Preprint: Krzysztof Baciejowski, Damian Garbala, Szymon Zmijewski, Lech Madeyski (2023). Are Code Review Smells and Metrics Useful in Pull Request-Level Software Defect Prediction?. In: Kryvinska, N., Greguš, M., Fedushko, S. (eds) Developments in Information and Knowledge Management Systems for Business Applications. Studies in Systems, Decision and Control, vol 466. Springer, Cham. DOI: 10.1007/978-3-031-27506-7_2 Preprint: https://madeyski.e-informatyka.pl/download/BaciejowskiEtAl23.pdf

Are code review smells and metrics useful in pull request-level software defect prediction?

Krzysztof Baciejowski, Damian Garbala, Szymon Żmijewski and Lech Madeyski

Abstract The process of software code review is a well-established practice in software engineering. Previous research identified quality metrics for code review. However, to our knowledge, this paper is the first that uses those review smells and metrics as predictors in software defect prediction. We used review process metrics used in other studies as well as created new ones. A machine learning model is fed with various process metrics (code review) and product metrics (software code) to be able to predict if a pull request might introduce a defect. For the GitHub repositories examined, the mean absolute errors for predictive models were equal to 0.26 (for the model built on product metrics only), 0.29 (for model built on review metrics only), and 0.25 (for model built on combined metrics). The results indicate that the quality of the code review conveys additional valuable information that can be utilized to better predict software defects. In fact, review metrics alone appeared to be almost as good predictors of software defects as investigated since a long time and widely used software product metrics.

Krzysztof Baciejowski(✉)
Wroclaw University of Science and Technology, Poland, e-mail: 246785@student.pwr.edu.pl ORCID: 0000-0001-9572-1625

Damian Garbala
Wroclaw University of Science and Technology, Poland, e-mail: 247025@student.pwr.edu.pl

Szymon Żmijewski
Wroclaw University of Science and Technology, Poland, e-mail: 246660@student.pwr.edu.pl

Lech Madeyski
Wroclaw University of Science and Technology, Department of Applied Informatics, Poland e-mail: lech.madeyski@pwr.edu.pl, ORCID: 0000-0003-3907-3357

1 Introduction

The software development process is bound to encounter unexpected defects in the code base. The frequency of introducing these defects should be reduced by the code review process. In fact, some studies have shown that greater coverage for code review and greater participation reduce the chances of introducing a bug to code [22, 23, 3]. Unfortunately, this process is often not able to filter all defective code. The defective code that passes through the code review process might be the result of poor review quality. The process of solving defects is very expensive, and, therefore, the possibility of in-advance determination if the code changes might introduce a bug would be invaluable.

In September 2020, Doğan [11] identified seven bad practices that could be correlated with a lower quality of reviews and named them "code review smells". Six of them were then described in detail and searched for in open-source projects. The results of this research were later published in [12]. Unfortunately, the article did not address the correlation between the quality of the review measured by "code review smells" and inducing defects.

Using these sources as a reference point, our objective was to utilize code review smells and metrics to predict inducing software defects with pull requests. Although there are some papers on the use of code smells as predictors of software defects, see [26], to our knowledge, this paper is the first that uses code review smells to predict software defects.

2 Background

Doğan and Tüzün [12] have identified seven and defined six smells of code reviews. The aim of this research is to check whether, based on code review smells and metrics, one is able to predict if a pull request might induce a defect. To achieve this goal, the following steps were executed:

- gathering information on other review qualities which possibly impact probability of inducing software defect,
- finding alternative definitions of those qualities,
- finding already used code review metrics,
- introducing new code review metrics,
- checking whether those code review metrics and smells can be helpful to predict software defects.

For the search for relevant literature, the following research questions were defined:

RQ1 Is it possible to utilize code review smells as predictors of software defects for pull requests?
RQ2 Is it possible to derive metrics from the code review smells defined by Doğan and Tüzün [12]?

RQ3 Is it possible to utilize code review quality metrics as predictors of software defects for pull requests?

2.1 Relevant Literature

In order to answer the defined research questions, code review smells and metrics needed to be gathered. During the study selection process, search strings were defined that cover the subject area, titles, abstracts, and keywords. The results of this query were then analyzed for relevance; possibly relevant sources were read in full to check whether they can be helpful when pursuing the goal of this paper.

2.1.1 Search strings

All subjects of our interest can be described with the following search string:

```
TITLE-ABS-KEY(code-review* AND (quality OR metric* OR
smell*OR impact)) AND (PUBYEAR > 2014) AND (LIMIT-TO(
SUBJAREA, "COMP")) AND (LIMIT-TO(LANGUAGE, "English"))
```

2.1.2 Results of research for relevant literature

After evaluating the aforementioned search string on April 4, 2022 Scopus returned 341 results. 28 were labeled possibly relevant based on their title and abstract, but after full-text analysis it appeared that only 16 of them contained information on review-related smells and metrics.

Figure 1 illustrates the process of literature selection; Appendix contains lists of articles accepted during first and second screening stage.

2.1.3 Already defined metrics

Table 1 contains information on explicitly defined metrics and those deduced from relevant articles that are considered to be related to the review and possibly influence code quality.

3 Methods and Materials

The conducted research consisted of several stages. First, we select repositories to evaluate, after that we aim to reproduce the research by Doğan and Tüzün [12] by implementing code review smells in our code base, then we develop new metrics

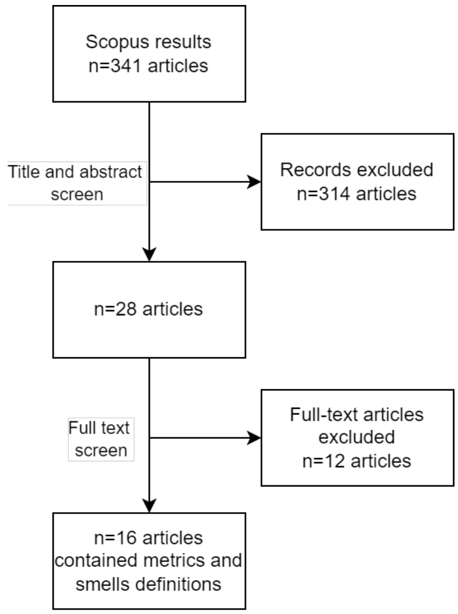


Fig. 1 Literature selection process

that potentially might have impact on inducing a bug. Afterwards, we import metrics from the dataset [17]. In the end, we feed a machine learning model with metrics and smells.

3.1 Reproduction of Doğan and Tüzün research

The process of reproduction does not fully reflect the process provided by Doğan and Tüzün[12], because our strategy relies on a larger amount of data. That is why the data are stored in a database. Data on pull requests, users, files, their changes, and reviews are retrieved from GitHub by means of the GitHub API for chosen repositories and saved to the database. Then it is processed according to the specification provided by the aforementioned paper.

For reproduction, VS Code, Tensorflow, and GitHub Desktop repositories were chosen. The smell detection results are shown in Table 2 and are consistent with the original research.

