ISSN NO: 9726-001X
Volume 12 Issue 01 Jan 2024

JOURNAL OF CURRENT SCIENCE

EMPOWERING DEVELOPERS WITH ADVANCED BUG PREDICTION MODEL

Mr.Gudala Karunakar[1], Swetha[2],Sowmya[3] ,Vasavi[4]

[1]Assistant Professor, Department of CSE, MallaReddy Engineering College for Women, Hyderabad, karun.capri@gmail.com

[2,3,4]UG Students, Department of CSE, Malla Reddy Engineering College for Women, Hyderabad, TS, India.

ABSTRACT

Several techniques have been proposed to accurately predict software defects. These techniques generally exploit characteristics of the code artefacts (e.g., size, complexity, etc.) and/or of the process adopted during their development and maintenance (e.g., the number of developers working on a component) to spot out components likely containing bugs. While these bug prediction models achieve good levels of accuracy, they mostly ignore the major role played by human-related factors in the introduction of bugs. Previous studies have demonstrated that focused developers are less prone to introduce defects than non-focused developers. According to this observation, software components changed by focused developers should also be less error prone than components changed by less focused developers. We capture this observation by measuring the scattering of changes performed by developers working on a component and use this information to build a bug prediction model. Such a model has been evaluated on 26 systems and compared with four competitive techniques. The achieved results show the superiority of our model, and its high complementarily with respect to predictors commonly used in the literature. Based on this result, we also show the results of a “hybrid” prediction model combining our predictors with the existing ones.

I. INTRODUCTION

Bug prediction techniques are used to identify areas of software systems that are more likely to contain bugs. These prediction models represent an important aid when the resources available for testing are scarce, since they can indicate where to invest such resources. The scientific community has developed several bug prediction models that can be roughly classified into two families, based on the information they exploit to

discriminate between "buggy" and "clean" code components. The first set of techniques exploits product metrics (i.e., metrics capturing intrinsic characteristics of the code components, like their size and complexity) [1], [2], [3], [4], [5], while the second one focuses on process metrics (i.e., metrics capturing specific aspects of the development process, like the frequency of changes performed to code components) [6], [7], [8], [9], [10], [11], [12]. While some studies highlighted the superiority of these latter with respect to the product metric based techniques [7], [13], [11] there is a general consensus on the fact that no technique is the best in all contexts [14], [15]. For this reason, the research community is still spending effort in investigating under which circumstances and during which coding activities developers tend to introduce bugs (see e.g., [16], [17], [18], [19], [20], [21], [22]). Some of these studies have highlighted the central role played by developer-related factors in the introduction of bugs.In particular, Eyolfson et al. [17] showed that more experienced developers tend to introduce less faults in software systems. Rahman and Devanbu [18] partly contradicted the study by Eyolfson et al. by showing that the experience of a developer has no clear link with the bug introduction. Bird et al. [20] found that high levels of ownership are associated with fewer bugs. Finally, Posnett et al. [22] showed that focused developers (i.e., developers focusing their attention on a specific part of the system) introduce fewer bugs than unfocused developers. Although such studies showed the potential of human-related factor sin bug prediction,thisinformation is not captured in state-of-the-art bug prediction models based on process metrics extracted from version history. Indeed, previous bug prediction models exploit predictors based on (i) the number of developers working on a code component [9] [10]; (ii) the analysis of changeproneness [13] [11] [12]; and (iii) the entropy of changes [8]. Thus, despite the previously discussed finding by Posnett et al. [22], none of the proposed bug prediction models considers how focused the developers performing changes are and how scattered these changes are. In our previous work [23] we studied the role played by scattered changes in bug prediction. We defined two measures, namely the developer's structural and semantic scattering. The first assesses how "structurally far" in the software project the code components modified by a developer in a given time period are.The "structural distance" between two code components is measured as the number of subsystems one needs to cross in order to reach one component from the other. The second measure (i.e., the semantic scattering) is instead meant to capture how much spread in terms of implemented responsibilities the code components modified by a developer in a given

