Boosting Spectrum-Based Fault Localization for Spreadsheets with Product Metrics in a Learning Approach

Adil Mukhtar
TU Graz
Graz, Austria
amukhtar@ist.tugraz.at

Birgit Hofer
TU Graz
Graz, Austria
bhofer@ist.tugraz.at

Dietmar Jannach
University of Klagenfurt
Klagenfurt, Austria
dietmar.jannach@aau.at

Konstantin Schekotihin
University of Klagenfurt
Klagenfurt, Austria
konstantin.schekotihin@aau.at

Franz Wotawa
TU Graz
Graz, Austria
wotawa@ist.tugraz.at

ABSTRACT

Faults in spreadsheets are not uncommon and they can have significant negative consequences in practice. Various approaches for fault localization were proposed in recent years, among them techniques that transferred ideas from spectrum-based fault localization (SFL) to the spreadsheet domain. Applying SFL to spreadsheets proved to be effective, but has certain limitations. Specifically, the constrained computational structures of spreadsheets may lead to large sets of cells that have the same assumed fault probability according to SFL and thus have to be inspected manually. In this work, we propose to combine SFL with a fault prediction method based on spreadsheet metrics in a machine learning (ML) approach. In particular, we train supervised ML models using two orthogonal types of features: (i) variables that are used to compute similarity coefficients in SFL and (ii) spreadsheet metrics that have shown to be good predictors for faulty formulas in previous work. Experiments with a widely-used corpus of faulty spreadsheets indicate that the combined model helps to significantly improve fault localization performance in terms of wasted effort and accuracy.

CCS CONCEPTS

• **Software and its engineering → Software testing and debugging**; *Software maintenance tools.*

KEYWORDS

Spreadsheets, Spectrum-based Fault Localization, Artificial Intelligence, Machine Learning

ACM Reference Format:
Adil Mukhtar, Birgit Hofer, Dietmar Jannach, Konstantin Schekotihin, and Franz Wotawa. 2022. Boosting Spectrum-Based Fault Localization for Spreadsheets with Product Metrics in a Learning Approach. In *37th IEEE/ACM International Conference on Automated Software Engineering (ASE '22), October 10–14, 2022, Rochester, MI, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3551349.3559546

1 INTRODUCTION

Spreadsheets are widely used in organizations for reporting and decision making tasks. Faults in spreadsheets are, however, not uncommon [22] and can represent a major risk to organizations. A variety of approaches have been proposed over the years to avoid, locate, and fix faults in spreadsheets [11]. In the context of debugging, adapting Spectrum-based Fault Localization (SFL) techniques [2, 26] for the spreadsheet domain proved to be effective [7, 9]. However, applying SFL to rank the elements of a spreadsheet with respect to their assumed fault probability may easily lead to having many elements with the same ranking, i.e., to large ties [7]. If the tie that contains the faulty cell(s)—the *critical* tie—is large, many elements must be inspected by the spreadsheet developer, ultimately leading to more *wasted effort* [3]. The problem of large critical ties when applying SFL techniques is particularly pronounced for spreadsheets, which have more limited computational structures than conventional programs, e.g., there are no loops.

In this work, we propose to combine SFL with orthogonal information for improved fault prediction in spreadsheets. Previous work has shown that *product metrics* (i.e. metrics that can be computed directly from the spreadsheet), including spreadsheet smells [8], can be highly effective predictors for faulty formulas in spreadsheets [5, 15]. Technically, we rely on a supervised machine learning approach where the features to predict if a given formula in a spreadsheet is faulty include (i) the statistics used for computing suspiciousness values in SFL and (ii) a set of product metrics [15]. The two types of features leverage different types of information. Whereas the SFL-based features are based on spreadsheet executions, the product metrics are statically derived from the structure of the spreadsheets and their formulas.

To validate our approach, we conducted experiments with the widely used EUSES spreadsheet corpus [10]. The experiments show that the combined learning approach is highly effective, leading to a substantial increase in prediction accuracy and, likewise, largely reduced amounts of wasted effort for fault localization.

A further analysis of the rankings produced by the traditional SFL procedure revealed that computing suspiciousness values, e.g., with the Ochiai coefficient, indeed often results in large critical ties. Our learning technique is successful to break these ties and thus leads to a finer-grained ranking. As a side result, we found that some effort-based metrics from the literature can be non-informative or misleading in case of large ties in the ranking.

