifier. Additionally, for the ordinal classification problem, they use 7 metamodels based on these base classifiers (or their regressor versions). One is the binary decomposition proposed by Frank and Hall [14]. The others are proposed by Schnappinger *et al.* : three chained binary classifiers, two probabilities classifiers and a rounded regressor classifier. For comparison, a baseline classifier that always predicts the majority class was implemented for each problem. All these classifiers were implemented using *scikit-learn* [32].

The original study uses many performance metrics to evaluate the models. The binary classification problem reports its results with the F-score, and the Area Under the receiver operating characteristic Curve (AUC) which is considered as a better choice for binary classification [6]. The ordinal classification problems uses micro-averaged accuracy (ACC), Cohen's Kappa (Cκ) and Matthews Correlation Coefficient (MCC). It also uses Mean-Square Error (MSE) by defining all intervals on the ordinal scale to be 1.

Every experiment was fully configured by a CSV file, which also stored the results. Each experiment was randomly sampled to explore the hyperparameters space without being too time-consuming. In total, $20,000$ experiments were sampled. Each experiment was repeated with 30 different random seeds. Like the original study, project-wise cross-validation was used, However the performance depends heavily on the chosen test project, because the dataset is heavily unbalanced between projects. That is why, a shuffled stratified 5-fold cross-validation was implemented for better statistical comparison. The reported results correspond to an average of these runs along the standard deviation.

2.2 Extension

2.2.1 Continuous maintainability

After the core replication, the problem of the continuous maintainability prediction was added. Continuous maintainability is defined as the expected value of each maintainability class. The scores ranging from 0 for *Class-A* to 3 for *Class-D* were assigned. This method allowed us to take into account disagreement between experts, for instance a continuous maintainability of 2.5 may correspond to half the experts assigning *Class-C* and the other half *Class-D*. Disagreement between experts could help to diagnose nontrivial maintainability problem.

For this problem, a baseline classifier that always predicts whole dataset expected maintainability was built, which is a maintainability of 0.75. As the other problems, the Mean-Square Error (MSE) was used for comparison. The models were also evaluated against the Mean Absolute Error (MAE), the Median Absolute Error (MedAE) and the R^2 score (R^2).

2.2.2 Size experiments

Finally, the overall influence of class size on maintainability prediction was explored. For that purpose, each experiment was duplicated and fed only the class size to the models. Moreover, as the variance with the project-wise cross-validation was quite important, the shuffled stratified 5-fold cross-validation was necessary for a better statistical com-

TABLE 2 – Results of the project-wise binary separation

Classifier	F-Score	AUC
Average expert	0.88	0.83
Baseline	0.87 ±0.14	0.50 ±0
Baseloc	0.92 ±0.07	0.67 ±0.17
Original study	0.91	0.82
This study	0.93 ±0.06	0.90 ±0.10

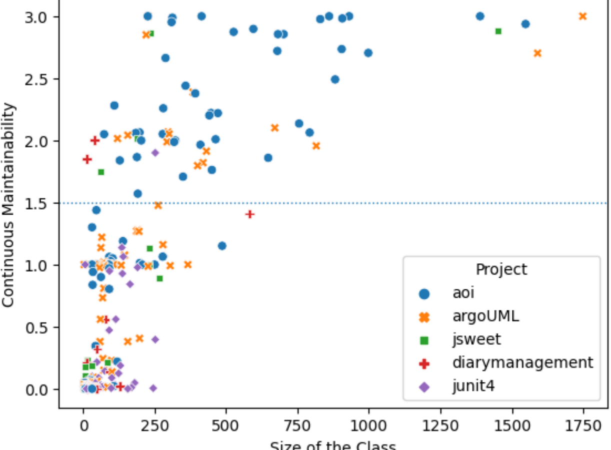

FIGURE 1 – The influence of size on maintainability

parison.
To truly challenge the prediction power of the trained models, some additional baseline classifier were implemented, hereafter referred to as *'baseloc'* classifiers. They adapt their prediction in function of the class size. For instance, for the binary separation problem, the baseloc classifier predicts good maintainability if the size is under 275 lines of code, otherwise it predicts poor maintainability.