time period are. The conjecture behind the proposed metrics is that high levels of structural and semantic scattering make the developer more error-prone. To verify this conjecture, we built two predictors exploiting the proposed measures, and we used them in a bug prediction model. The results achieved on five software systems showed the superiority of our model with respect to (i) the Basic Code Change Model (BCCM) built using the entropy of changes [8] and (ii) a model using the number of developers working on a code component as predictor [9] [10]. Most importantly, the two scattering measures showed a high degree of complementarity with the measures exploited by the baseline prediction models. In this paper, we extend our previous work [23] to further investigate the role played by scattered changes in bug prediction. In particular we: 1) Extend the empirical evaluation of our bug prediction model by considering a set of 26 systems. 2) Compare our model with two additional competitive approaches, i.e., a prediction model based on the focus metrics proposed by Posnett et al. [22] and a prediction model based on structural code metrics [24], that together with the previously considered models, i.e., the BCCM proposed by Hassan [8] and the one proposed by Ostrand et al. [9] [10], lead to a total of four different baselines considered in our study. 3) Devise and discuss the results of a hybrid bug prediction model, based on the best combination of predictors exploited by the five prediction models experimented in the paper. 4) Provide a comprehensive replication package [25] including all the raw data and working data sets of our studies. The achieved results confirm the superiority of our model, achieving a F-Measure 10.3% higher, on average, than the change entropy model [8], 53.7% higher, on average, with respect to what achieved by exploiting the number of developers working on a code component as predictor [9], 13.3% higher, on average, than the FMeasure obtained by using the developers' focus metric by Posnett et al. [22] as predictor, and 29.3% higher, on average, with respect to the prediction model built on top of product metrics [1]. The two scattering measures confirmed their complementarity with the metrics used by the alternative prediction models. Thus, we devised a "hybrid"model providing an average boost inprediction accuracy (i.e., F-Measure) of +5% with respect to the best performing model (i.e., the one proposed in this paper).Structure of the paper. Section 2 discusses the related literature, while Section 3 presents the proposed scattering measures. Section 4 presents the design of our empirical study and provides details about the data extraction process and analysis method. Section 5 reports the results of

the study, while Section 6 discusses the threats that could affect their validity. Section 7 concludes the paper.

II. LITERATURE REVIEW

A Developer Centered Bug Prediction Model, Dario Di Nucci; Fabio Palomba; Giuseppe De Rosa; Gabriele Bavota; Rocco Oliveto; Andrea De Lucia
Several techniques have been proposed to accurately predict software defects. These techniques generally exploit characteristics of the code artefacts (e.g., size, complexity, etc.) and/or of the process adopted during their development and maintenance (e.g., the number of developers working on a component) to spot out components likely containing bugs. While these bug prediction models achieve good levels of accuracy, they mostly ignore the major role played by human-related factors in the introduction of bugs. Previous studies have demonstrated that focused developers are less prone to introduce defects than non-focused developers. According to this observation, software components changed by focused developers should also be less error prone than components changed by less focused developers. We capture this observation by measuring the scattering of changes performed by developers working on a component and use this information to build a bug prediction model. Such a model has been evaluated on 26 systems and compared with four competitive techniques. The achieved results show the superiority of our model, and its high complementarity with respect to predictors commonly used in the literature. Based on this result, we also show the results of a “hybrid” prediction model combining our predictors with the existing ones.

III.EXISTING SYSTEM

- The Chidamber and Kemerer (CK) metrics [36] have been widely used in the context of bug prediction. Basili et al. [1] investigated the usefulness of the CK suite for predicting the probability of detecting faulty classes. They showed that five of the experimented metrics are actually useful in characterizing the bug-proneness of classes.

- The same set of metrics has been successfully exploited in the context of bug prediction by El Emam et al. [26] and Subramanyam et al. [27]. Both works

reported the ability of the CK metrics in predicting buggy code components, regardless of the size of the system under analysis.