Table 1 Metrics and smells found in or deduced from relevant literature

Smell / Metric	Type	Description	Source
No review	smell	PR closed without review	[22, 29, 28, 21]
Self review	smell	PR contains only self-review	[22, 29, 21]
Number of hasty reviews	metric	Number of reviews with checked 200loc/h	[22, 29, 21]
Number of short reviews	metric	Number of short reviews	[9]
Number of superficial reviews	metric	Number of superficial reviews	[9]
Number of reviews without inline comments	metric	Number of negative reviews without inline comments	[9]
Number of reviews	metric	Number of reviews	[3, 19, 20, 29, 28, 31, 10, 9, 25, 33, 21, 32]
Number of author responses	metric	Number of author responses to reviews	[19]
Number of reviews to churn ratio	metric	Number of reviews divided by number of changed loc	[21]
Review window	metric	Time PR was opened for reviewing	[29, 31, 25, 33]
Review window to churn ratio	metric	Time PR was opened for reviewing divided by changed loc	[21]
Review delay	metric	Time between PR was opened and first review	[28, 31, 10, 33, 32]
Number of reviewers	metric	Number of unique reviewers	[3, 19, 20, 31, 10, 25, 33, 21, 32]
Number of non-author reviewers	metric	Number of reviewers who didn't change code to be merged	[31, 9, 33]
Disagreement ratio	metric	Ratio of non-approval reviews	[31, 9, 33]
Number of revisions	metric	Number of commits after PR was opened	[28, 31, 16, 33, 21]
Number of revisions without review	metric	Number of non-commented commits added in review window	[31, 33]
Churn during code review	metric	Loc changed during review window	[31, 33]
Negative sentiment in reviews	metric	Determined coefficient of negative sentiment	[9]
Confused reviews	metric	Coefficient of confusion based on keywords	[13, 9]
Number of discussion observers	metric	Number of discussion observers	[19]
Important keywords used	metric	# of positive minus # of negative keywords	[6]
Reviewing time	metric	Average time spent on reviewing	[28, 16]
Review pace	metric	Average churn reviewed per hour	[31]
Shepherding time	metric	Average time spent on review-related activities	[16]

Table 2 Reproduction results for Doğan and Tüzün research

Smell	VS Code	Desktop	Tensorflow
Lack of review	57.57%	14.25%	12.48%
Missing PR description	24.51%	11.21%	43.93%
Large changeset	7.97%	5.25%	9.94%
Sleeping review	40.13%	41.39%	47.82%
Ping-pong	4.19%	10.67%	9.08%
At least one of: – lack of review – missing PR description – large changeset – sleeping review – ping-pong	83.37%	63.16%	81.89%
Review buddies[1]	3.25%	7.39%	11.71%

3.2 Metrics

Doğan and Tüzün [12] have divided the metrics based on their extraction/calculation method into ones extracted directly from pull requests, ones regarding single reviews (but still collected for each pull request) and those calculated for the whole repository (also assigning the results to single pull requests). Some additional metrics might also be implemented as part of future work.

3.2.1 Imported metrics

We imported product metrics assigned to commits in the dataset from the article by Keshavarz and Nagappan [18]. These metrics are:

- change date
- # of lines added
- # of lines deleted
- # of files touched
- # of directories touched
- # of of subsystems touched
- change entropy
- # of of distinct developers touched files
- the average time from last change
- # of of unique changes in files
- change author experience
- change author
- recent experience

[1] Smell *Review buddies* was defined but not measured by Doğan and Tüzün [12].

- change author subsystem experience

3.2.2 New metrics

We defined several metrics of the code review process to feed the machine learning model with:

- **Number of reviewers**
  It simply checks how many reviewers have reviewed a pull request.

- **Number of reviewers different than the pull-request author**
  This metric is an improvement of the Number of reviewers metric. In the calculation, it excludes the reviewer who created a pull request. It is worth mentioning that the author of a pull request may be different from the author of changes in code (especially when multiple people have been contributing to the code).

- **Number of reviews**
  This metric evaluates the number of reviews (without checking their authors) for a given pull request.

- **Number of commits after pull-request creation**
  This metric counts commits that were added after the pull request creation date. It is assumed that such commits introduce improvements and are the result of submitted reviews. It is also expected that a pull request containing these commits is less likely to introduce a defect.

- **Number of lines changed after pull-request creation**
  Reviews requesting some changes should result in new commits with improvements. This metric counts the number of lines that were changed as a result of a code review. It is worth mentioning that all changes introduced after a pull request's creation date are considered improvements, no matter if there were already some reviews submitted.

- **Review length (number of characters)**
  Review length metric counts the number of characters in each review for a pull request. At the time of implementation, it is still unclear whether a bigger or smaller number of characters is better. This metric is related to some other code review metrics, e.g., Number of reviews and some project metrics, e.g., number of changes lines.

- **Review window per changed lines**
  This metric takes into account both the time passed between opening and closing a pull request and the number of changed lines in order to calculate the ratio between those two values. This way it is possible to establish more flexible

threshold values to mark a pull request as smelly.

- **Reviewed lines per hour**
  Reviewed lines per hour metric measures how many lines were changed for a given pull request and calculates a ratio between this value and the opening hour a the pull request.

- **Review length per lines of code**
  This metric is an amplification of the Review length metric. It calculates the ratio between the number of characters in all reviews in a given pull request and the number of changed lines.

- **Review window**
  This metric is extracted by calculating the time passed between opening and closing a pull request. It has a few known flaws, as it does not consider the actual time spent reviewing a request (limitations are mentioned in Section 5). Hence, the request can be open for so long that it will be considered smelly, but still be reviewed superficially.

- **Review window per line**
  Reviews that last too long are considered smelly, but their duration should be related to number of changed lines. Obviously, it is possible to change a lot of lines of code barely changing the program logic or not changing it at all (e.g. by renaming a variable or a function). Such cases should rather be the minority.

3.3 Data preparation

We used the dataset by Keshavarz [17] to assign a value if a commit induces a bug. This dataset consisted of commits from 12 repositories of Apache projects. We have downloaded the data of pull requests and reviews for the commits. With the data, three random forest models can be trained.

We encountered a problem with granularity. We need to know if a pull request induces a misbehaviour, not a commit (as it is in the dataset). We solved the granularity problem deciding that a pull request is bug-inducing if one of the commits is. The product metrics were assigned to the pull request using trimmed means of 10%.

3.4 Implementation

We implemented models generation with functions detecting smells and evaluating metrics from review-related data using Python scripts. Our implementation can be found in Github repository as explained in Appendix

4 Results

Three models were created, trained on 513 pull requests (PRs) and tested on 171 remaining entries, and these models will be marked as:

M1 based on metrics from dataset by Keshavarz and Nagappan [18],
M2 based on five of the smells created by Doğan and Tüzün [12] (*large changesets* smell is not related to the quality of reviews, thus was omitted) and new metrics (see Section 3.2.2) we developed and
M3 model which combines both aforementioned sources of PR evaluators.

Table 3 presents errors obtained when predicting whether PR is buggy for each of the models for all GitHub repositories that exist in the dataset by Keshavarz and Nagappan. As one can see, M2 has similar performance to M1 and is more or less able to determine whether PR introduced bugs. It allows us to answer positively to RQ1 and RQ3.

Table 3 Errors for prepared models

Error type	M1	M2	M3
Mean absolute error	0.26	0.29	0.25
Mean squared error	0.13	0.16	0.13
Root mean squared error	0.37	0.40	0.36

In order to determine which metrics are most important for our experimental models, metrics importance was evaluated based on mean decrease in impurity (MDI). For M2, as shown in Figure 2, the most important features included *review window* (41%), *review window per line* (26%), *reviews characters per line of code* (10%) and *total number of reviews characters* (9%). For the combined model (M3), the *lines added* were the most influential (33%), the rest of the metrics and smells had an importance below 5% (e.g., *review window* 4%), this is shown in Figure 3. It can be explained for some metrics with their boxplots for buggy and non-buggy pull requests (see Appendix), where all other than *review window* and *review window per line* do not show significant differences between those two groups. The entire report is available in Appendix, including figures.

As can be seen in Figure 4, some metrics have a very high correlation; therefore, the ones with the highest correlation could be removed.

In the context of the study, we formulated the following research questions:

RQ1 Is it possible to utilize code review smells as predictors of software defects for pull requests?
Yes, they were successfully utilized in software defects predicting models M2 and M3 as shown in Section 4.