3 Results

This section presents the best results of nearly 20 000 experimental shots, each consisting of 150 individual machine learning trainings, that is 5 folds (project-wise or stratified) times 30 seeds. The dataset, the metrics extracted, the processing code, and all the results are available under the free (as in freedom) MIT licence at `https://gitlab.com/onepoint/research/maintainability-dataset-analysis`.
All our results are presented with the notation *mean* ±*std*, *std* being the standard deviation. As the original study, the tables 2, 3, 4, 5 present the best score that was obtained by a classifier, that is two scores on the same row may not have been performed by the same classifier.

The table 2 presents the replication of the first problem from Schnappinger *et al.* 's study. The table 3 presents the replication of the ordinal classification. Like in the original paper, the measured performance depends heavily on the test project. That is why, k-fold cross validation results were included to overcome differences between projects.

Continuous maintainability ranges from 0 to 3, lower being better. As shown by the figure 1, the increase of class size clearly set a floor value for maintainability. The table 4 presents the results of continuous maintainability prediction. The performances of a perfect ordinal classifier is included for comparison with the ordinal classification problem. This problem was not tested by Schnappinger *et al.* 's study.

Finally, the table 5 presents the comparison of the performance of models when fed with only the class size *vs.* all metrics. This table is meant to be read vertically to compare baseline, baseloc, size-only and all-metrics classifiers for each problem. For simplicity, only the k-fold case is presented, the variability on the project-wise case being too large.

4 Discussion

This section begins by discussing class-level maintainability prediction problems. Then, disagreement between experts will be addressed as they are at the heart of these problems. Finally, the overall influence of class-size will shed a new light on the way forward.

4.1 Class-Level Maintainability Prediction

As shown by tables 2 and 3, the study of Schnappinger *et al.* [37] was successfully replicated with the metrics extracted by *Javanalyser*. On the binary separation problem, the best classifiers outperformed the baseline and baseloc classifiers, and an average expert. On the ordinal classification problem, the best classifiers outperformed the baseline and baseloc classifiers, and reached human-level performance. Moreover, concerning the continuous maintainability problem extension, the best classifiers outperformed the baseline and baseloc classifiers. There is no data concerning an average expert on this last problem. Thus, with respect to the first three research questions (**RQ1**, **RQ2**, **RQ3**), trained models performed well on binary separation, ordinal classification and continuous maintainability regression (respectively).
However, the addition of the standard deviation showed that the observed performance with project-wise cross-validation was highly unstable. As a reminder, the standard deviation needs to be tripled to give a 99% confidence interval. This was due to the unbalanced dataset (see table 1), indeed a k-fold cross-validation approach shows better deviations. In fact, oversampling cannot compensate for imbalance between projects when using project-wise cross-validation. Moreover, the table 3 shows that the results are very close to the original study with a k-fold cross-validation setting. The original study includes 9 projects within its dataset (5 open-source and 4 closed-source). To reach production-grade standard deviations, the training dataset would need *Java* classes drawn from more projects, until no differences are observed between project-wise and k-fold cross-validation.

4.2 Dissent Between Experts

The dataset is built with a four-part Likert scale to not overwhelm the experts [35]. However, when a class is globally maintainable, a minor problem could still be present

TABLE 3 – Results of the ordinal classification

Classifier	ACC	MSE	Cκ	MCC
Average expert	0.70	0.41	0.53	0.53
Project-wise				
Baseline	0.58 ±0.27	1.39 ±1.30	0 ±0	0 ±0
Baseloc	0.68 ±0.15	0.53 ±0.43	0.37 ±0.23	0.38 ±0.24
Original study	0.73	0.31	0.51	0.53
This study	0.74 ±0.12	0.39 ±0.32	0.46 ±0.18	0.47 ±0.18
K-Fold				
Baseline	0.57 ±0.01	1.49 ±0.03	0 ±0	0 ±0
Baseloc	0.73 ±0.05	0.36 ±0.08	0.54 ±0.08	0.54 ±0.08
Original study	0.75	0.30	0.60	0.60
This study	0.77 ±0.04	0.27 ±0.06	0.61 ±0.07	0.61 ±0.07