- Ohlsson et al. [3] focused the attention on the use of design metrics to identify bug-prone modules. They performed a study on an Ericsson industrial system showing that at least four different design metrics can be used with equivalent results. The metrics performance are not statistically worse than those achieved using a model based on the project size.

- Zhou et al. [29] confirmed their results showing that size-based models seem to perform as well as those based on CK metrics except than the Weighted Method per Class on some releases of the Eclipse system. Thus, although Bell et al. [35] showed that more complex metric-based models have more predictive power with respect to size-based models, the latter seem to be generally useful for bug prediction.

Disadvantages

1. There is no Product and process metrics Technique in this system.
2. There is no technique called Structural scattering to find bugs effectively.

IV.PROPOSED SYSTEM

- The Proposed system is extended the empirical evaluation of our bug prediction model by considering a set of 26 systems.
- Compare our model with two additional competitive approaches, i.e., a prediction model based on the focus metrics proposed by Posnett et al. [22] and a prediction model based on structural code metrics [24], that together with the previously considered models, i.e., the BCCM proposed by Hassan [8] and the one proposed by Ostrand et al. [9] [10], lead to a total of four different baselines considered in our study.
- Devise and discuss the results of a hybrid bug prediction model, based on the best combination of predictors exploited by the five prediction models experimented in the paper.
- Provide a comprehensive replication package [25] including all the raw data and working data sets of our studies.

Advantages

ISSN NO: 9726-001X
Volume 12 Issue 01 Jan 2024

1. This system implements Research Questions and Baseline Selection which is effective in fining bigs.
2. The system has a technique to Detect bugs of Mining Software Repositories.