2 BACKGROUND & PREVIOUS WORK

Many techniques have been proposed over the years for the problem of (automated) fault localization in software [26]. Among them, SFL represents a modern statistical approach. Basically, it assumes that statements that are executed in many failing test cases and only in a few passing test cases are more likely to be responsible for failures [13]. Similarity coefficients are central in SFL-based approaches. They are used to compute the suspiciousness of individual program statements based on *four* execution statistics, i.e., on pass and fail statistics of executed and non-executed statements, see [20].

To use SFL for spreadsheets, some adaptations are necessary. Since there are no execution traces available in spreadsheets, information about which cells directly or indirectly influence the value of the correct and erroneous output cells is used instead [10]. Hofer *et al.* [9] empirically evaluated the performance of various similarity coefficients for spreadsheets and found that using the Ochiai index leads to good results. Therefore, we use SFL with the Ochiai index as a baseline method. Moreover, instead of combining the four input statistics in a pre-defined way (as done in the Ochiai index), we try to learn how to combine the inputs—and thus to learn the measure—from the data. An orthogonal approach to find more effective coefficients would be to learn how to combine multiple similarity coefficients as done in the Multric approach [29].

SFL, like other fault localization techniques, may suffer from the problem of *ties*, where several program statements receive the same suspiciousness score and, therefore, the same ranking [28]. As there is no unique ranking of the statements, researchers have proposed to report the *average*, *best*, and *worst* case of the metrics (e.g., the *wasted effort*) to simulate the scenarios where the user inspects the actual fault first, in the middle or last from a set of statements with the same suspicious score.

In our work, we combine SFL with product metrics for improved fault prediction (and tie breaking) in a machine learning approach. Other researchers [14, 19] have combined spectrum-based metrics with a variety of static (i.e., file, function, and statement metrics) and dynamic metrics (i.e., spectrum-based and mutation-based metrics). Differently from these works, we rely on product metrics as predictors and specifically focus on the domain of spreadsheets.

In the context of spreadsheet debugging, various heuristic, reasoning, or learning-based methods were put forward in the last years [11, 23], and prominent examples include Melford [25], CUSTODES [5], Warder [17], SGUARD [18], ExceLint [4], CACheck [6], EmptyCheck [27], and SmellChecker [15]. These methods are mainly designed for the problem of fault *detection* and fault *prediction*, whereas SFL (and our present work) focuses on fault *localization* and the root cause of an observed misbehavior. SFL therefore requires information about the correctness of outputs given the inputs. In our work, we combine such information with information that can be statically derived from spreadsheets in the form of product metrics. One effect of combining SFL with product metrics is that it leads to smaller ties, and thus improved accuracy on average. Alternative techniques for tie-breaking for spreadsheets scenarios were proposed earlier by Getzner *et al.* [7], but these techniques do not rely on additional information from metrics for tie-breaking.

3 METHOD

We use a *supervised learning* approach, where the features (predictor variables) include (i) execution statistics from test cases and (ii) product metrics derived from the spreadsheet. Each data point (row) corresponds to the outcomes of test case execution, and the *label* of the target variable is therefore either *correct* or *faulty*. Figure 1 illustrates the general setup. The goal is to train a machine learning model from the data to predict the correctness or faultiness of a specific formula. Like various previous works, we focus on the localization of *faulty formulas* and not on faulty input values.

SFL-based Features			*Product Metrics*			
$Feature_1$	…	$Feature_m$	$Feature_{m+1}$	…	$Feature_r$	*Label*
$value_{1,1}$	…	$value_{1,m}$	$value_{1,m+1}$	…	$value_{1,r}$	correct
…	…	…	…	…	…	…
$value_{n,1}$	…	$value_{n,m}$	$value_{n,m+1}$	…	$value_{n,r}$	faulty

Figure 1: General structure of the learning problem

Overall, our approach is generic and extensible in that the set of features—both the SFL-based ones and the product metrics—can be chosen depending on the specific situation. Moreover, our method is not tied to a particular machine learning approach, and any supervised learning method can in principle be applied.

