Assessing Personalized Software Defect Predictors

Beyza Eken
Faculty of Computer and Informatics Engineering
İstanbul Technical University
İstanbul, Turkey
beyzaeken@itu.edu.tr

ABSTRACT

Software defect prediction models guide developers and testers to identify defect prone software modules in fewer time and effort, compared to manual inspections of the source code. The state-of-the-art predictors on publicly available software engineering data could catch around 70% of the defects. While early studies mostly utilize static code properties of the software, recent studies incorporate the people factor into the prediction models , such as the number of developers that touched a code unit, the experience of the developer, and interaction and cognitive behaviors of developers. Those information could give a stronger clue about the defect-prone parts because they could explain defect injection patterns in software development. Personalization has been emerging in many other systems such as social platforms, web search engines such that people get customized recommendations based on their actions, profiles and interest. Following this point of view, customization in defect prediction with respect to each developer would increase predictions' accuracy and usefulness than traditional, general models. In this thesis, we focus on building a personalized defect prediction framework that gives instant feedback to the developer at change level, based on historical defect and change data. Our preliminary analysis of the personalized prediction models of 121 developers in six open source projects indicate that, a personalized approach is not always the best model when compared to general models built for six projects. Other factors such as project characteristics, developer's historical data, the context and frequency of contributions, and/or development methodologies might affect which model to consider in practice. Eventually, this topic is open to improvement with further empirical studies on each of these factors.

CCS CONCEPTS

• **Software and its engineering → Software defect analysis**;

KEYWORDS

personalized defect prediction, bug prediction, customization

ACM Reference Format:
Beyza Eken. 2018. Assessing Personalized Software Defect Predictors. In *ICSE '18 Companion: 40th International Conference on Software Engineering , May 27-June 3, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3183440.3183449

1 PROBLEM STATEMENT AND RESEARCH MOTIVATION

Software defect prediction (SDP) research has evolved over more than two decades in empirical software engineering through a variety of case studies, algorithm-oriented and data-oriented approaches. There are systematic literature reviews [8, 12, 18] that synthesize the current evidence of automated, learning-based defect prediction models reported in the literature from the aspect of metrics, algorithms and methods. Studies show that SDP models support developers during implementation, unit testing and maintenance phases by reducing their time and effort spent during bug detection. These models are reported as robust, i.e., quicker to build and catch bugs compared to inspection activities carried out by development teams, and successful at detecting around 70% of defects [13].

A variety of machine learning algorithms has been utilized in SDP field. To analyze the effectiveness of the algorithms on prediction performance, Lessmann et al. [11] report a meta-analysis on 10 publicly available software projects and 22 algorithms. Their analyses indicate that the top performing 17 algorithms are not significantly different in predicting defects. Later, Fu et al. [7] chose two of these 22 algorithms and showed that the tuning of algorithms may improve the predictor's precision performance from 0% to 60%. Menzies et al. [14] also reported a ceiling effect on the defect prediction performance when it is built using static code attributes only. Hence, these analyses lead us focus on enriching the information content of data used to train such models.

Recent trends on software defect prediction tend to capture process and people related features in some ways. Bell et al. [2] simply used the *number of developers* that worked on a code unit to predict defects. Di Nucci et al. [6] incorporated *developer focus* into prediction models based on the closeness and the similarity between code units modified by a developer. Lee et al. [10] also proposed *interaction behavior at micro level* based on browsing and editing times, frequencies and edited file information. Calikli et al. [5] modelled developers in terms of their *confirmation bias levels* to identify their relationships with the post-release defects.

The aforementioned studies do not propose customized models for different developers; but propose a common metric set that would represent people through developer characteristics. The studies, on the other hand, also discuss that fault proneness of individual programmers with respect to the system characteristics vary significantly [17]. Hence, a more focused analysis should be done at developer level to understand the reasons behind the faults injected by each developer.

