Performance Tuning for Automotive Software Fault Prediction

Harald Altinger*, Steffen Herbold†, Friederike Schneemann*, Jens Grabowski†, and Franz Wotawa‡
*Audi Electronics Venture GmbH, 85080 Gaimersheim, Germany
{harald.altinger,friederike.schneemann}@audi.de
†Institute of Computer Science, University of Göttingen, 37077 Göttingen, Germany,
{grabowski,herbold}@cs.uni-goettingen.de
‡Institute for Software Technology, Graz University of Technology, 8010 Graz, Austria
wotawa@ist.tugraz.at

***Abstract*—Fault prediction on high quality industry grade software often suffers from strong imbalanced class distribution due to a low bug rate. Previous work reports on low predictive performance, thus tuning parameters is required. As the State of the Art recommends sampling methods for imbalanced learning, we analyse effects when under- and oversampling the training data evaluated on seven different classification algorithms. Our results demonstrate settings to achieve higher performance values but the various classifiers are influenced in different ways. Furthermore, not all performance reports can be tuned at the same time.**

I. Introduction

In recent years, software gained importance in cars [7], as multiple features are only enabled by software. A modern day premium car as the 2016 Audi A4 might consist of up to 90 Electronic Control Unit (ECU), carry two Subscriber Identification Module (SIM) cards and 11 communication networks. Altogether a modern car might be powered by up to 1.000.000 Lines Of Code (LOC) [11]. Developing software in general is expensive, but maintaining it is even more cost intensive. In contrast to consumer electronics, the automotive industry has to consider its long Product Life Cycle (PLC) and development cycle [29]. Software maintenance costs raised from 49% in the 1970s to 75% during 1990s [23] and might climb up to 80% [8] nowadays. Fixing bugs late, i.e. after release, costs significant more money [9], [31] than during development phases. Thus, finding bugs early is of economic interest. Specific to automotive engineering, finding bugs prior to exhaustive testing phases with prototype cars, is even more important, as every bug fix patch has to traverse all prior test stages. Nowadays automotive software development uses the W-development [20] process where every specification stage has got an associated test definition stage. Multiple surveys [6], [14] report upon good knowledge and acceptance of this process by the engineers. The majority of automotive software does not use handwritten code, as model-based development techniques are more common in automotive industry and developer are used to it [6]. A recent survey [4] identifies MATLAB/Simulink as the most commonly used model-based development environment, whereas Simulink is used to design the models. dSpace TargetLink is the most common tool to generate C code. Simulation and testing might be executed using Polyspace or other dedicated tools. Automotive follows strict development guidelines *e.g.,* the MISRA-C [28] standard that reduces the available language features by *e.g.,* not allowing pointers. Similarly on model level, guidelines as MAAB [24] limit the available block sets or force treatment of every case output. These guidelines are enforced using automated conformance checks. Additionally commit guidelines enforce those checks prior a hook is accepted by the Source Control Management (SCM), conform [2].

One approach to improve software development, maintenance activities and identify faults as early as possible is Software Fault Prediction (SFP). With SFP, a model of the software under development is used to predict which instances of the software likely contain defects[1]. If sufficiently accurate, this knowledge can be used to steer the development and maintenance activities and reducing the risk of post-release bugs. While lots of foundational research was conducted on defect prediction [10], [16], to the best of our knowledge only one paper considered the application of SFP to automotive software [3]. The authors concluded that defect prediction within the automotive domain is a hard problem, despite the assumption that the restrictions due to the model-based nature of the development and the strict MISRA-C guidelines would allow for good defect prediction models. Within this paper, we want to revisit the problem with the aim to answer the following research questions for SFP in the automotive domain:

- ***RQ1:*** *Which classification models yield reliable SFP results even in case of highly imbalanced data with only few defects?*
- ***RQ2:*** *Can unsupervised resampling of the training data be used to deal with the data imbalance problem?*
- ***RQ3:*** *Which resampling rates are most suitable?*

The paper is structured as follows: Section II introduces our experimental setup, the used dataset, the classifiers to evaluate, performance characteristic to report and the resampling approach we choose. Afterwards, Section III presents the results we obtained. Then, Section IV discusses related

[1]The terms 'defects', 'bugs' and 'faults' are used as synonyms within this publication, thus they are interchangeable.