SFL-based Features. When applying SFL, we compute four statistics for each *formula cell*. These four statistics capture how often a formula cell was involved in the computation of a correct or erroneous output in a test case, and how often it was not involved.[1] They are commonly used to compute similarity coefficients that determine the suspiciousness of a formula, as mentioned above.

Differently from previous work, we propose not to rely on a static, predefined way to combine these statistics, but to *learn* to combine the statistics from the data. The four statistics are therefore part of the SFL-based features in our learning setup illustrated in Figure 1. Since our learning approach is generic, it allows us to easily add additional SFL-based features. In our experiments, we engineered a fifth feature. This feature is computed as the ratio of the number of involvements of a cell in an erroneous output (i.e., one of the four features from above) to the number of incorrect output cells in a spreadsheet. This feature thus estimates the overall involvement of a cell in the computation of erroneous output. The inclusion of additional derived features is part of our ongoing work.

Product Metrics. Koch *et al.* [15] proposed a catalog of 64 product metrics and demonstrated the predictive power of these metrics for fault localization in spreadsheets. These metrics can be statically derived from a given spreadsheet, and we therefore do not expect that a predictor based on product metrics *alone* will on average be more accurate than SFL-based methods, which rely on information about passed and failed test cases. However, our hypothesis is that the product metrics represent important information that is orthogonal to what is used in SFL, and that combining SFL features with metric-based features is synergistic, i.e., it will result in more accurate rankings in terms of fault suspiciousness.

[1] A formal definition of these metrics can be found in [9].

To avoid the inclusion of too many features in our prediction model and to balance the contribution of the SFL-based and metrics-based features, we ran a feature importance analysis for the product metrics on the data set used in this study (see later). Based on this analysis, we retained the *five* most important metrics-based features from [15]: (i) the number of range references referring to the cell; (ii) the number of direct references; (iii) the total number references to the cell, i.e., the sum of the two previous metrics; (iv) *"standard deviation (column)"*, i.e., the absolute difference to the mean of the numeric values in the same column; (v) *"standard deviation (row)"*. To foster future research in the area, we publicly share the dataset with all derived features (both SFL-based and metrics-based ones) for the corpus of faulty spreadsheets used in our experiments.[2]

Learning Method. Our final model for learning has 10 features. We experimented with an artificial neural network (ANN) and with a Random Forest (RF) model. Systematic hyperparameter tuning was applied using ten-fold cross-validation and with the F1-measure as optimization goal. Our optimized ANN has 10 input nodes, one hidden layer with 80 nodes, and a softmax output layer that represents the probability distribution of a formula being faulty.

When analyzing the available data from EUSES corpus, we found that it contains many duplicate entries. These duplicate rows are mainly the result of the limited computational structures in spreadsheets and a larger number of copy-equivalent formulas in the spreadsheets. To avoid that such duplicates distort the learning process, we applied deduplication on the training data, i.e., we only kept one representative of a set of duplicates. The training data is reduced by about 90 % through the deduplication process, thereby also making training more efficient. Moreover, the output classes in our problem settings are highly imbalanced, as most data points correspond to succeeding test cases. Therefore, before training the models, we applied oversampling of the minority class with SMOTE [16] until the classes in the training data were balanced.

4 EXPERIMENTAL EVALUATION

In this section, we first describe the experimental setup in Section 4.1 and then present our findings in Section 4.2.

4.1 Experiment Setup

We used the modified EUSES corpus [10] for our experiments. The corpus comprises 696 spreadsheets, each of them containing one single fault, which was injected based on mutation operators presented in [1]. The average number of cells per spreadsheet is 1,302 (14 to 41,097). Overall, there are 218,320 rows in the data. To evaluate our prediction models, we split the corpus into training (90 %) and test sets (10 %) within a ten fold cross-validation procedure. The training data was deduplicated and over-sampled for class balance as described in the previous section.

After learning the model on the training data, we used it to rank the cells with formulas in each spreadsheet according to the fault scores predicted by the model. We use two types of evaluation measures in our experiments:

- *Hit Ratio@n*: This metric indicates how often our model ranked the faulty cell within the first *n* elements.
- *Wasted Effort*: This metric, sometimes called EXAM score [12], captures how many non-faulty cells were ranked before (*or*: have to be inspected before) the faulty cell. The *wasted effort* is thus an absolute evaluation measure and not a percentage rank. Such measures are increasingly considered favorable to assess software fault localization techniques [3].

