Transferring Well-Trained Models for Cross-Project Issue Classification: A Large-Scale Empirical Study

Yue Yu
Laboratory of Software Engineering for Complex Systems,
National University of Defense Technology
Changsha, Hunan, China
yuyue@nudt.edu.cn

Yarong Zeng
College of Computer,
National University of Defense Technology
Changsha, Hunan, China
zengyarong16@nudt.edu.cn

Qiang Fan
College of Computer,
National University of Defense Technology
Changsha, Hunan, China
fanqiang09@nudt.edu.cn

Huaimin Wang
College of Computer,
National University of Defense Technology
Changsha, Hunan, China
hmwang@nudt.edu.cn

ABSTRACT

In modern software engineering practices, various kinds of automated and intelligent methodologies have been proposed to improve the efficiency of collaborative development. However, most of those approaches are heavily dependent on supervised or semi-supervised learning technologies, which would be restricted by the lack of training data. Inspired by the theories and techniques of transfer learning, cross-project approaches have been proposed, but hard to achieve a consistent and desirable performances. In this paper, we conduct an extensive empirical study to capture the determinants that affect the performances of transferring reusable models across projects in the context of issue classification. Starting from a large-scale dataset, containing 799 OSS projects and more than 795,000 issues, we have extracted 28 attributes grouped into 4 different dimensions. The results show that the performance of cross-project issue classification based on model transferring is sensitive and unstable, which is influenced by multiple factors spreading among transferred model training, project construction, and technical and socail relations between source and target.

CCS CONCEPTS

• **Software and its engineering** → *Software maintenance*;

KEYWORDS

Cross-Project, Transfer Learning, Issue Classification;

ACM Reference Format:
Yue Yu, Yarong Zeng, Qiang Fan, and Huaimin Wang. 2018. Transferring Well-Trained Models for Cross-Project Issue Classification: A Large-Scale Empirical Study. In *The Tenth Asia-Pacific Symposium on Internetware (Internetware '18), September 16, 2018, Beijing, China.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3275219.3275237

Internetware '18, September 16, 2018, Beijing, China

ACM ISBN 978-1-4503-6590-1/18/09…$15.00
https://doi.org/10.1145/3275219.3275237

1 INTRODUCTION

With the development of distributed software development, various kinds of automated and intelligent methodologies based on supervised or semi-supervised learning technologies have been proposed, to reduce the burden of project management, such as recommender systems for locating the qualified bug fixers [1, 8] or code reviewers [17, 23, 24], and automated management approaches for detecting duplicate issues [13, 20, 21]. However, a comprehensive and high-quality dataset for those training-based approaches is often not available, which restricts their performances in practise. Inspired by the theory and technique of transfer learning [14], cross-project approaches have been investigated in two aspects, *i.e.*, data transferring and model transferring. In terms of data-reusing, the main idea of transferring [11, 12, 19] is to learn transformation fuctions that mapping sufficient data from source projects (*i.e.*, the well-developed projects with plenty of historical data) onto a uniform distribution of target projects; then a intelligent model can be trained by using the selected data or features. Although the transformations vary from individual to individual, *i.e.*, run the training process for every target project according to the data characteristics, it is extremely hard to estimate the real distribution from the limited amount of samples in the target projects. Hence, the transfer models are hard to achieve a consistent and desirable performance, unless the models are frequently retrained or updated by continuously adding newly received data of the target project. Compared to data transferring with high learning cost, other researches [9, 10, 28] are tried to directly reuse (*i.e.*, transfer) well-trained models learned from source projects in targeted projects. However, those studies, restricted to relative small or median scale data sets, only focus on transferring **one single model**, *e.g.*, using the defect prediction model trained from Firefox in Internet Explorer [28], which leads to discouraging and inconsistent results [16].

In this paper, we conduct an extensive empirical study of transferring reusable models across projects in the context of issue classification [5, 7, 26, 27] (*i.e.*, distinguishing real bugs from nonbugs among all issues). Thus, we develop the main research question: **RQ:**

What factors affect the performances of transferring well-trained models for cross-project issue classification? Starting from the state-of-the-art classification approach [5], we investigate how the software project characteristics (measured by 28 metrics extracted from four different dimensions) affects the performance of model transferring. Although our study scenarios is different from the most of existing cross-project work about defect prediction [9–12, 19, 28], we argue that most of investigated factors (as discussed in Section 2.2) are widely discussed in various contexts, *e.g.*, contribution evluation [18, 22] and defect prediction [15, 28], so the empirical results of this paper probably can be extendted to other application scenarios.