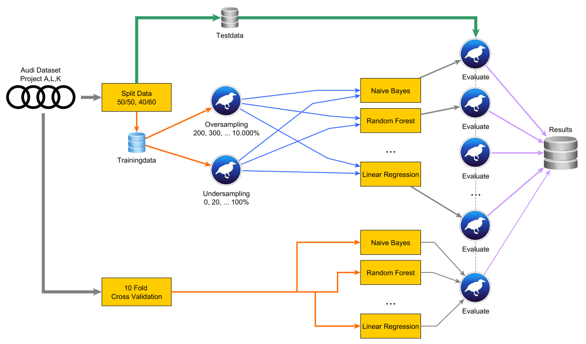

Fig. 1. experiment Setup

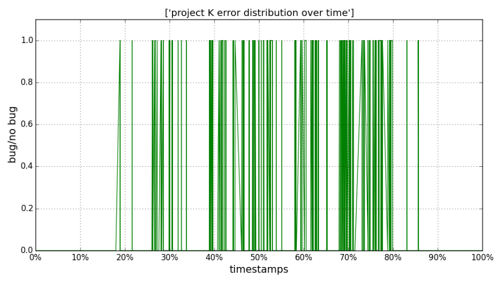

Fig. 2. bug density for project K

literature and Section V threads to the validity of our findings. Finally, Section VI concludes our work.

II. Experiment Design

Figure 1 visualizes our setup, in total 2.275 experiments were performed on all three automotive projects [2], hereafter referred to as A, K and L.

TABLE I
Parameters and their ranges used during experiments.

Parameter	min	step	max
undersampling %	0	10	100
oversampling %	200	100	900
oversampling %	1.000	500	3.500
oversampling %	4.000	1.000	10.000

A. Data

Our experiments are based on an industry grade dataset compiled of three automotive projects [3]. All of them were developed using model driven approaches utilizing Matlab Simulink and TargetLink to auto generate a MISRA-C [28] compliant source code. The sizes of the projects range between 10,000 LOC and 36,500 LOC. The projects natures are safety relevant, thus the testing effort is very high and restrictive development processes ensure high quality bug data information. For example, an author can only commit if he links the hook to an Issue Tracking System (ITS) ticket. For every feature there exists a dedicated ticket as well as for known bugs. This makes the identification of bug fixing commits reliable. The SZZ algorithm [2], [22] was used to identify the fault-inducing commits, which make up 0,7-3% percent of the commits, depending on the project. Previous work with this data set analysed the type of faults and determined a very low influence upon the gathered metrics [1].

The experiments we perform use historic data from the same project for training and prediction, defined as 'within-project' SFP. All three projects were developed over a period of three years with the first releases after one and a half year. Therefore, we decided to use the data prior this first release as training and all data from the subsequent development as test data. This is a realistic setting which can be used the same way for future projects. This lead to a data split of the projects A and L into 50% training and 50% test data for evaluation. For the project K, we use 40% as training and 60% as test data, due to a difference in the release strategy. Table II shows detailed information about the sample size of the training and test data, as well as the number of defects. It can be seen that by the definition of Weiss [33] all data sets suffer from class imbalance and can be considered as at least "modestly" imbalanced. Figure 2 shows the fault distribution of the project K. As can be seen, the defects are distributed over time. The same holds for projects A and L as well. We did not use cross validation, as cross validation cannot be used in practice like our training data setup, but only for estimation of the performance if one would use such a realistic setup. Moreover, cross validation may overestimate the performance of classification models, especially in terms of precision [32].

TABLE II
Total size and number of defects of training and test data.

	size		# faults		
	training	test	training	test	bug rate
Project A	596	1313	40	48	4.61%
Project K	941	1579	123	252	14.89%
Project L	1517	1376	32	44	2.63%

B. Classification Models

In order to gain insights for answering ***RQ1*** and to determine the impact of different classification models on the achievable performance, we evaluate eight different classification algorithms: Logistic Regression (LR), Näive Bayes (NB), Random Forest (RF), Support Vector Machine (SVM) with RBF Kernel, dcSVM [17], Ada Boost M1 with NB, Ada Boost M1 with RF and xgBoost [12]. All experiments were performed using the open-source suite of machine learning software WEKA [15] including all algorithms except for xgBoost and dcSVM. For these two, we utilized their implementations in R[2], which we bridged into WEKA using the rPlugin to ensure a constant workflow for the experiments. We did not perform any specific tuning of the parameters for these algorithms.