Table 1: Compared Approaches

SFL-Ochiai	"Standard" SFL with Ochiai coefficient [10].
ANN-SFL	ANN model using only 4 SFL-based features.
ANN-SFL+	ANN-SFL plus 5th engineered feature.
ANN-Metrics	ANN model based on five product metrics.
ANN-Combined	ANN model with all ten features.
RF-SFL	RF model using only 4 SFL-based features.
RF-SFL+	RF-SFL plus 5th engineered feature.
RF-Metrics	RF model based on five product metrics.
RF-Combined	RF model with all ten features.

As done, e.g., in [10] and as discussed above, we report the *average*, *best*, and *worst* results for both types of metrics. To understand the contributions of different types of features in our combined model, we created submodels that only use certain parts of the available information. Table 1 summarizes the approaches and machine learning (ML) models that we compared in our experiments.

4.2 Results

The main results of our experiments are shown in Table 2. The columns marked in gray show the *average* results for the ranking-related measures (Hit Ratio, Wasted Effort) for all examined methods. Both combined models (ANN-Combined, RF-Combined), which rely on both types of features, clearly outperform the best-performing SFL-based baseline SFL-Ochiai from [10]. The Hit Ratios increase by 30-50 % and the Wasted Effort is reduced by about 25 %. Interestingly, none of the two combined models consistently outperforms the other on these measures. The ANN-Combined method is best in terms of *Wasted Effort*, and the RF-Combined model succeeds in terms of the Hit Ratio. Similar effects are found for the *worst-case* results. The differences between the best model RF-Combined in terms of the Hit Ratio and the baseline SFL-Ochiai method are statistically significant (according to an ANOVA and t-tests with $\alpha = 0.05$). For the *Wasted Effort* the differences between baseline and best model (ANN-Combined) appear solid (about 25 %), but the variance is high for this absolute measure, which can take very large values for some of the huge spreadsheets in the corpus. As a result, statistical significance is not achieved (p=0.33). Removing outliers that fall behind three standard deviations from the mean would make the results significant ($p<0.01$).

The average wasted effort seems to be too high to be useful in proactive. However, it can be explained by the outliers. For this reason, we additionally indicate the median of the average case. The medians of the average wasted effort range between 2.87 and 8.37, meaning that for half of the spreadsheets less than 9 cells have to be manually inspected until the faulty cells are found. In the case of the *best-case* metrics, we find that the SFL-Ochiai and the RF-SFL (and RF-SFL+) methods lead to better results than the combined

[2]Code, data and hyperparameters: https://doi.org/10.5281/zenodo.6826795

Table 2: Hit Ratios, mean Average, Best, Worst Wasted Effort and mean size (and std. deviation) of critical ties. The best results across all methods are printed in bold face.

	HitRatio@1			HitRatio@5			Wasted Effort			Size of
	Avg.	Best	Worst	Avg.	Best	Worst	Avg. (Std., Median)	Best (Std.)	Worst (Std.)	critical ties (Std.)
SFL-Ochiai	0.18	0.92	0.11	0.48	0.96	0.31	40.85 (20.39, 4.87)	1.90 (1.05)	79.79 (40.61)	77.93 (421.91)
ANN-SFL	0.18	0.90	0.11	0.49	0.95	0.31	40.13 (19.97, 4.5)	2.48 (1.52)	77.78 (39.24)	75.33 (406.12)
ANN-SFL+	0.18	0.91	0.11	0.49	0.96	0.30	37.06 (22.08, 4.75)	2.37 (3.05)	71.74 (41.82)	69.46 (369.78)
ANN-Metrics	0.14	0.68	0.13	0.39	0.81	0.32	83.00 (38.61, 9.37)	26.10 (22.61)	139.90 (67.30)	113.71 (487.34)
ANN-Combined	0.28	0.68	0.26	0.61	0.87	**0.52**	**31.32** (19.96, 3.12)	18.42 (21.92)	**44.22** (20.63)	**25.83** (103.38)
RF-SFL	0.17	**0.94**	0.11	0.49	**0.98**	0.31	40.39 (19.95, 4.62)	**1.63** (3.36)	79.16 (40.18)	77.57 (421.62)
RF-SFL+	0.17	**0.94**	0.11	0.49	**0.98**	0.30	40.55 (19.65, 4.62)	1.78 (3.35)	79.32 (39.88)	77.55 (421.61)
RF-Metrics	0.19	0.73	0.18	0.42	0.84	0.34	96.37 (36.13, 8.37)	38.63 (24.79)	154.11 (63.34)	115.38 (488.37)
RF-Combined	**0.33**	0.77	**0.30**	**0.62**	0.90	0.51	38.93 (25.09, **2.87**)	24.88 (27.46)	52.98 (24.67)	28.12 (110.98)