TABLE 4 – Results of the continuous maintainability regression

Classifier	MSE	MAE	MedAE	R^2
Project-wise				
Baseline	0.85 ±0.50	0.79 ±0.18	0.82 ±0.20	-0.67 ±0.66
Baseloc	0.39 ±0.25	0.43 ±0.15	0.24 ±0.07	0.21 ±0.46
This study	0.28 ±0.22	0.33 ±0.14	0.19 ±0.12	0.41 ±0.44
K-Fold				
Baseline	0.91 ±0.04	0.80 ±0.02	0.73 ±0.01	0 ±0.01
Baseloc	0.29 ±0.07	0.37 ±0.05	0.19 ±0.01	0.68 ±0.08
This study	0.18 ±0.05	0.26 ±0.04	0.10 ±0.04	0.80 ±0.06
Perfect ordinal	0.02	0.08	0.02	0.98

TABLE 5 – Size-Only vs All-Metrics k-fold results comparison

Classifier	**Binary sep.** (AUC)	**Ordinal class.** (MSE)	**Continuous maint.** (MSE)
Average expert	0.83	0.41	—
Baseline	0.50 ±0	1.49 ±0.03	0.91 ±0.04
Baseloc	0.84 ±0.05	0.36 ±0.08	0.29 ±0.07
Size-Only	0.95 ±0.03	0.35 ±0.08	0.27 ±0.06
All-Metrics	0.97 ±0.02	0.27 ±0.06	0.18 ±0.05

within it. For instance a method could be slightly too long and should be split in two. Such problems would lead to disagreement between expert, some assigning the *Class-A* and some the *Class-B*.

On a continuous scale, the expected maintainability of a class subject to disagreement is between two integers. This disagreement occurs quite often as shown by the figure 1. When building their dataset, Schnappinger *et al*. report disagreement between experts in 73.4% of the ratings [35]. By measuring the distance between ratings, they estimate that significant disagreement occurs in 17.2% of the cases, and strong disagreement occurs in 1.2% of the cases.

A perfect ordinal classifier would have a MSE of 0.02 on the continuous maintainability prediction (and reciprocally). This shows that these problems are not far apart. However, the MSE significantly improves from the ordinal classification problem to the continuous maintainability problem. This shows that predicting contentious cases is more effective than predicting the winner (the majority class). Despite this fact, a coarse binary separation to detect the most problematic classes is very effective. This can be seen on the figure 1 by drawing a horizontal line going through the middle of the continuous maintainability scale (1.5).

4.3 The Overall Influence of Class-Size

Size-only models show surprisingly effective performances (see table 5). In fact on the binary separation and the ordinal classification, size-only classifiers outperformed an average expert. On the binary separation problem, other metrics only marginally improve performances. Concerning the forth research question (**RQ4**), this confirms that class size is a very effective metric to predict maintainability.

There is no consensus on the confounding effect of class size [11, 13, 23]. Kitchenham argues that each line of code as a whole has the same probability of exhibiting a defect [23]. Nevertheless, table 5 shows other metrics help to improve performances on the ordinal classification and continuous maintainability problems. Thus, whatever the correlation with class size, other metrics tend to be useful for finer problems.

On that matter, Lemberger and Morel states that aggregation of metrics is not natural [27]. From that point of view, it is not obvious to compute the class-level cyclomatic complexity by summation of the complexity of its methods. It would clearly be misleading to define the intelligence quotient of a team by the sum of the intelligence quotients of its members. The definition of scale-invariant metrics would be a necessary step to assess the influence of each of them.