2 RELATED STUDIES

Betterburg et al.[3] argue that software engineering data contains a large amount of variability, and models built with a global context are often irrelevant and less successful than models built with a local context. Following this, defect prediction research has evolved towards models built with a developer context. We found two studies [9, 22] that propose personalized models for each of the selected set of developers in open source projects based on change data. The analysis by Jiang et al. [9] indicate that, when the top 20% of defective lines of code are inspected, a personalized approach could detect up to 155 more defects than a traditional approach. The F1-score of the personalized model was 1 to 6% more than the traditional (general) model among six projects. Xia et al. [22] improves the performance of the previous personalized approach in [9] by enriching personal change data with data from other developers' changes using genetic algorithms. The results reveal that their proposed solution could detect up to 245 more defects than the previously built model.

3 PROPOSED RESEARCH

Personalization of recommendations and/or their rankings has been actively studied in many other businesses to reveal more satisfying results, i.e., search engines and social media platforms utilize users' features such as search history or locations to give better recommendations or advertisements [4, 21]. Following the advancements in other fields towards personalization, we argue that human characteristics and their development patterns are substantial for building better recommendation systems in software engineering. As one of those systems, defect prediction models should also employ developer-specific features to increase the accuracy, usefulness by considering cost-benefit ratio and/or novelty of recommendations [1]. Defects injected by developers of software teams may reveal distinguished patterns [17], because each developer has unique characteristics regarding their coding styles, the defect proneness of the code they produce, their programming experience, and development methodologies employed. Considering developers' own features and defect history, defect predictors should provide customized and instant feedback to each developer. Developer behaviours leading different development patterns could change over time. Consequently, defect predictors adapt to these changing patterns by including them into the historical data.

We propose building a personalized SDP approach through separate models trained for each developer based on his/her associated development history. A personalized model essentially differs from a traditional, general model in terms of data used for training, and the target audience. A general model utilizes all developers' change history, and its output is available to everyone in the team, whereas in a personalized model, a subset of change data associated with the selected developer is used, and predictions are made for that developer only. This way, we also filter irrelevant feedback to the developer.

To clarify our research proposal, we define the following research questions in this thesis:

- RQ1: How we can build a personalized SDP model?
- RQ2: What are the constraints that affect the performance and applicability of personalized SDP models?
- RQ3: How can a personalized SDP approach adapt to a newcomer?

Our research methodology consists of conducting case studies on projects from different contexts, since contextual factors such as application domain, size and maturity of the project may effect the performance of defect predictors [8]. We will conduct empirical analyses on commercial and open source projects which vary in terms of team structure, team size and development methodology. We plan to compare a personalized approach with a general model, and investigate the effect of project and developer characteristics on the model's performances. For example, large-scale projects may have more contributors although only a few have the role of committing a fixing change. In such a case, general model might perform better than a personalized approach. On the other hand, when a new developer joins the team, utilizing data from a subset of developers based on similar profiles might generate more useful and accurate predictions than data collected from the whole team. In other cases when the team is homogeneous, or there are groups of developers with similar characteristics, or personalized data is too small, both models could contribute to each other through a mixed, weighted approach. All of these aspects will be considered and investigated throughout this research.

We aim to contribute to the state-of-the-art study [9] by taking into account the contextual information of software projects, extending performance evaluation techniques, clustering developers with respect to the developer profiles and solving the problem of a newcomer. We will conduct empirical analyses on the effect of project and team structure on the performance of personalized approach. We intend to assess the model performance with an enriched set of measures, e.g. false alarms, recall and cost-benefit analysis. Also, there is no proposed personalized approach for the developers who recently joined to the team, or who have little contribution to the project. We plan to fill this gap.

4 PRELIMINARY RESULTS AND DISCUSSION

To answer our RQ1, we conceptually replicated Jiang et al.'s approach [9] to build a baseline for our personalized SDP framework. The authors proposed personalized models for a set of developers within six projects, and compared them with a general model and a weighted combination of both. The selection criterion of developers to build SDP models is identified as top 10 developers with the maximum number of commits in each project. Personalized models of the selected 10 developers are built on 100 consecutive commits of the developers, whereas the general model is built on the aggregated total of 1000 commits of the 10 selected developers. The authors evaluated their models using F1 and NofB20 measures. NofB20 stands for the number of bugs detected when the top 20% of the lines of code is inspected [9].