methods that rely on more features. The explanation of this effect lies in the average size of the critical ties, also shown in Table 2, where SFL-Ochiai and RF-SFL lead to the largest critical ties on average among the SFL-based techniques. This means that these methods have the strongest tendency to assign identical scores to a larger number of cells. Now if the true fault is among such a large critical tie, all the cells in this tie will be ranked before all other cells, and in the *best-case* measurement, it is assumed that the true fault is at position one. Larger ties may therefore directly lead to better results in the *best* case. This observation leads us to question the value of reporting the *best* case in general. Imagine a ranking method that assigns the same score to all cells of a spreadsheet (or all statements in a regular program). In this case, the true fault will be assumed to be at the first position in the ranking across all cells, and thus necessarily lead to the highest recall possible and the lowest wasted effort.

Looking more closely at the rankings of the other compared methods or submodels, we observe the following:

- ANN-SFL+ performs slightly better than ANN-SFL, but there is almost no difference between RF-SFL and RF-SFL+. This suggests that the ANN model is better able to leverage the information contained in the fifth engineered SFL feature that we proposed in Section 3.
- The models that solely rely on product metrics (RF-Metrics, ANN-Metrics) do not perform well compared to the SFL-based models. As described, this can be attributed to the fact that product metrics only rely on a static analysis of the spreadsheets and cannot leverage information from test cases like SFL. However, the performance of the combined models (ANN-Combined, RF-Combined) indicates that the product metrics are helpful to break critical ties, and to thereby increase the average results significantly.
- The combined models (ANN-Combined and RF-Combined) lead to a significant reduction of the average size of the critical ties ($p<0.01$), and they thus likely contributed to the observed gains in ranking-based measures.
- Additional experiments show that leaving out the deduplication and oversampling steps consistently leads to results that are worse in terms of *Hit Ratio* and *Wasted Effort* compared to the combined models. Deduplication in the training data generally seems to be helpful to reduce the *Wasted Effort* but lead to drops in *Hit Ratio*. More experiments are still needed to better understand the interplay of deduplication and oversampling and their effects on ranking-based measures.

5 FUTURE PLANS

To our knowledge, our work is the first to explore the value of combining SFL with product metrics for spreadsheet programs. More work is thus required to validate the generalizability of the findings reported in this paper. In particular, it is important to expand the scope of the experiments to additional spreadsheet corpora, e.g., ENRON [24] or INFO1 [7]. In order to perform such experiments, it is, however, first required to create a suite of test cases for each corpus to derive the SFL execution statistics.

We will further improve our machine learning models. Currently, no clear winner was found across the metrics. Therefore, we will in particular explore the value of using more complex deep learning architectures in the future, which have the promise to be able to better model non-linear dependencies in the data. Moreover, we will investigate to what extent other types of static and dynamic features that were successfully explored in related works (e.g., mutation-related ones) are suited to further increase prediction accuracy.

Furthermore, we will develop a plugin similar to the SmellChecker plugin [21] to perform user studies. One last future strand of our research will be directed towards the question if the proposed method is also effective for programs written in functional, procedural or object-oriented programming languages.

6 CONCLUSION

We propose a novel approach to significantly improve the performance of SFL-based methods for spreadsheets, which combines SFL-based features regarding execution statistics with product metrics that can be derived through static analyses. Overall, we see our work as another step towards improved fault localization support for spreadsheets, which often serve as a basis even for critical decision-making processes in organizations.

ACKNOWLEDGMENTS

Work funded by the Austrian Science Fund (FWF) under contract number P 32445 (Interactive Spreadsheet Debugging).