5 Conclusions

This study successfully replicated the study of Schnappinger *et al*. on human-level ordinal maintainability prediction [37]. However, the standard deviation remained very high for the project-wise ordinal classification problem. All the processing code, the dataset and the results are publicly available to allow further research. [7] The extension of the study with the continuous maintainability problem shows that all baseline and baseloc classifiers are outperformed by trained models. Finally, the analysis showed that models trained with only the class size are very efficient to detect coarse-grained maintainability problems and surprisingly outperformed an average expert. However, the class size is not enough when the problem is sufficiently complex, like the ordinal classification or the continuous maintainability. This study shows that there is a need to better design the maintainability prediction problem and to better define metrics in order to go beyond class-level analysis and to be able to pinpoint maintainability problems within classes.

Future works include building size-robust datasets with classes from many projects. Designing better scale-invariant metrics for static code analysis will be essential to go further in maintainability prediction and analysis. Then, executing controlled experiments to assert the individual influence of these metrics over the maintainability would be very insightful to complement the experimental data collected in real situations. In the end, such metrics should be part of a wider maintainability model.

Acknowledgments

We thank our collaborators at *onepoint* [8] for their insightful advices, in particular Alexandra Delmas, Jérôme Fillioux, Denis Maurel, Sylvain Métayer, and Guillaume Meurisse.

Références

[1] Hadeel Alsolai and Marc Roper. A systematic literature review of machine learning techniques for software maintainability prediction. *Information and Software Technology*, 119 :106214, March 2020.

[2] Luca Ardito, Riccardo Coppola, Luca Barbato, and Diego Verga. A Tool-Based Perspective on Software Code Maintainability Metrics : A Systematic Literature Review. *Scientific Programming*, 2020 :1–26, August 2020.

[3] Rajiv D. Banker, Srikant M. Datar, Chris F. Kemerer, and Dani Zweig. Software complexity and maintenance costs. *Communications of the ACM*, 36(11) :81–94, November 1993. `https://doi.org/10.1145/163359.163375`.

[4] Sébastien Bertrand, Pierre-Alexandre Favier, and Jean-Marc André. Building an operable graph representation of a Java program as a basis for automatic software maintainability analysis. In *EASE '22 : Proceedings of the International Conference on Evaluation and Assessment in Software Engineering 2022*, EASE 2022, pages 243–248, Gothenburg, Sweden, June 2022. Association for Computing Machinery. `https://doi.org/10.1145/3530019.3534081`.

7. `https://gitlab.com/onepoint/research/maintainability-dataset-analysis`
8. `https://www.groupeonepoint.com/`

[5] Barry W. Boehm, John R. Brown, and M. Lipow. Quantitative evaluation of software quality. In Raymond T. Yeh and C. V. Ramamoorthy, editors, *Proceedings of the 2nd International Conference on Software Engineering, San Francisco, California, USA, October 13-15, 1976*, pages 592–605. IEEE Computer Society, 1976. http://dl.acm.org/citation.cfm?id=807736.

[6] Andrew P. Bradley. The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7) :1145–1159, July 1997. https://www.sciencedirect.com/science/article/pii/S0031320396001422.

[7] Leo Breiman. Random Forests. *Machine Learning*, 45(1) :5–32, October 2001. https://doi.org/10.1023/A:1010933404324.

[8] Dennis Breuker, Jacob Brunekreef, Jan Derriks, and Ahmed Nait Aicha. Reliability of software metrics tools. page 14, November 2009.

[9] Celia Chen, Reem Alfayez, Kamonphop Srisopha, Barry Boehm, and Lin Shi. Why Is It Important to Measure Maintainability and What Are the Best Ways to Do It ? In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, pages 377–378, May 2017.

[10] Don Coleman, Bruce Lowther, and Paul Oman. The application of software maintainability models in industrial software systems. *Journal of Systems and Software*, 29(1) :3–16, April 1995.

[11] Khaled El Emam, S. Benlarbi, N. Goel, and S. N. Rai. The confounding effect of class size on the validity of object-oriented metrics. *IEEE Transactions on Software Engineering*, 27(7) :630–650, July 2001.