[2]Packages: xgboost and SwarmSVM

C. Resampling

Multiple studies from the literature support the hypothesis that resampling the training data by either oversampling the minor or downsampling the major class enhances classifier performance when dealing with imbalanced data [18], [21]. In this work, we use the unsupervised Resample filter of WEKA, which generates a randomly drawn set by copied samples from the original data into the re-sampled data. Using the 'upsampling' parameter, it is possible to create smaller data set (undersampling) or a larger data set (oversampling). For example, upsampling by 200% means that twice the number of samples will be drawn in comparison to the size of the original data, 0% means we do not touch the original data and 100% represents the same number of samples as original, but the distribution might be effected by resampling. Table I shows the sampling rates we use in this experiment. We train each classification model with each of the resampling rates. Through different values for the upsampling parameters, we determine the impact of different sampling strategies on the prediction results. These results will be used to answer ***RQ2***.

D. Performance Metrics

To report the performance of the different classification models achieved with multiple resampling rates, we use the F1 score and the G-Measure as suggested by literature [35], [19], [33] as we deal with imbalanced class distribution.

$$recall = pd = \frac{TP}{TP + FN} \tag{1a}$$

$$precision = \frac{TP}{TP + FP} \tag{1b}$$

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + reall} \tag{1c}$$

$$pf = \frac{FP}{TN + FP} \tag{1d}$$

$$G\,measure = 2 \cdot \frac{pd \cdot (1 - pf)}{pd + (1 - pf)} \tag{1e}$$

III. Results

Below, we present the detailed results for the K project which are representative for projects A and L as well. The complete results for all classifiers, sampling rates and projects are available online together with scripts for the replication of our results.[3]

Figures 3 shows the rate for the F1 score. Without resampling (value at resampling 0%), there is only a small difference between all classification models except dcSVM, which is significantly worse. The undersampling requires at least 40% of the sample size to remain, as the performance degrades drastically for all classifiers otherwise. For all other sampling rates, the impact depends on the classification model. Table III presents an overview denoting whether the impact is positive (+) or negative (-) or negligible (0). We define positive if any sampling setting archives a higher F1 or G score as the original dataset, negative if the Score is below. Figure 3 and 4 among others are used for this impact analysis. Undersampling only has a positive effect for rbfSVM. Oversampling shows good results for Ada Boost NB, NB, and the dcSVM. In case of massive oversampling, xgBoost also shows improvements. With regard to the overall performance, the sampling also leads to differences between the classifiers. The best single value is achieved with Ada Boost NB and a sampling rate of 700%. However, the results of Ada Boost NB jump a lot depending on the sampling rate and are, therefore, unreliable. In contrast, NB yields a very solid performance for all sampling rates $>600\%$ with nearly no changes and only a slightly worse performance than Ada Boost NB. Considering no sampling (shown as 0% within Figure 4) NB performs best and dcSVM the worst, all other classifiers deliver comparable results. The resampling impact is summarized in Table IV. Using the G-measure, the effect on NB is relatively small, and actually slightly negative especially for large resampling rates. Boosting algorithms as Ada M1 and xgBoost are effected the most unpredictable by sampling, SVM based algorithm performance seem to degenerate with high oversampling.

> ***RQ1:*** *NB yields the most stable and reliably good performance of all classification models, as measured by our performance metrics.*

Moreover, our results show that resampling can work, but it depends on the classification model.

> ***RQ2:*** *Resampling can improve classification models. However, undersampling and oversampling can have both a positive or negative effect depending on the classification model and should, therefore, be chosen carefully.*

To get further insights into how the sampling affects the classifiers, we analysed how the recall, precision, and pf, on which the F1 and the G-measure are based, behave for different sampling rates for the NB classifier. Figure 5 shows the results for different sampling rates. The figure shows a clear trend: undersampling improves the recall at cost of the precision, oversampling improves the precision at cost of the recall.

> ***RQ3:*** *For NB the sampling strategy should be chosen depending on whether one values recall (i.e., finding all faults) or precision (i.e., all findings are actually faults) higher. Undersampling improves recall, oversampling improves precision at the cost of the other.*

IV. Related Work

SFP is a quite active field of research, various early approaches [30], [5], [36] were presented using linear regression

[3]GitHub: https://github.com/sherbold/replication-kit-saner-2017
ZIP Archive: http://hdl.handle.net/21.11101/0000-0001-8177-7

TABLE III
IMPACT OF RESAMPLING ON THE F1 SCORE.