The work of Jiang et al. [9] show that a personalized SDP model is more successful than a general one, although there were assumptions made by the authors on preparation of dataset and model construction, e.g. choosing only the most contributing 10 developers, 100 commits for training the model. There are also unstated parts, e.g. buggy commit ratio for each developer in data, false alarm rates of the models. These missing information lead us to further investigate the intrinsic features of personalized models. Therefore, we built a total of 121 personalized SDP models and six general models for six open source projects [15] to evaluate if the personalized approach is generalizable across different environments with different set of metrics, and to examine if this new approach is as successful as stated in [9].

In each of the six projects, we determined the list of developers who have contributed the most based on their number of commits within each project. We intended to include the developers who own the majority of the code changes in version control systems into our SDP models. This selection idea based on Pareto principle which implies that generally 80% of the development made by 20% of the software team [23]. Once we ranked the developers according to their number of commits from the most to the least, we selected the developers from the ranked list whose contributions constituted of 80% of the total commits. Note that the number of developers selected based on this criteria are different among six projects. For example, we chose 22 developers from Gimp project, whereas we chose 2 developers from Maven-2 project.

Three different SDP models are reported on the selected developers and projects in this paper. First, a personalized model (PM) is built for each of the selected developers based on the developer's own commit data, within each project. Second, a selected personalized model (SM) is built, combining the training data from all selected developers. We built this model to make a comparison between our SM and the general model in the study we replicated, because the general model of Jiang et al. [9] corresponds to our SM. Third, we built a general model (GM) that includes all commit data in the project combining all selected and non selected developers' changes.

Two different versions of PM and SM are built based on data sampling criteria we applied. We prepared the training sets for PMs in two ways: a) randomly selecting N commits for a developer, where N specifies the minimum number of commits by the selected developers in a project, and b) selecting all commits corresponding to a developer. For instance, the first approach randomly picks 132 commits from each of the selected 22 developers' commit data in Gimp project, since there are at least 132 commits made by the 22 developers. Training set for SM are prepared by combining the commit data from all the selected developers: a) N commits x k selected developer and b) $\sum_{i=1}^{k} N_k$ where N_k is the number of commits for developer i. Training set for GM includes all commit data in the project, combining all selected and non selected developers' changes without applying any sampling criteria.

We chose two different machine learning algorithms, namely Naive Bayes (NB) and Random Forests (RF), which are well performed algorithms in SDP task [11, 12]. We evaluate the models' performances in terms of probability of detection (pd), probability of false alarms (pf), precision, and F1-measure [8]. Nemenyi test [16] is applied to examine if the results are statistically significant.

Preliminary findings of our three models are presented in Table 1. We only report the findings using RF algorithm and the second approach for training data preparation, i.e., choosing all commits of a developer while building PM and SM, because it gives the best performance over all projects and developers. The number of selected developers in each project is given in parenthesis next to the project name. Data size column indicates the size of the data used to evaluate the model after 10x10-fold cross validation is applied on the models. PM results in the table are the average values over all PMs of a project. It can be seen that, all three models perform very similar to each other.

Apart from average findings on PMs in Table 1, individual PMs perform differently for each developer. One of the best performances of PM achieves 78% pd and 16% pf rates for the Developer-4 in the Gimp project. Beside this, some distinguished examples are as follows: For the Developer-2 in Gimp, PM increases pd from 12% to 43%, while pf remains the same for NB method compared to SM. On the other hand, there is not a significant difference between PM, SM and GM when applied RF with sampled commits. For the Developer-2 in PostgreSQL, PM decreases the pf from 58% to 7%, while pd remains the same for NB with sampled commits, when compared to SM. The difference between PM and GM, on the other hand, is not significant. For the Developer-22 in Gimp, NB increases both pd and pf in SM and GM models.

In Table 2, we also report the pairwise comparisons between the models in terms of the number of times a PM model performs better, worse or equal to the SM and GM models. In terms of pd, PMs are better in 49 out of 121 developers' cases only compared to SMs. PMs also produce higher pf values than SMs. F1-measure of SMs is also higher than that of PMs in 59 out of 121 developers cases. The same pattern also holds for PM-GM comparisons. Regarding the comparison between SM and GM, it is found that both pd, pf and F1-measure rates are higher in SMs.