[12] Sara Elmidaoui, Laila Cheikhi, Ali Idri, and Alain Abran. Empirical Studies on Software Product Maintainability Prediction : A Systematic Mapping and Review. *e-Informatica Software Engineering Journal*, 13(1) :141–202, 2019.

[13] W.M. Evanco. Comments on "The confounding effect of class size on the validity of object-oriented metrics". *IEEE Transactions on Software Engineering*, 29(7) :670–672, July 2003.

[14] Eibe Frank and Mark Hall. A Simple Approach to Ordinal Classification. In Luc De Raedt and Peter Flach, editors, *Machine Learning : ECML 2001*, volume 2167, pages 145–156, Freiburg, Germany, 2001. Springer.

[15] Jerome H. Friedman. Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38(4) :367–378, February 2002. https://www.sciencedirect.com/science/article/pii/S0167947301000652.

[16] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine Learning*, 63(1) :3–42, April 2006. https://doi.org/10.1007/s10994-006-6226-1.

[17] Trevor Hastie, Saharon Rosset, Ji Zhu, and Hui Zou. Multi-class AdaBoost. *Statistics and Its Interface*, 2(3) :349–360, 2009. https://www.intlpress.com/site/pub/pages/journals/items/sii/content/vols/0002/0003/a008/abstract.php.

[18] Péter Hegedűs, Tibor Bakota, László Illés, Gergely Ladányi, Rudolf Ferenc, and Tibor Gyimóthy. Source Code Metrics and Maintainability : A Case Study. In Tai-hoon Kim, Hojjat Adeli, Haeng-kon Kim, Heau-jo Kang, Kyung Jung Kim, Akingbehin Kiumi, and Byeong-Ho Kang, editors, *Software Engineering, Business Continuity, and Education*, Communications in Computer and Information Science, pages 272–284, Berlin, Heidelberg, 2011. Springer.

[19] Péter Hegedűs, Gergely Ladányi, István Siket, and Rudolf Ferenc. Towards Building Method Level Maintainability Models Based on Expert Evaluations. In Tai-hoon Kim, Carlos Ramos, Haeng-kon Kim, Akingbehin Kiumi, Sabah Mohammed, and Dominik Ślęzak, editors, *Computer Applications for Software Engineering, Disaster Recovery, and Business Continuity*, Communications in Computer and Information Science, pages 146–154, Berlin, Heidelberg, 2012. Springer.

[20] ISO/IEC. Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models. Standard ISO/IEC 25010 :2011, ISO/IEC, March 2011. https://www.iso.org/standard/35733.html.

[21] Dennis Kafura and Geereddy R. Reddy. The Use of Software Complexity Metrics in Software Maintenance. *IEEE Transactions on Software Engineering*, SE-13(3) :335–343, March 1987.

[22] Arvinder Kaur and Kamaldeep Kaur. Statistical comparison of modelling methods for software maintainability prediction. *International Journal of Software Engineering and Knowledge Engineering*, 23(06) :743–774, August 2013. https://www.worldscientific.com/doi/abs/10.1142/S0218194013500198.

[23] Barbara Kitchenham. What's up with software metrics ? - A preliminary mapping study. *Journal of Systems and Software*, 83(1) :37–51, 2010.

[24] György Kovács. Smote-variants : A python implementation of 85 minority oversampling techniques. *Neurocomputing*, 366 :352–354, November 2019. https://www.sciencedirect.com/science/article/pii/S0925231219311622.

[25] Lov Kumar, Debendra Kumar Naik, and Santanu Kumar Rath. Validating the Effecti-

veness of Object-Oriented Metrics for Predicting Maintainability. *Procedia Computer Science*, 57 :798–806, January 2015. https://www.sciencedirect.com/science/article/pii/S1877050915020086.

[26] Meir Manny Lehman, Dewayne E. Perry, and Juan F. Ramil. Implications of evolution metrics on software maintenance. In *1998 International Conference on Software Maintenance, ICSM 1998, Bethesda, Maryland, USA, November 16-19, 1998*, page 208. IEEE Computer Society, 1998.