In particular, the key contributions of this paper include:

- We collect a comprehensive dataset of 232,200 transferring pairs generated from 799 projects in GitHub.
- We design a comprehensive regression model containing 28 widely used softawre metrics (as shown in Table 1), which could reveal the in-depth correlations among source and target projects.
- We claim that the performance of cross-project issue classification based on model transferring is sensitive and unstable, which is influenced by multiple factors of the relations between source and target projects, in terms of model training, project construction, social and technical aspects.

2 METHODS

2.1 Two-stage Issue Classification

In the modern issue tracking system (*e.g.*, GitHub Issue 2.0), developers are only required a short textual abstract to submit a new issue report. There are many types of issues in the IST, including bug, feature request, enhancement, documentation, and so on. For each received issue, the core team needs to understand the contributor's purpose, identify the type and filter out the duplicate and undesirable one. To reduce the human cost and hasten the management process, automated and intelligent categorization approaches based on text mining [5] has been receiving increasing attention from the research community. In this paper, we use the two-stage classification model (state-of-the-art [5]) to build the issue classifiers within projects. In the first stage, we use free text of issue report (*i.e.*, title and description) to train the SVM classifier and get semantic probability and perplexity. In the second stage, we combine the structured developer imformation and the outputs from the first stage to train the final prediction model by using logistic regression.

2.2 Regression Analysis

We aims to determine the factors that influence the performances of cross-project classification. Supposing that there is a project set C, for P_i, P_j $(i \neq j)$ in C, we consider a combination (P_i, P_j) to train a classification model using P_i, and test how well it works in P_j. Then, we collect a set of measures between cross-project pairs which can represent the association relationships between the source projects and target projects. Finally, we use multiple linear regression to analyze the factors that affect the performance of cross-project classification. Our models are fitted using the *lme4*[1] package in R. The outcome of the regression model is the performance of cross-project classification, the predictors are those measures between cross-project pairs.

In regression analysis, all numeric variables are first log transformed (plus 0.5) to stabilize variance and reduce heteroscedasticity [4]. We compute the variance inflation factors (*VIFs*) for each predictor to check whether there exist multicollinearity problems or not (all remained well below 5, indicating absence of multicollinearity) [4]. For each model predictor, we report its coefficients, standard error, and significance level. We consider coefficients important if they are statistically significant ($p < 0.05$). In addition to the coefficients, we get effect sizes (*Sum square*) of each predictor from *ANOVA* analyses. The goodness-of-fit of regression models can be evaluated by pseudo R-squareds, *i.e.*, conditional R^2 coefficient of determination for generalized mixed-effects models (*i.e.*, the higher R^2 value is, the better the fitting effect is), which is implemented in the *MuMIn* [2] package of R.

2.3 Predictors

Our predictors of regression analysis cover training model-level, project-level, social-level, and technical-level.

2.3.1 *Model-level measures*.

Model issue num: The number of the training samples of the classification model. As more training samples are available, the model corpus is richer, then the generalization performance of the model may be better.

Local model performance: The classification performance of the training project on their own test data sets. We use f_{avg}(see Equation 1) to evaluate the model performance to consist with cross-project effect validation.

Model issue Gini coefficient: A measure of statistical dispersion, which intends to represent the issue distribution of a project's contributors. The value zero indicates that all values are the same, namely all contributors of a project submit the same number of issues. The value one represents the maximal inequality among values, when one person submits almost all the issues, and there are only a few other people, the Gini coefficient will be very close to one. We suspect that the degree of confusion in the issue distribution may have an impact on the cross-project classification performance.

2.3.2 *Project-level measures*.

Project popularity: The popularity of the project. We use the number of *Watch* to measure the popularity of project in GitHub.

Project age: The time from project creation to the current time in months, which is used as a proxy for maturity.