	Undersampling (40-90%)	Oversampling (200-1000%)	Oversampling (>1000%)
Ada Boost NB	-	+	+
Ada Boost RF	0	0	0
xgBoost	-	0	+
dcSVM	-	+	-
rbfSVM	+	-	-
RF	0	0	0
NB	-	+	+

TABLE IV
IMPACT OF RESAMPLING ON THE G-MEASURE.

	Undersampling (40-90%)	Oversampling (200-1000%)	Oversampling (>1000%)
Ada Boost NB	-	+	-
Ada Boost RF	0	0	0
xgBoost	-	+	+
dcSVM	-	+	0
rbfSVM	+	-	-
RF	0	0	0
NB	0	0	-

to predict bugs in software. This work specific discusses the metrics to use and prediction model to use. Later work [25], [27] uses machine learning algorithm and argues to use whatever metric is available. Within their opinion the choice of which algorithm to choose is of lower priority as most of them achieve an overall comparable performance. The majority of publications [30], [25], [27] reports upon no modification on the date during training phase. Further [34] reports on a comparable experiment using the NASA dataset [26]. Within their results the predictive performance using over and undersampling is strongly project dependant and outperformed by reducing the training set to high density fault modules.
[18] presents under- and oversampling as methods to overcome the imbalanced class problem. The authors study an artificial dataset with binary class distribution. Upon their results both strategies are valid. Another work dealing with SFP and imbalanced class problem is [13], they suggest to use undersampling. The authors evaluated their findings with the C4.5 classifier and use supervised resampling. [21] presents work on an Eclipse dataset dealing with imbalanced class distribution. They focus on reducing the number of input dimensions (software metrics) by applying feature ranking. Within their analysis the reduced set performs better than the full set. In addition they discovered sampling the data has no significant impact on the resulting performance.

Based on the definition by [33] our dataset is a mid-imbalanced two class problem with a rate of 4:100. They suggest to use no 'divide and conquer' based predictor, such as RF would be affected by data fragmentation as one of their leafs will contain rare data samples. Further they recommend to use recall and precision as performance report values[4], as

[4]or any other which is based on recall and/or precision, such as F1

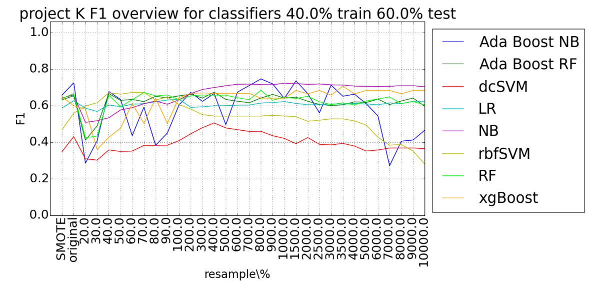

Fig. 3. classifier F1 performance overview on K using 40% train and 60% test

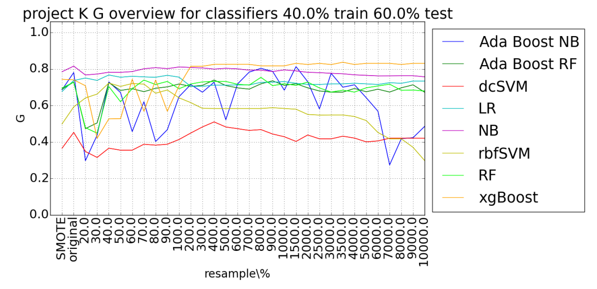

Fig. 4. classifier G-measure performance overview on K using 40% train and 60% test

they can be targeted at the minority class and will not be deformed. [19] perform tests on four datasets with binary classes suffering from imbalanced distribution. They are mainly using SVM to predict. Upon their analysis AUC seems to mask poor predictive data, whereas precision, F1 score and recall are less effected. Thus authors suggest to use F1, precision and recall to report predictive performance on imbalanced datasets.

V. THREATS TO VALIDITY

We obtained the best results using high oversampling, thus causes high computational efforts. We admit for small dataset oversampling is computationally possible which might be no option for bigger datasets. We performed no specific parameter tuning for machine learning algorithm. This will include cost sensitive evaluation as well as weighted output classes. Thus one might achieve higher performance values than we report.

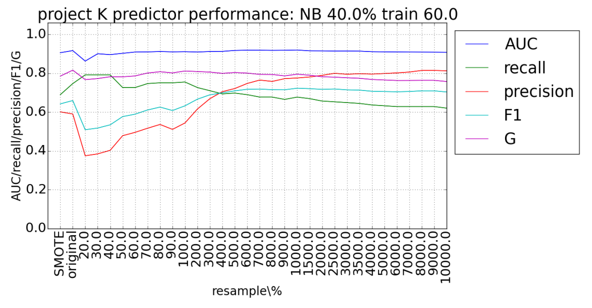

Fig. 5. Performance trend on project K using 40% training and 60% testing evaluation NB.