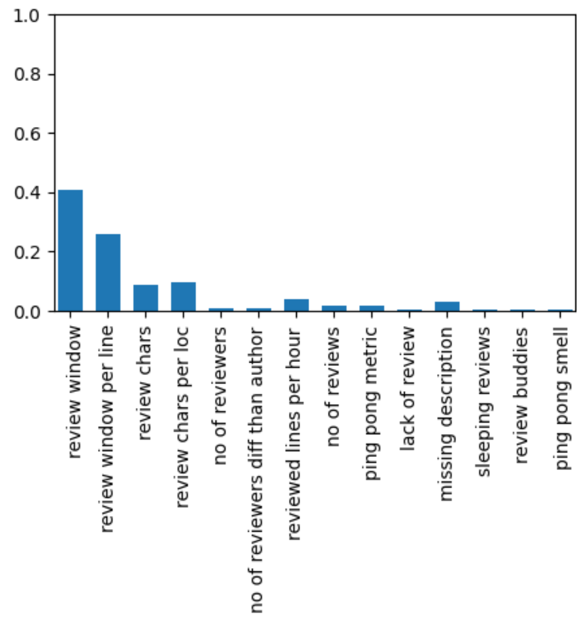


Fig. 2 Metrics and smells importance for M2

RQ2 Is it possible to derive metrics from code review smells defined by Doğan and Tüzün?
Yes, we have derived 11 metrics what was presented in Section 3.2.2.

RQ3 Is it possible to utilize code review quality metrics as predictors of software defects for pull requests?
Yes, they were successfully utilized in software defects predicting models M2 and M3 as shown in Section 4.

5 Discussion

Analysis of nine Apache repositories showed code review smells and metrics can be used to predict bug introduction with accuracy similar to model based on product-related metrics. This opens the possibility to enhance existing models by adding a whole new category of metrics and smells, which would be related to the review process.

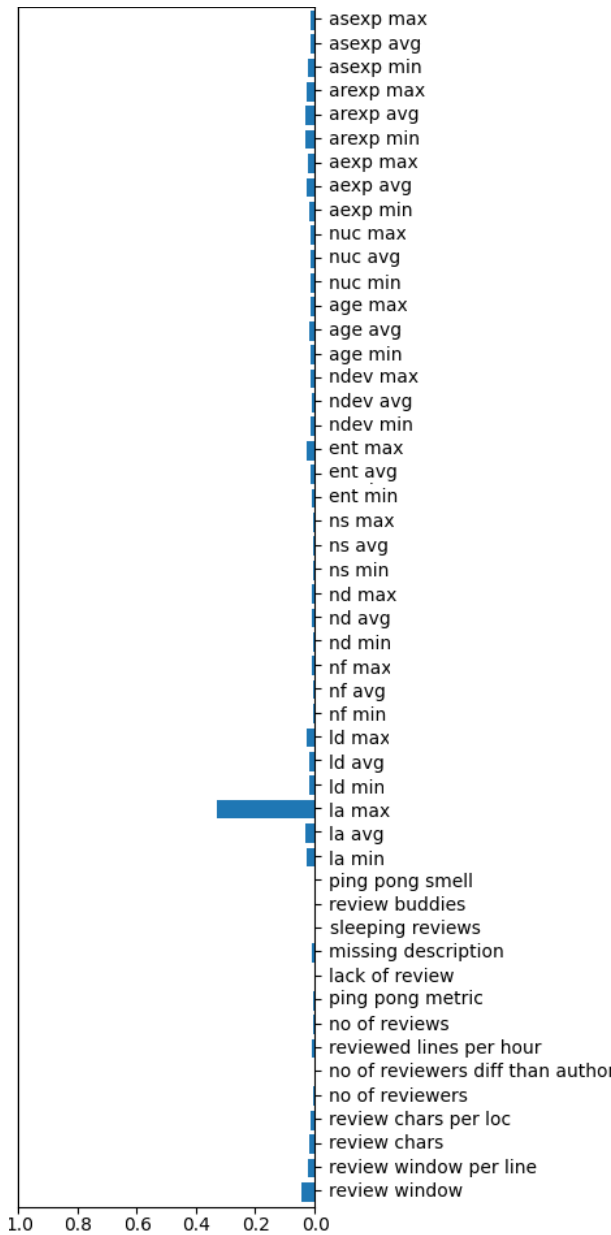


Fig. 3 Metrics and smells importance for M3

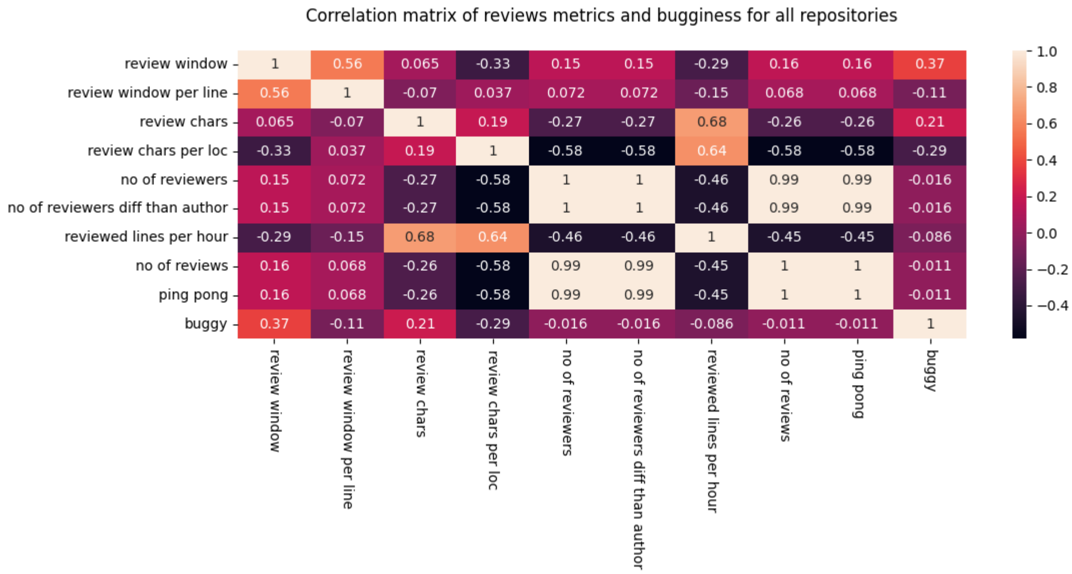


Fig. 4 Correlation of metrics from Section 3.2.2

During the process of developing the model certain limitations were identified:

- Some commits that the dataset included could not be found from the GitHub API. That caused the model to use less training and testing data than it would be relevant.
- The dataset and our research had different levels of granularity; therefore, the metrics for commit could not be easily applied on a pull request.
- GitHub does not measure the time that a reviewer has spent on a review, hence this attribute is not accessible for our research. Other review tools—for instance, Crucible—include the time spent. This is especially important, considering the fact that the metrics based on time - *review window* appeared to be the most impactful.

This work could have more reliable results and a broader scope if these limitations were overcome.

There are areas where this work can be improved and included in future works:

- *User metrics*
  such as reviewer reputation based on number of reviews made, the number of projects contributed, etc.
- *Review content*
  NLP (natural language processing) of reviews to estimate their relevance and assess code changes based on reviewers' opinions.

- *Conflicting reviews*
  Calculated by checking if multiple reviews regarding the same changes in code approve and disapprove them.
- *Dataset*
  The results might differ once more data is provided, as mentioned in limitations. Then the introduced model could be used to retrieve more relevant information on the impact of the developed metrics.

It was discovered that reviewing process descriptors, such as those in Table 1, have high potential when it comes to predicting bug introduction and should be included in relevant models.

6 Conclusions

All posed research questions were answered and the results open up a new promising research direction.

Using the answers and models defined in Section 4, it was shown that the review smells and metrics allow predicting pull request bugginess to a similar extent as classic software product metrics; however, this model is not fully satisfying and performs better with more review metrics or when combined with software product metrics. Thus, standard product metrics still remain important features of prediction models. There is also room to introduce improvements and modifications to the method used in this investigation, as described in Section 5.