Same Owner Type: Whether the owner type of the two projects are the same or not. Owner type in GitHub can be divided into two types, Organization and User. Organizations are shared accounts where businesses and open-source projects can collaborate across many projects at once [6]. Users indicate a personal account.

Same License: Whether the open source license of the two projects are the same or not. There are some popular licenses, *e.g.*, *MIT*, *GPL*, *Apache-2.0*.

[1] https://cran.r-project.org/web/packages/lme4/lme4.pdf
[2] https://cran.r-project.org/web/packages/MuMIn/MuMIn.pdf

Same Language: Whether the programming language of the two projects are the same or not. A project may have a variety of development languages. We only consider the first language which covers the largest amount of code.

Similarity of Readme File: The similarity of README file between two projects, We remove the markdown tags of readme file and extracted text information. Then use Term frequency-inverse document frequency(*TF-IDF*) technique to convert the raw text to a features vector, and we calculate cosine similarity between the two vectors.

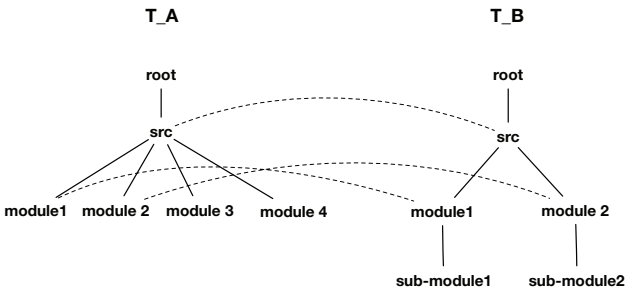

Figure 1: The directory structure of two projects

2.3.3 ***Social-level measures***.

Core members: The number of core members in project. The core members refer to someone who have extensive permissions, include the repository owner and collaborators in GitHub.

External contributors: The number of external contributors who have contributed to the project. The common contribution activities include contribute code, submit issue reports, or just report comment.

Core members difference: The difference in the number of project's core member.

External contributors difference: The difference in the number of project's external contributors.

Participants intersection: The intersections in the number of project's participants(include core member and external contributors). The same group has contributed to two projects, the development activity data they generated will be very similar, and the correlation between projects is stronger.

Similarity of National distribution for participants: The similarity of national distribution of all participants between two projects. We use unidimensional vector to represent the national distribution of a project, the vector length is the number of countries, and the number of participants in each country of the project is represented by the value of the status bit. We extract the participants information of projects in GitHub from Ghtorrent[3] *user* table. For participants with incomplete nationality information, such as have *city* and *location* but no *state* information, we use state and city relations table provided by *Maxmind* [4] to complement the nationality information.

[3]http://www.ghtorrent.org

[4]https://www.maxmind.com/en/free-world-cities-database

2.3.4 ***Technical-level measures***.

Code size: The code size of project, counted by bytes.

Code size difference: The code size difference between two projects.

Code dependency: The number of dependencies(code library or package) of proejct. We extracted project dependencies which hosted in GitHub through libraries.io[5], and complemented dependency through parsing GitHub dependency graph web pages.

Code commen dependency: The number of common dependencies between two projects.

Code dependency difference: The difference in the number of code dependencies between two projects.

Similarity of Project structure: The similarity of directory structure between two projects. First, We use JSON tree to portray the project directory structure, then calculate the edit distance between trees using the algorithm proposed by Zhang et al [25]. The smaller the distance, the more similar the projects structure. The tree edit distance is defined as follows. An edit script S between two trees $T1$ and $T2$ is a sequence of edit operations turning $T1$ into $T2$, the cost of S is the sum of the costs of the operations in S. The minimum cost between $T1$ and $T2$ called the tree edit distance [2]. Figure 1 gives an example. Trees T_A and T_B represent the directory structure of project A and project B, respectively. The figure shows the way to turn T_A into T_B, it corresponds to the sequence (delete (node with module 3,module 4), insert (node with sub-module1,sub-module2)). Thus, the tree edit distance between these two projects is equal to 4.

Comment cross-references: The number of direct links between two projects which appear in issue comments. Cross-references to other projects appearing in GitHub comments indicate the existence of a technical dependency [3].