REFERENCES

[1] R. Abraham and M. Erwig. 2009. Mutation Operators for Spreadsheets. *IEEE Transactions on Software Engineering* 35, 01 (2009), 94–108. https://doi.org/10.1109/TSE.2008.73

[2] Rui Abreu, Peter Zoeteweij, and Arjan J. C. van Gemund. 2007. On the Accuracy of Spectrum-Based Fault Localization. In *Proceedings of the Testing: Academic and Industrial Conference Practice and Research Techniques, TAICPART-MUTATION '07*. 89–98.

[3] Aaron Ang, Alexandre Perez, Arie Van Deursen, and Rui Abreu. 2017. Revisiting the practical use of automated software fault localization techniques. In *IEEE 28th International Symposium on Software Reliability Engineering Workshops (ISSREW 2017)*. 175–182. https://doi.org/10.1109/ISSREW.2017.68

[4] Daniel W. Barowy, Emery D. Berger, and Benjamin G. Zorn. 2018. ExceLint: automatically finding spreadsheet formula errors. *Proceedings of the ACM on Programming Languages* 2 (2018), 148:1–148:26.

[5] Shing-Chi Cheung, Wanjun Chen, Yepang Liu, and Chang Xu. 2016. CUSTODES: Automatic Spreadsheet Cell Clustering and Smell Detection Using Strong and Weak Features. In *Proceedings of the International Conference on Software Engineering (ICSE)*. 464–475.

[6] Wensheng Dou, Chang Xu, S. C. Cheung, and Jun Wei. 2017. CACheck: Detecting and Repairing Cell Arrays in Spreadsheets. *IEEE Transactions on Software Engineering* 43, 3 (2017), 226–251.

[7] Elisabeth Getzner, Birgit Hofer, and Franz Wotawa. 2017. Improving Spectrum-Based Fault Localization for Spreadsheet Debugging. In *2017 IEEE International Conference on Software Quality, Reliability and Security (QRS)*. 102–113. https://doi.org/10.1109/QRS.2017.21

[8] Felienne Hermans, Martin Pinzger, and Arie van Deursen. 2015. Detecting and refactoring code smells in spreadsheet formulas. *Empirical Software Engineering* 20, 2 (2015), 549–575.

[9] Birgit Hofer, Alexandre Perez, Rui Abreu, and Franz Wotawa. 2015. On the empirical evaluation of similarity coefficients for spreadsheets fault localization. *Automated Software Engineering* 22, 1 (2015), 47–74. https://doi.org/10.1007/s10515-014-0145-3

[10] Birgit Hofer, André Riboira, Franz Wotawa, Rui Abreu, and Elisabeth Getzner. 2013. On the Empirical Evaluation of Fault Localization Techniques for Spreadsheets. In *Proceedings of the 16th International Conference on Fundamental Approaches to Software Engineering (FASE) (Lecture Notes in Computer Science, Vol. 7793)*. Springer, 68–82. https://doi.org/10.1007/978-3-642-37057-1_6

[11] Dietmar Jannach, Thomas Schmitz, Birgit Hofer, and Franz Wotawa. 2014. Avoiding, finding and fixing spreadsheet errors - A survey of automated approaches for spreadsheet QA. *Journal of Systems and Software* 94 (2014), 129–150. https://doi.org/10.1016/j.jss.2014.03.058

[12] Jiajun Jiang, Ran Wang, Yingfei Xiong, Xiangping Chen, and Lu Zhang. 2019. Combining Spectrum-Based Fault Localization and Statistical Debugging: An Empirical Study. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 502–514. https://doi.org/10.1109/ASE.2019.00054

[13] James A. Jones, Mary Jean Harrold, and John T. Stasko. 2002. Visualization of test information to assist fault localization. In *Proceedings of the 22rd International Conference on Software Engineering*, Will Tracz, Michal Young, and Jeff Magee (Eds.). 467–477. https://doi.org/10.1145/581339.581397

[14] Yunho Kim, Seokhyeon Mun, Shin Yoo, and Moonzoo Kim. 2019. Precise Learn-to-Rank Fault Localization Using Dynamic and Static Features of Target Programs. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 28, 4 (oct 2019), 1–34. https://doi.org/10.1145/3345628

[15] P. Koch, K. Schekotihin, D. Jannach, B. Hofer, and F. Wotawa. 2019. Metric-based Fault Prediction for Spreadsheets. *IEEE Transactions on Software Engineering* (2019), forthcoming. https://doi.org/10.1109/TSE.2019.2944604

[16] Guillaume Lemaître, Fernando Nogueira, and Christos K. Aridas. 2017. Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning. *Journal of Machine Learning Research* 18, 17 (2017), 1–5.