CRediT authorship contribution statement

Krzysztof Baciejowski: Software, Data curation, Investigation, Writing – original draft, Writing – review & editing, Visualization. **Damian Garbala**: Software, Investigation, Writing – original draft **Szymon Żmijewski**: Software, Investigation, Writing – original draft **Lech Madeyski**: Conceptualization, Methodology, Writing – review & editing, Supervision.

Acknowledgment

The paper is an outcome of the Research and Development Project in Software Engineering at Wroclaw University of Science and Technology.

Appendix

Reproduction

Code utilized to perform reproduction of Doğan and Tüzün [12] is available on Github (github.com/pwr-pbr22/M7/tree/reproduction). Scripts used to prepare models are available in the same repository on the main branch (github.com/pwr-pbr22/M7). Reproduction instructions are available in respective README files.

Relevant literature search

As mentioned in Section 2.1.2 28 articles passed title and abstract screen, 16 of them passed full text screen and 12 were excluded. These articles can are listed below.

- Articles which passed title and abstract screen, but were excluded during full text screen: [24, 27, 2, 7, 34, 1, 30, 5, 15, 4, 14, 8].
- Articles which passes title and abstract screen and were deemed relevant after full text screen: [22, 3, 20, 6, 19, 29, 28, 31, 13, 10, 9, 25, 16, 33, 21, 32].

Appendix

Appendix includes the report, located below, from the implemented Jupyter Notebook.

Model based solely on data from *A Large Dataset for Just-In-Time Defect Prediction*

```
C:\Users\kbaci\AppData\Local\Programs\Python\Python310\lib\site-packages\pandas\core\indexes
\base.py:6982: FutureWarning: In a future version, the Index constructor will not infer numer
ic dtypes when passed object-dtype sequences (matching Series behavior)
  return Index(sequences[0], name=names)
```

Selected training and testing sets

```
Training features shape:         (513, 36)
Training labels shape:           (513,)
Testing features shape:          (171, 36)
Testing labels shape:            (171,)
```

Prediction errors

```
Mean absolute error:            0.26
Mean squared error:             0.13
Root Mean squared error:        0.36
```

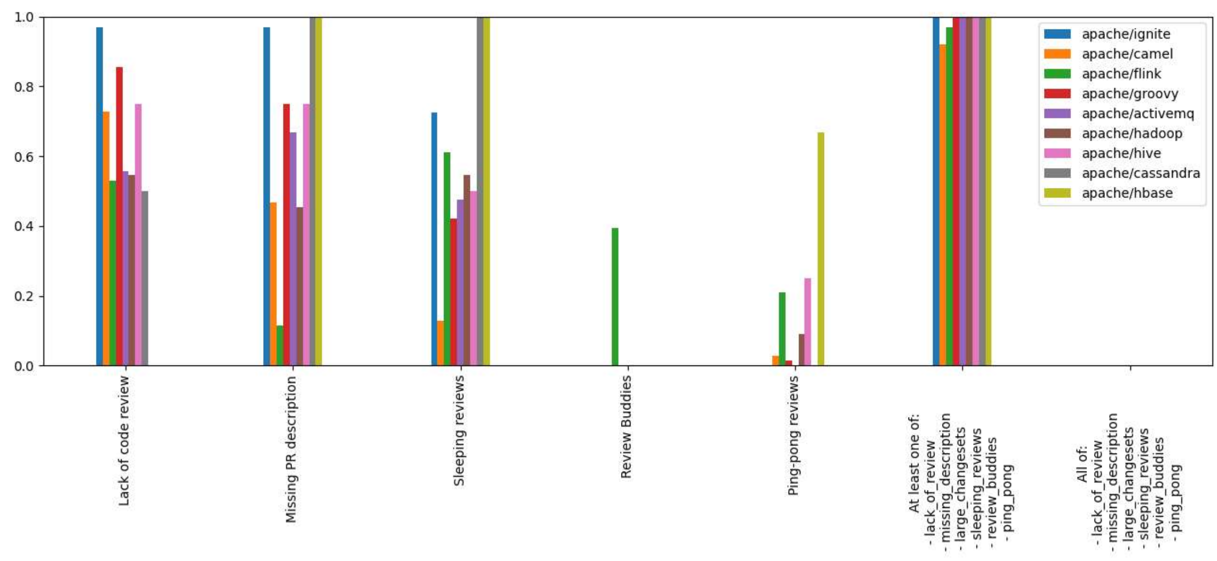
Model based on our metrics and smells

Smells

Share of smelly pulls

```
Smell / repository              ignite          camel           flink           groovy
activemq        hadoop          hive            cassandra       hbase
Lack of code review                 96.77%          72.66%          52.97%          85.53%
55.56%      54.55%          75.0%           50.0%           0.0%
Missing PR description              96.77%          46.76%          11.35%          75.0%
66.67%      45.45%          75.0%          100.0%          100.0%
Sleeping reviews                    72.58%          12.95%          61.08%          42.11%
47.62%      54.55%          50.0%          100.0%          100.0%
Review Buddies                        0.0%            0.0%          39.46%            0.0%
0.0%          0.0%           0.0%            0.0%            0.0%
Ping-pong reviews                     0.0%            2.88%         21.08%            1.32%
0.0%          9.09%          25.0%            0.0%           66.67%
At least one of:
- lack_of_review
- missing_description
- large_changesets
- sleeping_reviews
- review_buddies
- ping_pong                         100.0%          92.09%          96.76%          100.0%
100.0%      100.0%         100.0%          100.0%          100.0%
All of:
- lack_of_review
- missing_description
- large_changesets
- sleeping_reviews
- review_buddies
- ping_pong                           0.0%            0.0%            0.0%            0.0%
0.0%          0.0%           0.0%            0.0%            0.0%
```