3 EXPERIMENT SETUP

3.1 Dataset

By using the 2017-05 GHTorrent [6] dump, we filter out the 12,797 projects that have at least 100 issues in total. Then, we do a further data cleaning by removing: 1) forked and pure documentation projects; 2) the projects using non-Engilish to report issues to avoids the bias of language deviation; 3) projects have less than 500 labeled issues to guarantees that the classification models have enough training and testing data.

Finally, we collect issues data for each project through GitHub public API, and get a relative desired dataset contains 779 projects, and 795,284 issues. The smallest project has 502 issues and the largest project has 13,793 issues in total. The mean number of issues per project is 1020 and the median is 672.

3.2 Experimental Design

In order to explore what can make cross-project issue classification work, we run a large-scale experiment based on 779 projects. Firstly, we separate our dataset into two groups. One group is a set of projects S ($|S| = 465$) which have sufficient issues data (*i.e.*, at least 500 issues received before 2016-10-01) to train a good classification model within the project. The other group is a set of projects T ($|T| = 500$) which have at least 50 issues received after 2016-10-01,

[5]https://libraries.io

[6]http://ghtorrent.org/downloads.html

Table 1: Summary of measures for our dataset

Levels	Statistics	Mean	St. Dev.	Min	Median	Max
Model-level	model_issue_num	1037	724.61	500	775	5523
	model_issue_gini	0.67	0.18	0	0.72	0.96
	local_model_acc	0.80	0.05	0.61	0.81	0.95
Project-level	prjA_popularity	255.1	374.26	3	122	2777
	prjB_popularity	247.4	363.35	2	117	2777
	prjA_age	49.56	19.61	9	47	103
	prjB_age	43.26	19.1	4	41	103
	same_owner_type	0.64	0.48	0	1	1
	same_license	0.18	0.50	0	0	1
	readme_sim	0.08	0.05	0	0.07	0.75
	same_language	0.13	0.34	0	0	1
Social-level	prjA_core_num	9.827	12.04	1	7	142
	prjB_core_num	9.62	11.51	1	6	142
	prjA_external_num	994.1	1362.54	2	426	7888
	prjB_external_num	944.2	1293.99	1	425	7888
	abs(core_num_diff)	9.25	13.87	0	5	141
	abs(external_num_diff)	1215	1436.1	0	681	7887
	participant_intersection	3.35	13.52	0	2	1625
	national_distribution_sim	0.6489	0.30	0.00	0.76	1
Technical-level	prjA_code_size	7186000	9435694	4177	3814000	63500000
	prjB_code_size	6867000	8689209	4177	3771000	65470000
	prjA_code_depen	68.08	149.70	0	7	1404
	prjB_code_depen	84.37	170.79	0	12	1404
	abs(codesize_diff)	7890000	10125680	33	4263000	65460000
	abs(code_depen_diff)	121.1	193.02	0	38	1404
	code_commen_depen	2.73	20.70	0	0	795
	project_structure_sim	538.2	667.38	0	293	4558
	comment_cross_reference	0.03	1.37	0	0	241

to verify the performance of the cross classifiers. For $A \in S$ and $B \in T$, we consider a cross-project pair (A, B) to train a model from A to predict B, if and only if $A \neq B$. Thus we can get 232, 200 combinations based on our dataset. Figure 2 shows the classification performance of one-to-one cross-project in these combinations, as well as the corresponding classification performance within the project. From the classification results, we can infer that there is still a gap between the one-to-one cross-project and within-project. This also confirmed the findings of previous study [28] that directly transferring one single model leads to discouraging and inconsistent results. Next, we collect measures between these cross-project pairs according to the method described in Section 2.3. The summary of our measures are shown in Table 1, where the prefix *prjA* indicates the source project, and the *prjB* indicates the target project. Finally, we use multiple linear mixed effect models to investigate the determinants that affect cross-project classification performance as mentioned in Section 2.2.