[27] Pirmin Lemberger and Médéric Morel. Two Measures of Code Complexity. pages 195–206. January 2013.

[28] Valentina Lenarduzzi, Fabiano Pecorelli, Nyyti Saarimäki, Savanna Lujan, and Fabio Palomba. A Critical Comparison on Six Static Analysis Tools : Detection Agreement and Precision. *SSRN Electronic Journal*, 2022. https://www.ssrn.com/abstract=4044439.

[29] Wei Li and Sallie Henry. Object-Oriented Metrics that Predict Maintainability. *Journal of Systems and Software*, 23(2) :111–122, November 1993. http://www.sciencedirect.com/science/article/pii/016412129390077B.

[30] Rüdiger Lincke, Jonas Lundberg, and Welf Löwe. Comparing software metrics tools. In *Proceedings of the 2008 International Symposium on Software Testing and Analysis - ISSTA '08*, page 131, Seattle, WA, USA, 2008. ACM Press.

[31] Ruchika Malhotra and Kusum Lata. An empirical study on predictability of software maintainability using imbalanced data. *Software Quality Journal*, 28(4) :1581–1614, December 2020. https://doi.org/10.1007/s11219-020-09525-y.

[32] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn : Machine Learning in Python. *Journal of Machine Learning Research*, 12(85) :2825–2830, 2011. http://jmlr.org/papers/v12/pedregosa11a.html.

[33] Markus Pizka and Thomas Panas. Establishing Economic Effectiveness through Software Health-Management. May 2009. https://www.semanticscholar.org/paper/Establishing-Economic-Effectiveness-through-Pizka-Panas/744c02e1cce4aa3228ace764bb0d97029dd33303.

[34] Nicolino J. Pizzi, Arthur R. Summers, and Witold Pedrycz. Software Quality Prediction Using Median-Adjusted Class Labels. In *Proceedings of the 2002 International Joint Conference on Neural Networks*, volume 3, pages 2405–2409 vol.3, Honolulu, HI, USA, May 2002.

[35] Markus Schnappinger, Arnaud Fietzke, and Alexander Pretschner. Defining a Software Maintainability Dataset : Collecting, Aggregating and Analysing Expert Evaluations of Software Maintainability. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 278–289, Adelaide, Australia, September 2020. IEEE.

[36] Markus Schnappinger, Arnaud Fietzke, and Alexander Pretschner. A Software Maintainability Dataset. https://figshare.com/articles/dataset/A_Software_Maintainability_Dataset/12801215/3, 2020.

[37] Markus Schnappinger, Arnaud Fietzke, and Alexander Pretschner. Human-level Ordinal Maintainability Prediction Based on Static Code Metrics. In *EASE 2021 : Evaluation and Assessment in Software Engineering*, pages 160–169, Trondheim, Norway, June 2021. ACM.

[38] Markus Schnappinger, Mohd Hafeez Osman, Alexander Pretschner, and Arnaud Fietzke. Learning a classifier for prediction of maintainability based on static analysis tools. In *Proceedings of the 27th International Conference on Program Comprehension*, ICPC '19, pages 243–248, Montreal, Quebec, Canada, May 2019. IEEE Press. https://doi.org/10.1109/ICPC.2019.00043.

[39] Chikako van Koten and Andrew Gray. An application of Bayesian network for predicting object-oriented software maintainability. *Information and Software Technology*, 48(1) :59–67, January 2006. https://www.sciencedirect.com/science/article/pii/S0950584905000339.

[40] Yuming Zhou and Hareton Leung. Predicting object-oriented software maintainability using multivariate adaptive regression splines. *Journal of Systems and Software*, 80(8) :1349–1361, August 2007. https://www.sciencedirect.com/science/article/pii/S0164121206003372.

[41] Yuming Zhou, Baowen Xu, Hareton Leung, and Lin Chen. An in-depth study of the potentially confounding effect of class size in fault prediction. *ACM Transactions on Software Engineering and Methodology*, 23(1) :10 :1–10 :51, February 2014. https://doi.org/10.1145/2556777.