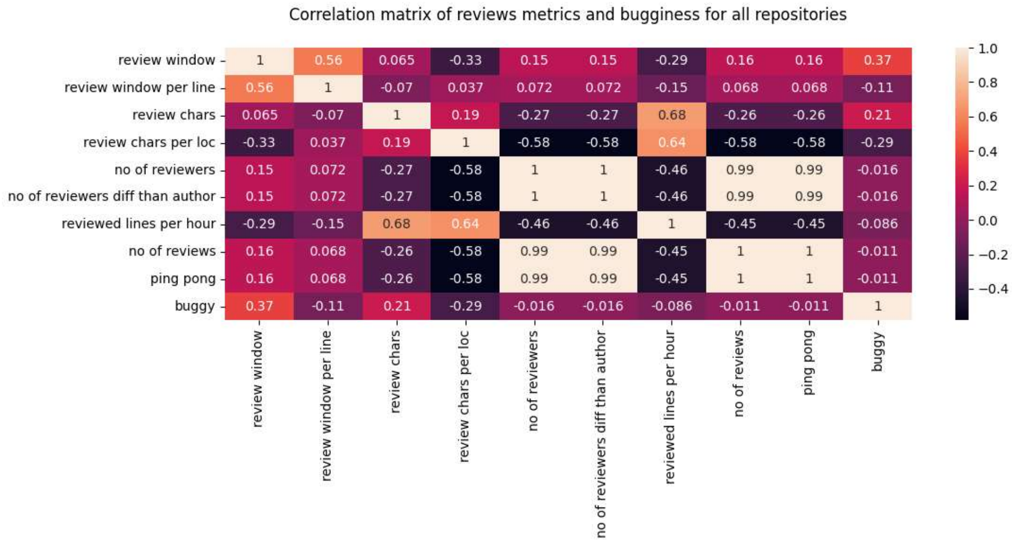
Figure



Metrics

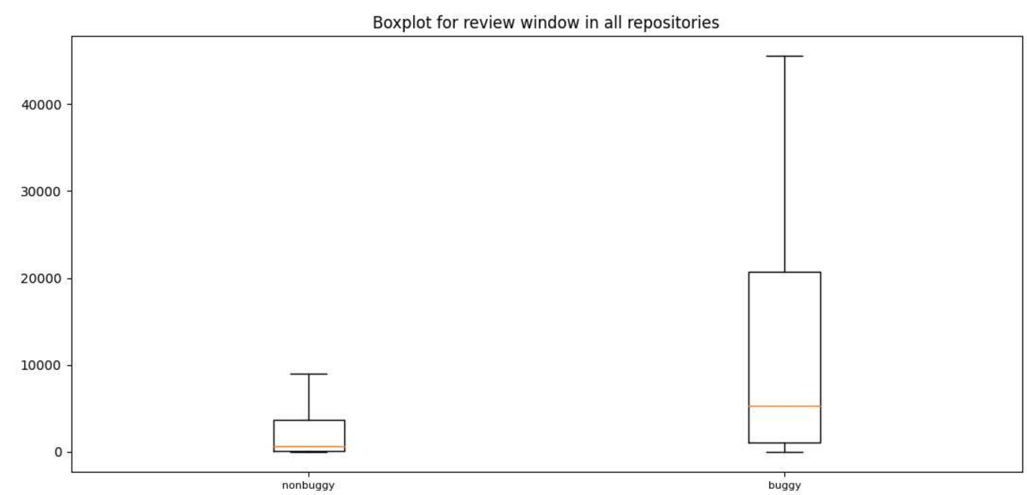
Correlation between metrics

Figure



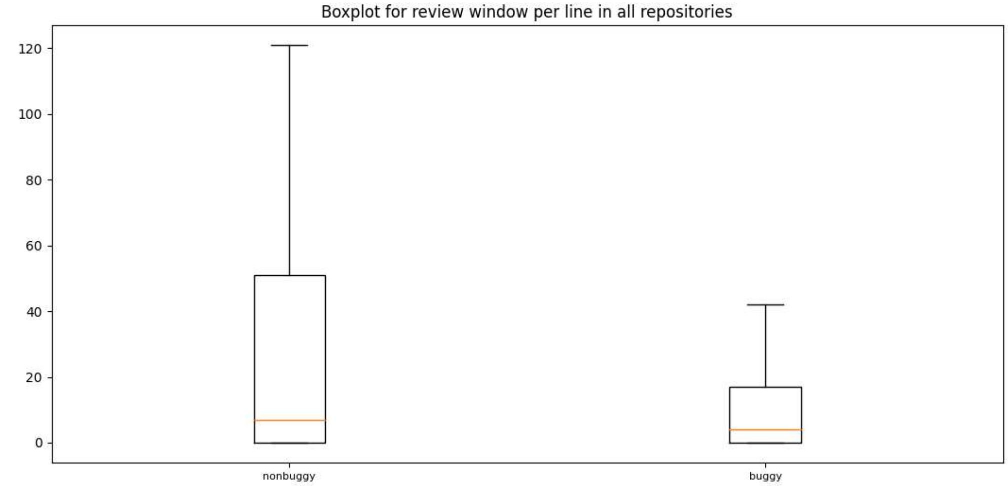
Boxplots for different metrics

Figure



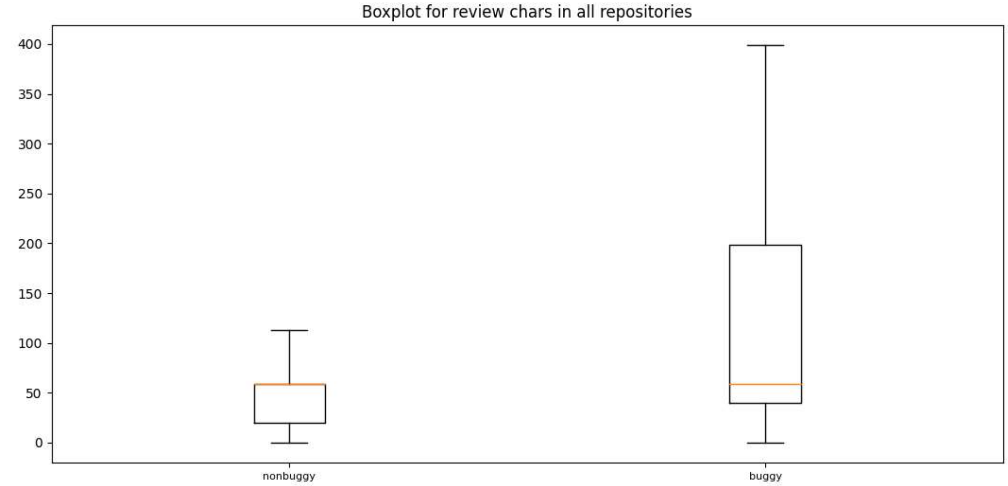
Figure

Boxplot for review window per line in all repositories



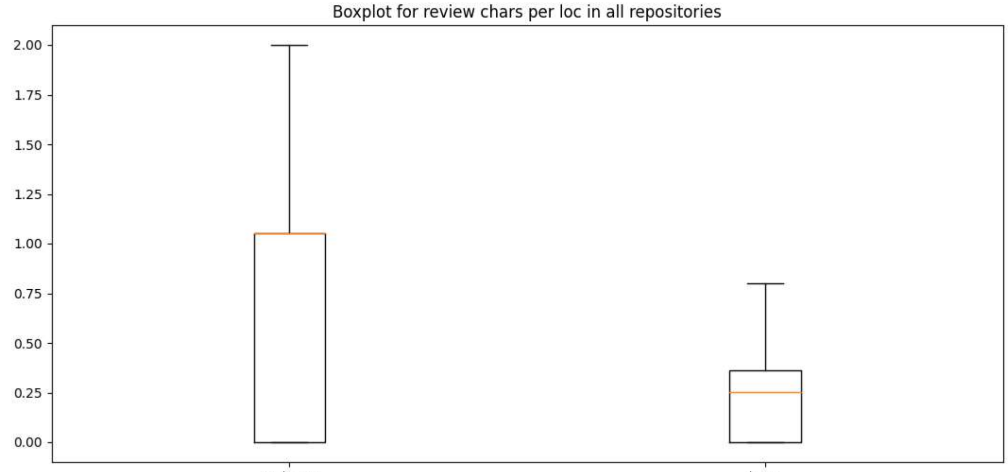
Figure

Boxplot for review chars in all repositories

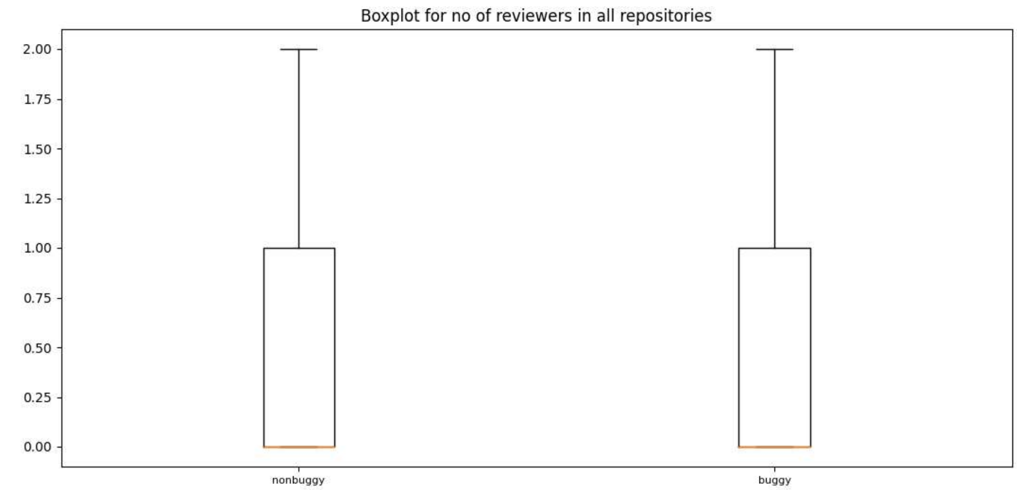


Figure

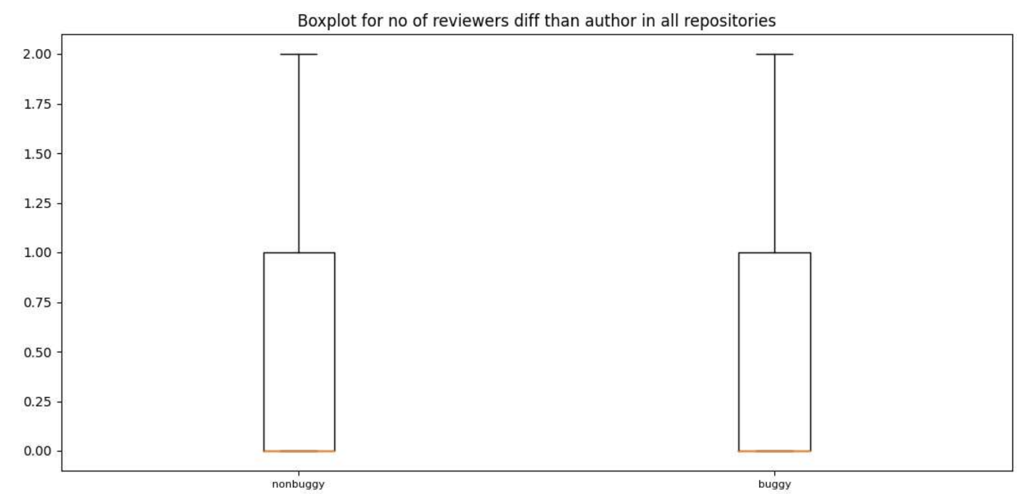
Boxplot for review chars per loc in all repositories