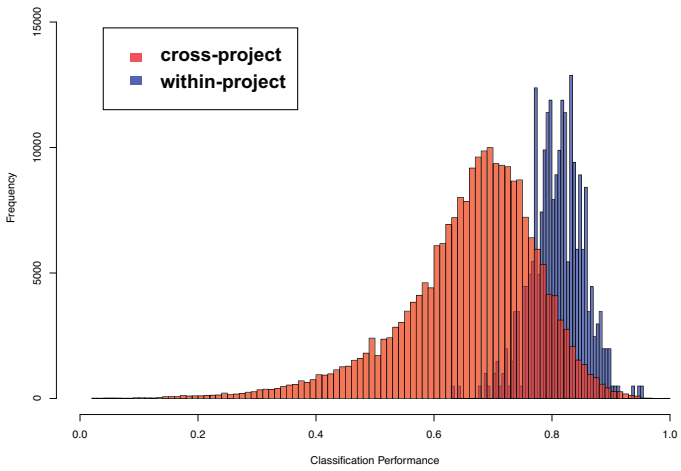

Figure 2: Distribution of classification performance of one-to-one cross-project and within-project

4 RESULTS

Table 2 shows the result of regression analysis. The regression model achieves a good fit (R^2 = 49.27%). In our model, the *VIFs* of all the predictors are well below three, thereby indicating the absence of multicollinearity [4]. From Table 2, we find the performance of cross-project classification is affected by the measures from four dimensions. Next, we discuss the effects of these measures.

All model-level measures are highly significant and positive for the performance of cross-project classification. Increasing the

Table 2: Regression Analysis of Cross-project Transferability

	Coeffs(Errors)	Sum Sq
(Intercept)	0.3256(7.37)***	
log(model_issue_num)	2.188(0.707) **	689.7 **
log(model_issue_gini + 0.5)	12.37(2.408) ***	2254.1 ***
log(local_model_acc)	21.31(5.370) ***	1345.5 ***
log(prjA_popularity)	-0.191(0.534)	11.0
log(prjB_popularity)	0.302(0.386)	52.6
log(prjA_age)	0.981(0.933)	94.6
log(prjB_age)	0.394(0.536)	46.4
same_owner_type TRUE	0.318(0.057) ***	2592.2 ***
same_license TRUE	0.440(0.080) ***	2580.0 ***
log(readme_sim + 0.5)	2.172(0.403) ***	2482.1 ***
same_language TRUE	0.727(0.067) ***	9925.7 ***
log(prjA_core_num)	0.239(0.441)	25.3
log(prjB_core_num)	-0.041(0.311)	1.5
log(prjA_external_num)	0.691(0.473)	182.8
log(prjB_external_num)	-0.505(0.328)	202.5
log(abs(core_num_diff) + 0.5)	0.087(0.026) **	892.6 **
log(abs(external_num_diff) + 0.5)	-0.209(0.019) ***	9509.9 ***
log(participant_intersection + 0.5)	0.701(0.052) ***	15354.8 ***
log(national_distribution_sim)	1.912(0.471) ***	1406.7 ***
log(prjA_code_size)	0.268(0.261)	89.9
log(prjB_code_size)	0.656(0.214) **	800.2 **
log(prjA_code_depen)	0.060(0.143)	15.2
log(prjB_code_depen)	0.087(0.106)	59.0
log(abs(codesize_diff))	-0.182(0.020) ***	6973.7 ***
log(abs(code_depen_diff) + 0.5)	-0.064(0.018) ***	991.7 ***
(code_commen_depen > 0) TRUE	0.164(0.087) .	305.5 .
log(project_structure_sim + 0.5)	-0.128(0.038) ***	966.0 ***
(comment_cross_reference > 0) TRUE	0.579(0.293) *	333.4 *
Conditional R-squared	0.4927	

signif.: p < 0.001 '***', p < 0.01 '**', p < 0.05 '*'

number of training samples of source project can improve the classification effect. And if the model is good at categorizing its own issue data, it can also perform well for other projects. Perhaps unexpectedly, we find the greater the difference between the number of issues reported by each contributor, the model can classify better on target project.

For project-level measures, the *same_language* is a strong predictor for cross-project classification, explaining 16% of the variance explained (calculated as: the percentage of *SumSq*. of above variables accounting for the whole *SumSq*. of regression model). Because the two projects have same grammatical structure and definition when having same language, we infer that their issue texts are similar. Thus the cross-project issue classifier can perform well. The same goes for *readme_sim*. The *prj_age* and *prj_popularity* are not significant in the model. Therefore, we can infer that the maturity of the project and its popularity have no effect on cross-project classification. In addition, some obvious measures, such as *same_license*, *same_owner_type* are significant, which means that if two projects have same license,owner type or similar project structure, the performance of cross-project classification will be better.