[17] Da Li, Huiyan Wang, Chang Xu, Fengmin Shi, Xiaoxing Ma, and Jian Lu. 2019. WARDER: Refining Cell Clustering for Effective Spreadsheet Defect Detection via Validity Properties. In *19th International Conference on Software Quality, Reliability and Security*. 139–150.

[18] Da Li, Huiyan Wang, Chang Xu, Ruiqing Zhang, Shing-Chi Cheung, and Xiaoxing Ma. 2019. SGUARD: A Feature-Based Clustering Tool for Effective Spreadsheet Defect Detection. In *34th IEEE/ACM International Conference on Automated Software Engineering*. 1142–1145.

[19] Xia Li, Wei Li SUSTech, Yuqun Zhang, Lingming Zhang, and Wei Li. 2019. DeepFL: Integrating Multiple Fault Diagnosis Dimensions for Deep Fault Localization. In *28th ACM SIGSOFT International Symposium on Software Testing and Analysis*. ACM, 169–180. https://doi.org/10.1145/3293882

[20] Lucia, David Lo, Lingxiao Jiang, Ferdian Thung, and Aditya Budi. 2014. Extended comprehensive study of association measures for fault localization. *Journal of Software: Evolution and Process* 26, 2 (2014), 172–219. https://doi.org/10.1002/smr.1616

[21] Adil Mukhtar, Birgit Hofer, Dietmar Jannach, and Franz Wotawa. 2022. Spreadsheet debugging: The perils of tool over-reliance. *Journal of Systems and Software* 184 (2022). https://doi.org/10.1016/j.jss.2021.111119

[22] Raymond R. Panko. 2008. Thinking is Bad: Implications of Human Error Research for Spreadsheet Research and Practice. *CoRR* abs/0801.3114 (2008). arXiv:0801.3114

[23] Konstantin Schekotihin, Birgit Hofer, Franz Wotawa, and Dietmar Jannach. 2021. AI-based Spreadsheet Debugging. In *Artificial Intelligence Methods for Software Engineering*. World Scientific, 371–399. https://doi.org/10.1142/9789811239922_0013

[24] Thomas Schmitz and Dietmar Jannach. 2016. Finding errors in the Enron spreadsheet corpus. In *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. Cambridge, UK. https://doi.org/10.1109/vlhcc.2016.7739679

[25] Rishabh Singh, Benjamin Livshits, and Benjamin Zorn. 2017. *Melford: Using Neural Networks to Find Spreadsheet Errors.* Technical Report Microsoft Technical Report MSR-TR-2017-5. Microsoft.

[26] W. Eric Wong, Ruizhi Gao, Yihao Li, Rui Abreu, and Franz Wotawa. 2016. A Survey on Software Fault Localization. *IEEE Transactions on Software Engineering* 42, 8 (2016), 707–740. https://doi.org/10.1109/TSE.2016.2521368

[27] Liang Xu, Shuo Wang, Wensheng Dou, Bo Yang, Chushu Gao, Jun Wei, and Tao Huang. 2018. Detecting faulty empty cells in spreadsheets. In *IEEE International Conference on Software Analysis, Evolution and Reengineering*. 423–433.

[28] Xiaofeng Xu, Vidroha Debroy, W. Eric Wong, and Donghui Guo. 2011. Ties within Fault Localization Rankings: Exposing and Addressing the Problem. *International Journal of Software Engineering and Knowledge Engineering* 21, 6 (2011), 803–827. https://doi.org/10.1142/S0218194011005505

[29] Jifeng Xuan and Martin Monperrus. 2014. Learning to combine multiple ranking metrics for fault localization. In *30th International Conference on Software Maintenance and Evolution (ICSME 2014)*. 191–200. https://doi.org/10.1109/ICSME.2014.41