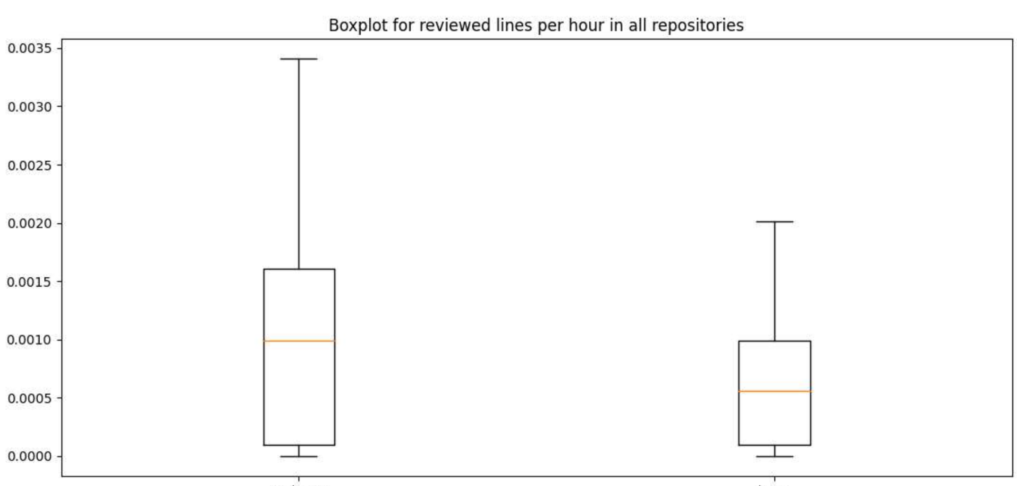
Figure



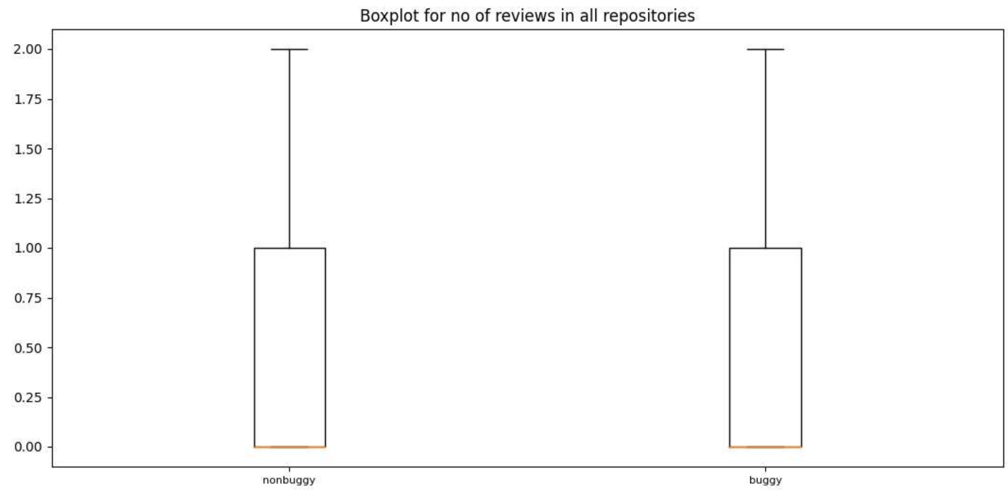
Figure



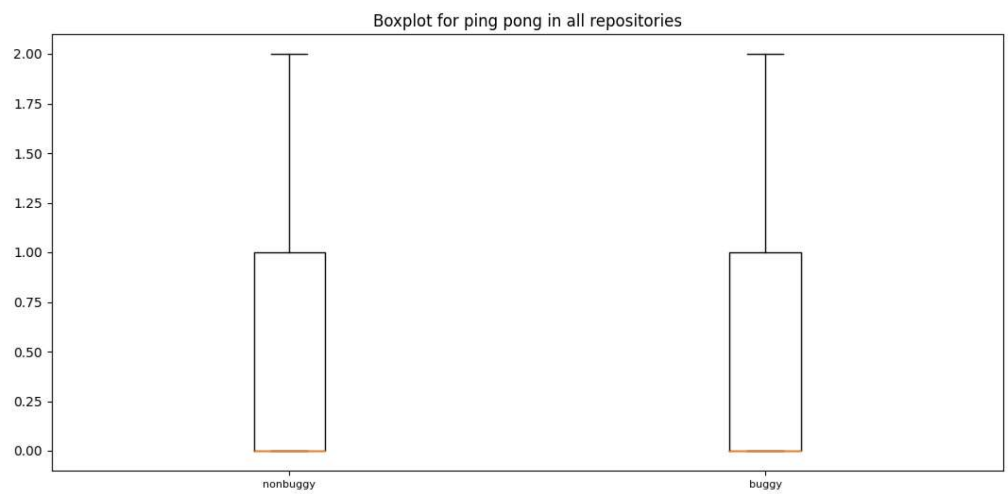
Figure



Figure



Figure



Model

Selected training and testing sets

```
Training features shape:        (513, 14)
Training labels shape:          (513,)
Testing features shape:         (171, 14)
Testing labels shape:           (171,)
```

Prediction errors

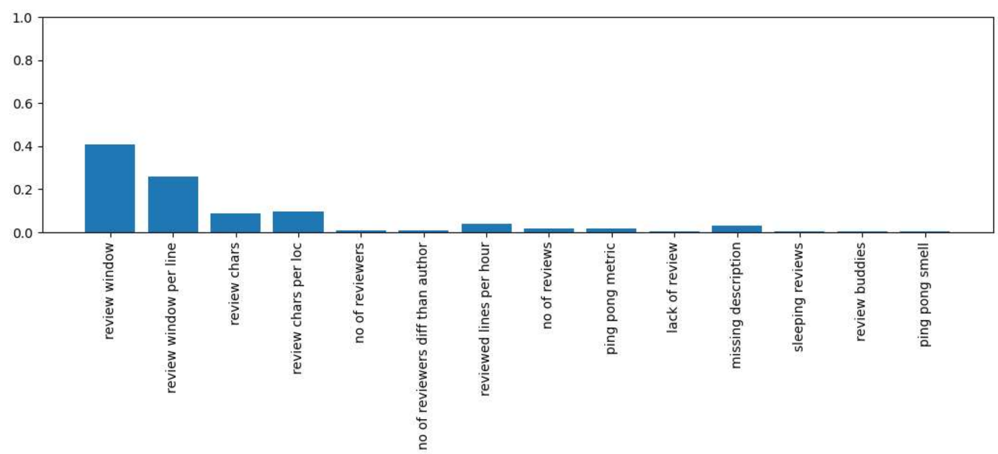
```
Mean absolute error:          0.29
Mean squared error:           0.16
Root Mean squared error:      0.4
```