For social-level measures, we confirm that social variables play an importance role on the success of cross-project issue classification, as all measures of this level together account for 43.5% of the variance explained. Among them, the *participant_intersection* has the greatest impact on cross-project classification over the other variables in our model. We argue that the development activity data (including issue data) generated by same participants are very similar. Thus, for projects which have big *participant_intersection* with the target project tends to perform well because of the similar issue structure. With the difference in the number of external contributors between projects increase, the performance of cross-project classification will decrease. But this is the opposite for core member, albeit with little influence. Moreover, The similarity of national distribution of participants between two projects has a positive effect on the classification performance. We infer that people in different nationals tend to have different ways of expression.

There are three technical-level measures that exert significant and negative effects on cross-project transferability. If there are great differences in the code size and number of code dependencies between projects, the cross-project classification result will be bad. If two projects have similar organizational stucture, the performance of cross-project classification will be better. One thing to note is the similarity of project structure (*project_structure_sim*) in Table 2 is negative, that is because we use the edit distance [25] to test the structural similarity in our model, the smaller the distance, the more similar the structure. Additionally, whether there exist common code dependencies or cross reference between projects has small significant effect on cross-project classification. All variables in this level together account for 17.17% of the variance explained.

5 CONCLUSION AND FUTURE WORK

In this paper, we investigate the performances of transferring well-trained models from source projects to target projects in the context of issue classification. Regression analysis is used to identify determinants that affect the performance of cross-project issue classification. We find many factors have significant influence on the success of cross-project issue classification, among which the intersection of participants is the dominant one.

In future, we plan to design a novel cross-project approach based on our empirical results. In addition, we will investigate the effectiveness of our cross-project approach in different software engineering scenarios, *e.g.*, defect prediction, developer recommendation and effort estimation.

ACKNOWLEDGMENT

This research is supported by National Science Foundation of China (Grant No.61702534, 61432020, 61472430 and 61502512) and National Key R&D Program of China (2016-YFB1000805).

REFERENCES

[1] John Anvik, Lyndon Hiew, and Gail C Murphy. 2006. Who should fix this bug?. In *Proceedings of the 28th international conference on Software engineering*. ACM, 361–370.

[2] Philip Bille. 2005. A survey on tree edit distance and related problems. *Theoretical computer science* 337, 1 (2005), 217–239.

[3] Kelly Blincoe, Francis Harrison, and Daniela Damian. 2015. Ecosystems in GitHub and a method for ecosystem identification using reference coupling. In

Proceedings of the 12th Working Conference on Mining Software Repositories. IEEE Press, 202–207.

[4] Jacob Cohen, Patricia Cohen, Stephen G West, and Leona S Aiken. 2013. *Applied multiple regression/correlation analysis for the behavioral sciences*. Routledge.

[5] Qiang Fan, Yue Yu, Gang Yin, Tao Wang, and Huaimin Wang. 2017. Where Is the Road for Issue Reports Classification Based on Text Mining?. In *Empirical Software Engineering and Measurement (ESEM), 2017 ACM/IEEE International Symposium on*. IEEE, 121–130.

[6] Github Help. 2018. About organizations. https://help.github.com/articles/about-organizations/. (2018).

[7] Kim Herzig, Sascha Just, and Andreas Zeller. 2013. It's not a bug, it's a feature: how misclassification impacts bug prediction. In *Proceedings of the 2013 international conference on software engineering*. IEEE Press, 392–401.

[8] Gaeul Jeong, Sunghun Kim, and Thomas Zimmermann. 2009. Improving bug triage with bug tossing graphs. In *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*. ACM, 111–120.

[9] Rahul Krishna and Tim Menzies. 2017. Simpler Transfer Learning (Using" Bellwethers"). *arXiv preprint arXiv:1703.06218* (2017).

[10] Rahul Krishna, Tim Menzies, and Wei Fu. 2016. Too much automation? The bellwether effect and its implications for transfer learning. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*. ACM, 122–131.