VI. Conclusion & Further Work

We performed SFP on a dataset with a bug rate between 2.63% and 14.89%, thus suffering from a strong imbalanced class distribution. To overcome this we applied a wide range of under- and oversampling parameters to determine the achievable performance and influences upon seven classification algorithms. The original dataset is split in test and training data prior any modification, using the train data on the sampling approach and the test data to evaluate the performance. We focus on a realistic setting keeping the data in chronological order and do not draw random samples for evaluation. To report our findings we choose F1 score and G measure along with recall and precision as literature reports the lowest impacts on them by the imbalanced class distribution. Our three research questions identify NB as the most stable predictor resulting in reliable performance results, under- and oversampling to have positive impacts on performance specific to each classifier. When using under- and oversampling one needs to carefully choose the classifier and decide whether to favour precision or recall.

References

[1] H. Altinger, Y. Dajsuren, S. Sieg, J. J. Vinju, and F. Wotawa. On Error-Class Distribution in Automotive Model-Based Software. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering*, pages 688 – 692, Osaka, Japan, Mar. 2016. IEEE.

[2] H. Altinger, S. Herbold, J. Grabowski, and F. Wotawa. Novel Insights on Cross Project Fault Prediction Applied to Automotive Software. In K. El-Fakih, G. Barlas, and N. Yevtushenko, editors, *Testing Software and Systems*, volume 9447, pages 141–157. Springer International Publishing, 2015.

[3] H. Altinger, S. Siegl, Y. Dajsuren, and F. Wotawa. A Novel Industry Grade Dataset for Fault Prediction based on Model-Driven Developed Automotive Embedded Software. In *Proceedings of the 12th Working Conference on Mining Software Repositories*, Florence, Italy, May 2015. IEEE.

[4] H. Altinger, F. Wotawa, and M. Schurius. Testing Methods Used in the Automotive Industry: Results from a Survey. In *Proceedings of the 2014 Workshop on Joining AcadeMiA and Industry Contributions to Test Automation and Model-Based Testing*, pages 1–6, San Jose, CA, July 2014. ACM.

[5] R. M. Bell, T. J. Ostrand, and E. J. Weyuker. Looking for bugs in all the right places. In *Proceedings of the 2006 international symposium on Software testing and analysis*, pages 61–72, 2006.

[6] F. Bock, D. Homm, S. Siegl, and R. German. A Taxonomy for Tools, Processes and Languages in Automotive Software Engineering. *CoRR*, abs/1601.03528, 2016.

[7] M. Broy. Challenges in automotive software engineering. In *Proceedings of the 28th international conference on Software engineering*, pages 33–42, New York, NY, USA, 2006. ACM.

[8] G. Canfora, A. Cimitile, and P. B. Lucarelli. Software maintenance. *Handbook of Software Eng. and Knowledge Eng*, pages 91–120, 2002.

[9] J. Capers. A short history of the cost per defect metric, May 2009.

[10] C. Catal and B. Diri. A systematic review of software fault prediction studies. *Expert systems with applications*, 36(4):7346–7354, 2009.

[11] R. N. Charette. This car runs on code. *IEEE Spectrum*, 46(3):3, 2009.

[12] T. Chen and C. Guestrin. XGBoost: A Scalable Tree Boosting System. *CoRR*, abs/1603.02754, 2016.

[13] C. Drummond, R. C. Holte, and others. C4. 5, class imbalance, and cost sensitivity: why under-sampling beats over-sampling. In *Workshop on learning from imbalanced datasets II*, volume 11. Citeseer, 2003.

[14] P. Haberl, A. Spillner, K. Vosseberg, and M. Winter. Survey 2011: Software Test in Practice. Technical report, German testing board, 2011.

[15] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[16] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell. A Systematic Literature Review on Fault Prediction Performance in Software Engineering. *IEEE Trans. Softw. Eng.*, 38(6):1276–1304, Nov. 2012.

[17] C.-J. Hsieh, S. Si, and I. S. Dhillon. A Divide-and-Conquer Solver for Kernel Support Vector Machines. In *Proceedings of the 31st International Conference on Machine Learning*, volume 32, Beijing, China, 2014. JMLR: W&CP.