Metrics importance

```
Metrics                           Importance
review window                     0.41
review window per line            0.26
review chars per loc              0.1
review chars                      0.09
reviewed lines per hour           0.04
missing description               0.03
no of reviews                     0.02
ping pong metric                  0.02
no of reviewers                   0.01
no of reviewers diff than author 0.01
sleeping reviews                  0.01
review buddies                    0.01
lack of review                    0.0
ping pong smell                   0.0
```

```
C:\Users\kbaci\AppData\Local\Temp\ipykernel_8764\4159194392.py:3: UserWarning: FixedFormatter should only be used together with FixedLocator
  ax.set_xticklabels(list(map(lambda e: e.replace('_',' '), feature_list)),rotation="vertica
l")
```

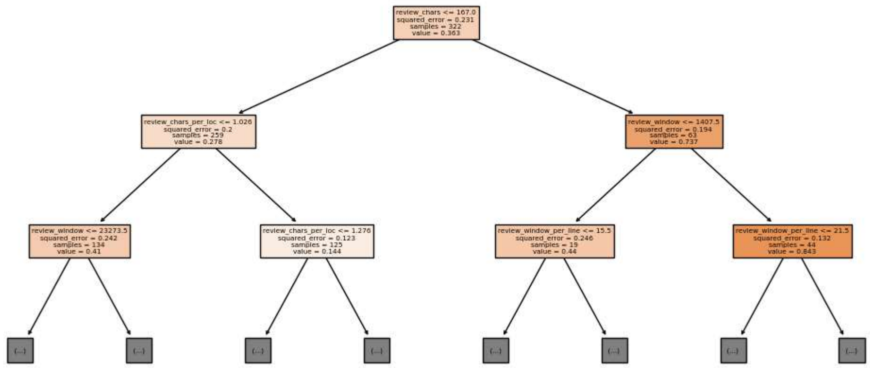
Figure



Example of a tree

Figure

depth = 19



Combined model

```
C:\Users\kbaci\AppData\Local\Programs\Python\Python310\lib\site-packages\pandas\core\indexes
\base.py:6982: FutureWarning: In a future version, the Index constructor will not infer numer
ic dtypes when passed object-dtype sequences (matching Series behavior)
  return Index(sequences[0], name=names)
```

Selected training and testing sets

```
Training Features Shape: (513, 50)
Training Labels Shape: (513,)
Testing Features Shape: (171, 50)
Testing Labels Shape: (171,)
```

Prediction errors

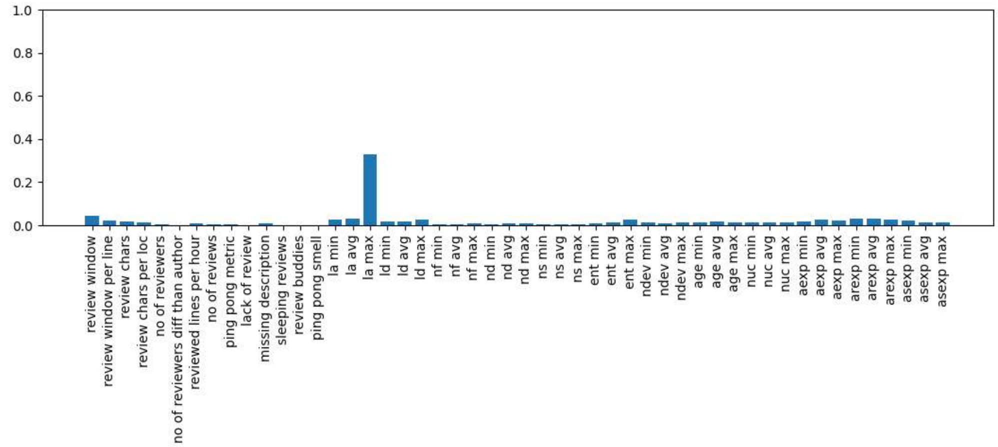
```
Mean absolute error:            0.25
Mean squared error:             0.13
Root Mean squared error:        0.36
```

Metrics importance

Metrics	Importance
la max	0.33
review window	0.04
la min	0.03
la avg	0.03
ld max	0.03
aexp avg	0.03
arexp min	0.03
arexp avg	0.03
arexp max	0.03
review window per line	0.02
review chars	0.02
ld min	0.02
ld avg	0.02
ent max	0.02
age avg	0.02
nuc min	0.02
aexp min	0.02
aexp max	0.02
asexp min	0.02
asexp avg	0.02
review chars per loc	0.01
reviewed lines per hour	0.01
missing description	0.01
nf avg	0.01
nf max	0.01
nd avg	0.01
nd max	0.01
ns avg	0.01
ent min	0.01
ent avg	0.01
ndev min	0.01
ndev avg	0.01
ndev max	0.01
age min	0.01
age max	0.01
nuc avg	0.01
nuc max	0.01
asexp max	0.01
no of reviewers	0.0
no of reviewers diff than author	0.0
no of reviews	0.0
ping pong metric	0.0
lack of review	0.0
sleeping reviews	0.0
review buddies	0.0
ping pong smell	0.0
nf min	0.0
nd min	0.0
ns min	0.0
ns max	0.0

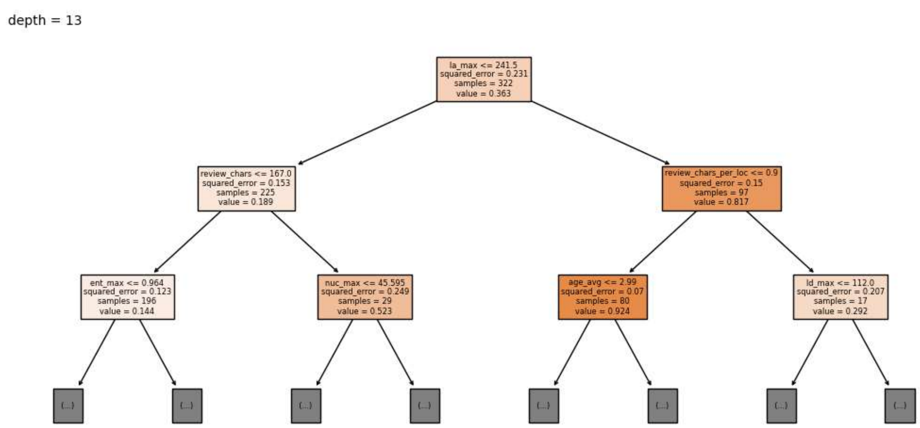
```
C:\Users\kbaci\AppData\Local\Temp\ipykernel_8764\4159194392.py:3: UserWarning: FixedFormatter should only be used together with FixedLocator
  ax.set_xticklabels(list(map(lambda e: e.replace('_',' '), feature_list)),rotation="vertical")
```

Figure



Example of tree

Figure



References

[1] Alami A, Cohn ML, Wasowski A (2019) Why does code review work for open source software communities? In: Proceedings of the 41st International Conference on Software Engineering, IEEE Press, ICSE '19, p 1073–1083, DOI 10.1109/ICSE.2019.00111, URL https://doi.org/10.1109/ICSE.2019.00111

[2] Baum T, Liskin O, Niklas K, Schneider K (2016) Factors influencing code review processes in industry. In: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, Association for Computing Machinery, New York, NY, USA, FSE 2016, p 85–96, DOI 10.1145/2950290.2950323, URL https://doi.org/10.1145/2950290.2950323