[11] Ying Ma, Guangchun Luo, Xue Zeng, and Aiguo Chen. 2012. Transfer learning for cross-company software defect prediction. *Information and Software Technology* 54, 3 (2012), 248–256.

[12] Jaechang Nam, Sinno Jialin Pan, and Sunghun Kim. 2013. Transfer defect learning. In *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 382–391.

[13] Anh Tuan Nguyen, Tung Thanh Nguyen, Tien N Nguyen, David Lo, and Chengnian Sun. 2012. Duplicate bug report detection with a combination of information retrieval and topic modeling. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*. ACM, 70–79.

[14] Sinno Jialin Pan and Qiang Yang. 2010. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering* 22, 10 (2010), 1345–1359.

[15] Foyzur Rahman and Premkumar Devanbu. 2013. How, and why, process metrics are better. In *Software Engineering (ICSE), 2013 35th International Conference on*. IEEE, 432–441.

[16] Foyzur Rahman, Daryl Posnett, and Premkumar Devanbu. 2012. Recalling the imprecision of cross-project defect prediction. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. ACM, 61.

[17] Patanamon Thongtanunam, Chakkrit Tantithamthavorn, Raula Gaikovina Kula, Norihiro Yoshida, Hajimu Iida, and Ken-ichi Matsumoto. 2015. Who should review my code? A file location-based code-reviewer recommendation approach for modern code review. In *Software Analysis, Evolution and Reengineering (SANER), 2015 IEEE 22nd International Conference on*. IEEE, 141–150.

[18] Jason Tsay, Laura Dabbish, and James Herbsleb. 2014. Influence of social and technical factors for evaluating contribution in GitHub. In *Proceedings of the 36th international conference on Software engineering*. ACM, 356–366.

[19] Burak Turhan, Tim Menzies, Ayşe B Bener, and Justin Di Stefano. 2009. On the relative value of cross-company and within-company data for defect prediction. *Empirical Software Engineering* 14, 5 (2009), 540–578.

[20] Xiaoyin Wang, Lu Zhang, Tao Xie, John Anvik, and Jiasu Sun. 2008. An approach to detecting duplicate bug reports using natural language and execution information. In *Software Engineering, 2008. ICSE'08. ACM/IEEE 30th International Conference on*. IEEE, 461–470.

[21] Yue Yu, Zhixing Li, Gang Yin, Tao Wang, and Huaimin Wang. 2018. A dataset of duplicate pull-requests in github. In *Proceedings of the 15th International Conference on Mining Software Repositories*. ACM, 22–25.

[22] Yue Yu, Huaimin Wang, Vladimir Filkov, Premkumar Devanbu, and Bogdan Vasilescu. 2015. Wait for it: Determinants of pull request evaluation latency on GitHub. In *Mining software repositories (MSR), 2015 IEEE/ACM 12th working conference on*. IEEE, 367–371.

[23] Yue Yu, Huaimin Wang, Gang Yin, and Charles X Ling. 2014. Reviewer recommender of pull-requests in GitHub. In *IEEE International Conference on Software Maintenance and Evolution*. IEEE, 609–612.

[24] Yue Yu, Huaimin Wang, Gang Yin, and Tao Wang. 2016. Reviewer recommendation for pull-requests in GitHub: What can we learn from code review and bug assignment? *Information and Software Technology* 74 (2016), 204–218.

[25] Kaizhong Zhang and Dennis Shasha. 1989. Simple fast algorithms for the editing distance between trees and related problems. *SIAM journal on computing* 18, 6 (1989), 1245–1262.

[26] Y. Zhou, Y. Tong, R. Gu, and H. Gall. 2014. Combining Text Mining and Data Mining for Bug Report Classification. In *2014 IEEE International Conference on Software Maintenance and Evolution*. IEEE Press, 311–320.

[27] Yu Zhou, Yanxiang Tong, Ruihang Gu, and Harald Gall. 2016. Combining text mining and data mining for bug report classification. *Journal of Software: Evolution and Process* 28, 3 (2016), 150–176.

[28] Thomas Zimmermann, Nachiappan Nagappan, Harald Gall, Emanuel Giger, and Brendan Murphy. 2009. Cross-project defect prediction: a large scale experiment on data vs. domain vs. process. In *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*. ACM, 91–100.