Overall, our preliminary analysis contradicts the findings in Jiang et al. [9]. There is not a superior model among all that we analyzed. Moreover, the performance of the defect predictors of Maven-2, Perl and Rails projects have achieved neither the state-of-the-art performance [13], nor manual review inspection performance reported in the literature [20]. Different settings may lead to the selection of different models. These settings could be characterized as dataset and developer characteristics, training data selection methods, used metrics and project development methodologies. In cases when developers have a relatively fewer commits and buggy changes, a selected model works better than a personalized model. In other cases, we may also need to build a hybrid framework which combines general and personalized approaches.

5 EXPECTED CONTRIBUTIONS

Our goal in this thesis is to design a personalized defect prediction framework that improves the state-of-the-art performance by tailoring predictions for each developer. The thesis work began in February 2017 and it is planned to be completed by June 2019. The analysis reported in this paper map to RQ1; showing that the personalization of SDP is open for further investigation. We consider to contribute to this field by implementing following items:

A framework that dynamically configures itself to choose between PM, SM, and GM. Our findings related to RQ1 show us that there is not a single best model for all developers and projects. We need a framework that will choose the best suitable model according to the relevant developer, project and development data. Instead of using

Table 1: Preliminary models' performance [1]

Project (# selected authors)	Model	Data Size	pd	pf	prec.	F1
Gimp (22)	PM	132 - 9.222	0,597	0,121	0,741	0,687
	SM	25.579	0,723	0,104	0,820	0,768
	GM	32.875	0,669	0,091	0,807	0,732
Maven 2 (2)	PM	868 - 2.296	0,331	0,039	0,627	0,434
	SM	3.164	0,319	0,028	0,638	0,424
	GM	5.399	0,265	0,018	0,633	0,374
Perl (30)	PM	224 - 7.850	0,351	0,069	0,614	0,452
	SM	34.731	0,381	0,069	0,647	0,480
	GM	50.485	0,361	0,062	0,647	0,465
PostgreSql (8)	PM	632 - 12.755	0,598	0,119	0,700	0,656
	SM	30.268	0,612	0,144	0,726	0,664
	GM	35.005	0,601	0,148	0,718	0,656
Rails (56)	PM	33 - 3.314	0,355	0,086	0,585	0,428
	SM	17.486	0,395	0,092	0,625	0,484
	GM	32.866	0,301	0,044	0,612	0,405
Rhino (3)	PM	274 - 1.095	0,701	0,159	0,739	0,722
	SM	2.249	0,736	0,208	0,763	0,749
	GM	2.955	0,689	0,189	0,740	0,715

[1]Personalized models are given in the link: https://kovan.itu.edu.tr/index.php/s/gGAwrUr8KzUQIkY

Table 2: Pairwise comparisons among PM, SM, GM

Models to be compared	Comparisons	pd >, <, =	pf >, <, =	prec. >, <, =	F1 >, <, =
PM . SM	121	49, 67, 5	49, 54, 18	33, 62, 26	45, 59, 17
PM . GM	121	62, 50, 9	70, 39, 12	35, 58, 28	55, 47, 19
SM . GM	6	6, 0, 0	5, 0, 1	2, 0, 4	6, 0, 0

a single model for a developer, it would tune its analytics to identify the best performing one according to the characteristics of a change. Further analysis on RQ2 would also improve the configuration with respect to contextual factors.

Factors affecting the performance of PM. An empirical analysis will be done to identify extrinsic factors such as project size, development methodology, and intrinsic factors such as developer experience, coding habits, frequently made defects, and metric set, that influence the performance of PMs. We will start this analysis to address our RQ2 by investigating code ownership [19] and scattering of changes [6], and constructing defect predictors on a commercial dataset collected from an industrial partner.

A group customized model. Instead of using separated prediction models for each developer, a group customized model could be used for a group of developers to make the framework more lightweight. Developers can be grouped according to their development behaviours or their relationship in terms of co-changed modules. This idea maps to our RQ3 and would also be helpful for overcoming the cold start problem occurred when a new developer is involved into a project.

Acknowledgment. This research is supported in part by Scientific Research Projects Division of Istanbul Technical University with the project number MGA-2017-40712, and the Scientific and Technological Research Council of Turkey (TUBITAK) with the project number 5170048. Thanks to my PhD supervisor Ayşe Tosun for her valuable contributions.