[18] N. Japkowicz. Learning from imbalanced data sets: a comparison of various strategies. In *AAAI workshop on learning from imbalanced data sets*, volume 68, pages 10–15. Menlo Park, CA, 2000.

[19] L. A. Jeni, J. F. Cohn, and F. D. L. Torre. Facing Imbalanced DataRecommendations for the Use of Performance Metrics. In *Affective Computing and Intelligent Interaction (ACII), 2013 Humaine Association Conference on*, pages 245–251, Sept. 2013.

[20] L. Jin-Hua, L. Qiong, and L. Jing. The w-model for testing software product lines. In *Computer Science and Computational Technology, 2008. ISCSCT'08. International Symposium on*, volume 1, pages 690–693, 2008.

[21] T. M. Khoshgoftaar, K. Gao, and N. Seliya. Attribute Selection and Imbalanced Data: Problems in Software Defect Prediction. In *2010 22nd IEEE International Conference on Tools with Artificial Intelligence*, volume 1, pages 137–144, Oct. 2010.

[22] S. Kim, T. Zimmermann, K. Pan, and E. J. Whitehead. Automatic identification of bug-introducing changes. In *Automated Software Engineering, 2006. ASE'06. 21st IEEE/ACM International Conference on*, pages 81–90, Tokyo, Japan, 2006. IEEE Computer Society.

[23] D. Kozlov, J. Koskinen, and M. Sakkinen. Fault-Proneness of Open Source Software: Exploring its Relations to Internal Software Quality and Maintenance Process. *Open Software Engineering Journal*, 7:1–23, 2013.

[24] T. Mathworks. Mathworks Automotive Advisory Board Checks (MAAB), 2014.

[25] T. Menzies, J. Greenwald, and A. Frank. Data mining static code attributes to learn defect predictors. *Software Engineering, IEEE Transactions on*, 33(1):2–13, 2007.

[26] T. Menzies, R. Krishna, and D. Pryor. *The Promise Repository of Empirical Software Engineering Data*. North Carolina State University, 2015.

[27] T. Menzies, Z. Milton, B. Turhan, B. Cukic, Y. Jiang, and A. Bener. Defect prediction from static code features: current results, limitations, new approaches. *Automated Software Engineering*, 17(4):375–407, 2010.

[28] Motor Industry Software Reliability Association. *MISRA-C:2004 - Guidelines for the use of the C language in critical systems*. MISRA, Warwickshire, 2 edition, 2004.

[29] S.-O. Mueller, M. Brand, S. Wachendorf, H. Schroeder, T. Szot, S. Schwab, and B. Kremer. Integration vernetzter Fahrerassistenz-Funktionen mit HiL fuer den VW Passat CC. *ATZextra*, 14(4):60–65, 2009.

[30] T. J. Ostrand and E. J. Weyuker. The distribution of faults in a large industrial software system. In *ACM SIGSOFT Software Engineering Notes*, volume 27, pages 55–64, 2002.

[31] F. Shull, V. Basili, B. Boehm, P. Costa, M. Lindvall, D. Port, I. Rus, R. Tesoriero, M. Zelkowitz, and others. What we have learned about fighting defects. In *Software Metrics, 2002. Proceedings. Eighth IEEE Symposium on*, pages 249–258, 2002.

[32] M. Tan, L. Tan, S. Dara, and C. Mayeux. Online Defect Prediction for Imbalanced Data. In *Proceedings of the 37th International Conference on Software Engineering - Volume 2*, ICSE '15, pages 99–108, Piscataway, NJ, USA, 2015. IEEE Press.

[33] G. M. Weiss. Foundations of Imbalanced Learning. In *Imbalanced Learning*, pages 13–41. John Wiley & Sons, Inc., 2013.

[34] H. Zhang, A. Nelson, and T. Menzies. On the Value of Learning from Defect Dense Components for Software Defect Prediction. In *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, PROMISE '10, pages 14:1–14:9, New York, NY, USA, 2010. ACM.

[35] H. Zhang and X. Zhang. Comments on "Data Mining Static Code Attributes to Learn Defect Predictors". *IEEE Transactions on Software Engineering*, 33(9):635–637, Sept. 2007.

[36] T. Zimmermann, R. Premraj, and A. Zeller. Predicting defects for eclipse. In *Predictor Models in Software Engineering, 2007. PROMISE'07: ICSE Workshops 2007. International Workshop on*, pages 9–9. IEEE, 2007.