[3] Bavota G, Russo B (2015) Four eyes are better than two: On the impact of code reviews on software quality. In: 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp 81–90, DOI 10.1109/ICSM.2015.7332454

[4] Baysal O, Kononenko O, Holmes R, Godfrey MW (2016) Investigating technical and non-technical factors influencing modern code review. Empirical Software Engineering 21(3):932–959, DOI 10.1007/s10664-015-9366-8, URL https://doi.org/10.1007/s10664-015-9366-8

[5] di Biase M, Bruntink M, Bacchelli A (2016) A security perspective on code review: The case of chromium. In: 2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation (SCAM), pp 21–30, DOI 10.1109/SCAM.2016.30

[6] Bosu A, Greiler M, Bird C (2015) Characteristics of useful code reviews: An empirical study at microsoft. In: 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories, pp 146–156, DOI 10.1109/MSR.2015.21

[7] Bosu A, Carver JC, Bird C, Orbeck J, Chockley C (2017) Process aspects and social dynamics of contemporary code review: Insights from open source development and industrial practice at microsoft. IEEE Transactions on Software Engineering 43(1):56–75, DOI 10.1109/TSE.2016.2576451

[8] Caulo M, Lin B, Bavota G, Scanniello G, Lanza M (2020) Knowledge Transfer in Modern Code Review, Association for Computing Machinery, New York, NY, USA, p 230–240. URL https://doi.org/10.1145/3387904.3389270

[9] Chouchen M, Ouni A, Kula RG, Wang D, Thongtanunam P, Mkaouer MW, Matsumoto K (2021) Anti-patterns in modern code review: Symptoms and prevalence. In: 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), pp 531–535, DOI 10.1109/SANER50967.2021.00060

[10] Davila N, Nunes I (2021) A systematic literature review and taxonomy of modern code review. Journal of Systems and Software 177:110,951, DOI https://doi.org/10.1016/j.jss.2021.110951, URL https://www.sciencedirect.com/science/article/pii/S0164121221000480

[11] Doğan E (2020) Towards a taxonomy of code review smells. Master's thesis, Bilkent University, URL http://hdl.handle.net/11693/54183
[12] Doğan E, Tüzün E (2022) Towards a taxonomy of code review smells. Information and Software Technology 142:106,737, DOI https://doi.org/10.1016/j.infsof.2021.106737, URL https://www.sciencedirect.com/science/article/pii/S0950584921001877
[13] Ebert F, Castor F, Novielli N, Serebrenik A (2017) Confusion detection in code reviews. In: 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp 549–553, DOI 10.1109/ICSME.2017.40
[14] Ebert F, Castor F, Novielli N, Serebrenik A (2019) Confusion in code reviews: Reasons, impacts, and coping strategies. In: 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp 49–60, DOI 10.1109/SANER.2019.8668024
[15] Efstathiou V, Spinellis D (2018) Code review comments: Language matters. In: Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results, Association for Computing Machinery, New York, NY, USA, ICSE-NIER '18, p 69–72, DOI 10.1145/3183399.3183411, URL https://doi.org/10.1145/3183399.3183411
[16] Egelman CD, Murphy-Hill E, Kammer E, Hodges MM, Green C, Jaspan C, Lin J (2020) Predicting developers' negative feelings about code review. In: 2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE), pp 174–185
[17] Keshavarz H, Nagappan M (2022) ApacheJIT: A Large Dataset for Just-In-Time Defect Prediction. DOI 10.5281/zenodo.5907847, URL https://doi.org/10.5281/zenodo.5907847
[18] Keshavarz H, Nagappan M (2022) Apachejit: A large dataset for just-in-time defect prediction. ArXiv abs/2203.00101
[19] Kononenko O, Baysal O, Guerrouj L, Cao Y, Godfrey MW (2015) Investigating code review quality: Do people and participation matter? In: 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp 111–120, DOI 10.1109/ICSM.2015.7332457
[20] Kononenko O, Baysal O, Godfrey MW (2016) Code review quality: How developers see it. In: 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), pp 1028–1038, DOI 10.1145/2884781.2884840
[21] Krutauz A, Dey T, Rigby PC, Mockus A (2020) Do code review measures explain the incidence of post-release defects? Empirical Software Engineering 25(5):3323–3356, DOI 10.1007/s10664-020-09837-4, URL https://doi.org/10.1007/s10664-020-09837-4
[22] McIntosh S, Kamei Y, Adams B, Hassan AE (2014) The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects. In: Proceedings of the 11th Working Conference on Mining Software Repositories, Association for Computing Machinery, New York, NY, USA, MSR 2014, p 192–201, DOI 10.1145/2597073.2597076, URL https://doi.org/10.1145/2597073.2597076

[23] McIntosh S, Kamei Y, Adams B, Hassan AE (2016) An empirical study of the impact of modern code review practices on software quality. Empirical Software Engineering 21(5):2146–2189, DOI 10.1007/s10664-015-9381-9, URL https://doi.org/10.1007/s10664-015-9381-9

[24] Morales R, McIntosh S, Khomh F (2015) Do code review practices impact design quality? a case study of the qt, vtk, and itk projects. In: 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER), pp 171–180, DOI 10.1109/SANER.2015.7081827

[25] Paul R, Turzo AK, Bosu A (2021) Why security defects go unnoticed during code reviews? a case-control study of the chromium os project. In: 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), pp 1373–1385, DOI 10.1109/ICSE43902.2021.00124

[26] Piotrowski P, Madeyski L (2020) Software defect prediction using bad code smells: A systematic literature review. In: Poniszewska-Marańda A, Kryvinska N, Jarząbek S, Madeyski L (eds) Data-Centric Business and Applications: Towards Software Development (Volume 4), vol 40 of book series Lecture Notes on Data Engineering and Communications Technologies, Springer International Publishing, Cham, pp 77–99, DOI 10.1007/978-3-030-34706-2_5, URL https://doi.org/10.1007/978-3-030-34706-2_5

[27] Rahman MM, Roy CK, Kula RG (2017) Predicting usefulness of code review comments using textual features and developer experience. In: 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR), pp 215–226, DOI 10.1109/MSR.2017.17

[28] dos Santos EW, Nunes I (2018) Investigating the effectiveness of peer code review in distributed software development based on objective and subjective data. Journal of Software Engineering Research and Development 6(1):14, DOI 10.1186/s40411-018-0058-0, URL https://doi.org/10.1186/s40411-018-0058-0

[29] Shimagaki J, Kamei Y, Mcintosh S, Hassan AE, Ubayashi N (2016) A study of the quality-impacting practices of modern code review at sony mobile. In: 2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C), pp 212–221

[30] Sri-iesaranusorn P, Kula RG, Ishio T (2021) Does code review promote conformance? a study of openstack patches. In: 2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR), pp 444–448, DOI 10.1109/MSR52588.2021.00056

[31] Thongtanunam P, McIntosh S, Hassan AE, Iida H (2015) Investigating code review practices in defective files: An empirical study of the qt system. In: 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories, pp 168–179, DOI 10.1109/MSR.2015.23

[32] Thongtanunam P, McIntosh S, Hassan AE, Iida H (2017) Review participation in modern code review. Empirical Software Engineering 22(2):768–817, DOI 10.1007/s10664-016-9452-6, URL https://doi.org/10.1007/s10664-016-9452-6

[33] Uchôa A, Barbosa C, Oizumi W, Blenilio P, Lima R, Garcia A, Bezerra C (2020) How does modern code review impact software design degradation? an in-depth empirical study. In: 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp 511–522, DOI 10.1109/ICSME46990.2020.00055

[34] Zanaty FE, Hirao T, McIntosh S, Ihara A, Matsumoto K (2018) An empirical study of design discussions in code review. In: Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, Association for Computing Machinery, New York, NY, USA, ESEM '18, DOI 10.1145/3239235.3239525, URL https://doi.org/10.1145/3239235.3239525