REFERENCES

[1] I. Avazpour, T. Pitakrat, L. Grunske, and J. Grundy. 2014. Dimensions and metrics for evaluating recommendation systems. In *Recommendation systems in software engineering*. Springer, 245–273.

[2] R. M. Bell, T. J. Ostrand, and E. J. Weyuker. 2013. The limited impact of individual developer data on software defect prediction. *Empirical Software Engineering* 18, 3 (2013), 478–505.

[3] N. Bettenburg, M. Nagappan, and A. E. Hassan. 2012. Think locally, act globally: Improving defect and effort prediction models. In *Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on*. IEEE, 60–69.

[4] Google Official Blog. 2009. Personalized search for everyone, https://googleblog.blogspot.com.tr/2009/12/personalized-search-for-everyone.html. (2009).

[5] G. Calikli and A. Bener. 2015. Empirical analysis of factors affecting confirmation bias levels of software engineers. *Software Quality Journal* 23, 4 (2015), 695–722.

[6] D. Di Nucci, F. Palomba, G. De Rosa, G. Bavota, R. Oliveto, and A. De Lucia. 2017. A developer centered bug prediction model. *IEEE Transactions on Software Engineering* (2017).

[7] W. Fu, T. Menzies, and X. Shen. 2016. Tuning for software analytics: Is it really necessary? *Information and Software Technology* 76 (2016), 135–146.

[8] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell. 2012. A systematic literature review on fault prediction performance in software engineering. *IEEE Transactions on Software Engineering* 38, 6 (2012), 1276–1304.

[9] T. Jiang, L. Tan, and S. Kim. 2013. Personalized defect prediction. In *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*. IEEE, 279–289.

[10] T. Lee, J. Nam, D. Han, S. Kim, and H. P. In. 2016. Developer Micro Interaction Metrics for Software Defect Prediction. *IEEE Transactions on Software Engineering* 42, 11 (2016), 1015–1035.

[11] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch. 2008. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *IEEE Transactions on Software Engineering* 34, 4 (2008), 485–496.

[12] R. Malhotra. 2015. A systematic review of machine learning techniques for software fault prediction. *Applied Soft Computing* 27 (2015), 504–518.

[13] T. Menzies, J. Greenwald, and A. Frank. 2007. Data mining static code attributes to learn defect predictors. *IEEE transactions on software engineering* 33, 1 (2007).

[14] T. Menzies, B. Turhan, A. Bener, G. Gay, B. Cukic, and Y. Jiang. 2008. Implications of ceiling effects in defect predictors. In *Proceedings of the 4th international workshop on Predictor models in software engineering*. ACM, 47–54.

[15] A. Tosun Misirli, E. Shihab, and Y. Kamei. 2016. Studying high impact fix-inducing changes. *Empirical Software Engineering* 21, 2 (2016), 605–641.

[16] P. Nemenyi. 1963. *Distribution-free multiple comparisons. unpublished Ph. D.* Ph.D. Dissertation. thesis, Princeton University, Princeton, New Jersey.

[17] T. J. Ostrand, E. J. Weyuker, and R. M. Bell. 2010. Programmer-based fault prediction. In *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*. ACM, 19.

[18] D. Radjenović, M. Heričko, R. Torkar, and A. Živkovič. 2013. Software fault prediction metrics: A systematic literature review. *Information and Software Technology* 55, 8 (2013), 1397–1418.

[19] F. Rahman and P. Devanbu. 2011. Ownership, Experience and Defects: A Fine-grained Study of Authorship. In *Proceedings of the 33rd International Conference on Software Engineering (ICSE '11)*. ACM, New York, NY, USA, 491–500. https://doi.org/10.1145/1985793.1985860

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

[21] C. E. Tucker. 2014. Social networks, personalized advertising, and privacy controls. American Marketing Association.

[22] X. Xia, D. Lo, X. Wang, and X. Yang. 2016. Collective personalized change classification with multiobjective search. *IEEE Transactions on Reliability* 65, 4 (2016), 1810–1829.

[23] K. Yamashita, S. McIntosh, Y. Kamei, A. E. Hassan, and N. Ubayashi. 2015. Revisiting the applicability of the pareto principle to core development teams in open source software projects. In *Proceedings of the 14th International Workshop on Principles of Software Evolution*. ACM, 46–55.