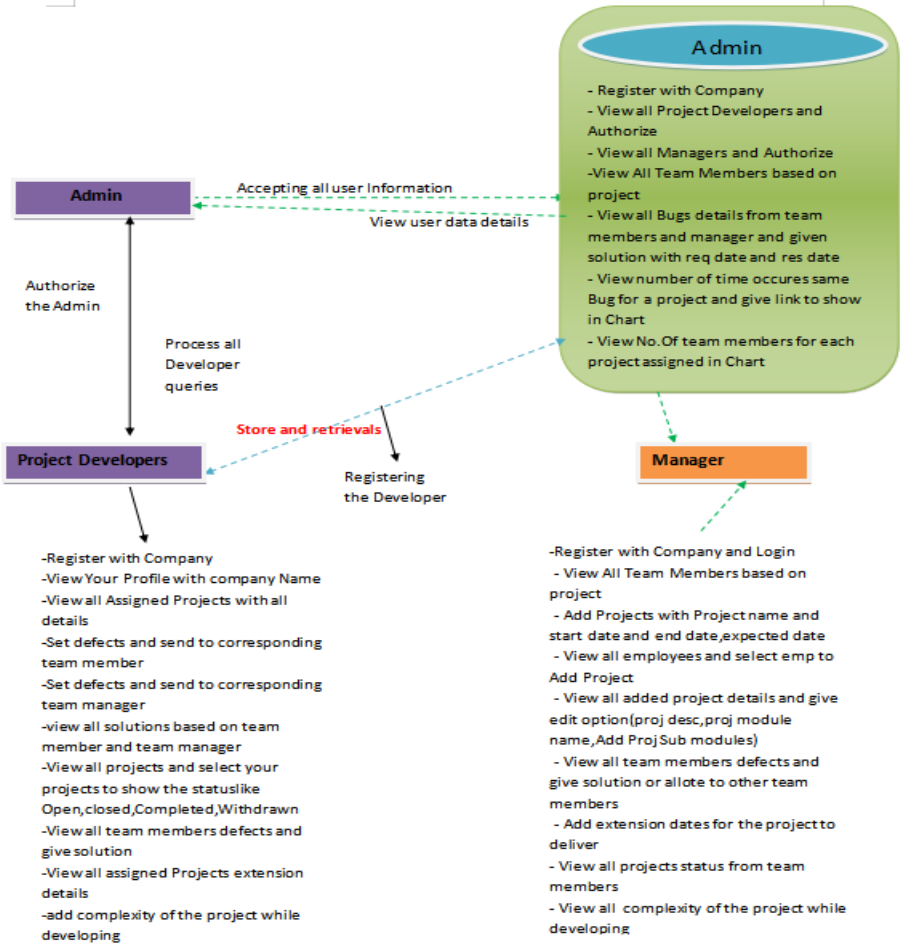

Fig1: Architecture diagram

V.MODULES

Admin

In this module, the Admin has to login by using valid user name and password. After login successful he can do some operations such as View all Project Developers and Authorize, View all Managers and Authorize, View all Team Members based on project, View all Bugs details from team members and manager and given solution with req date and res date , View number of time occures same Bug for a project and give link to show in Chart, View No.Of team members for each project assigned in Chart.

ISSN NO: 9726-001X
Volume 12 Issue 01 Jan 2024

View and Authorize Users

In this module, the admin can view the list of users who all registered. In this, the admin can view the user's details such as, user name, email, address and admin authorizes the users.

Manager

In this module, there are n numbers of managers are present. Manager should register before doing any operations. Once manager registers, their details will be stored to the database. After registration successful, he has to login by using authorized user name and password. Once Login is successful manager will do some operations like View all Team Members based on project, Add Projects with Project name and start date and end date,expected date, View all employees and select emp to Add Project, View all added project details and give edit option(proj desc,proj module name,Add Proj Sub modules), View all team members defects and give solution or allote to other team members, Add extension dates for the project to deliver, View all projects status from team members, View all complexity of the project while developing.

Project Developers

In this module, there are n numbers of users are present. User should register before doing any operations. Once user registers, their details will be stored to the database. After registration successful, he has to login by using authorized user name and password. Once Login is successful user will do some operations like View Your Profile with company Name,View all Assigned Projects with all details,Set defects and send to corresponding team member,Set defects and send to corresponding team manager,view all solutions based on team member and team manager,View all projects and select your projects to show the status like Open,closed,Completed,Withdrawn ,View all team members defects and give solution,View all assigned Projects extension details, add complexity of the project while developing.

VI.CONCLUSION

A lot of effort has been devoted in the last decade to analyze the influence of the development process on the likelihood of introducing bugs. Several empirical studies have been carried out to assess under which circumstances and during which coding activities developers tend to introduce bugs. In addition, bug prediction techniques built

on top of process metrics have been proposed. However, changes in source code are made by developers that often work under stressing conditions due to the need of delivering their work as soon as possible. The role of developer-related factors in the bug prediction field is still a partially explored area. This paper makes a further step ahead, by studying the role played by the developer's scattering in bug prediction.Specifically, we defined two measures that consider the amount of code components a developer modifies in a given time period and how these components are spread structurally (structural scattering) and in terms of the responsibilities they implement (semantic scattering). The defined measures have been evaluated as bug predictors in an empirical study performed on 26 open source systems. In particular, we built a prediction model exploiting our measures and compared its prediction accuracy with four baseline techniques exploiting process metrics as predictors. The achieved results showed the superiority of our model and its high level of complementarity with respect to the considered competitive techniques. We also built and experimente da"hybrid"predictionmodel on top of the eleven predictors exploited by the five competitive techniques. The achieved results show that (i) the "hybrid" is able to achieve a higher accuracy with respect to each of the five models taken in isolation, and (ii) the predictors proposed in this paper play a major role in the best performing "hybrid" prediction models. Our future research agenda includes a deeper investigation of the factors causing scattering to developers, and negatively impacting their ability of dealing with code change tasks. We plan to reach such an objective by performing a large survey with industrial and open source developers. We also plan to apply our technique at different levels of granularity, to verify if we can point out buggy code components at a finer granularity level (e.g., methods).

VII. REFERENCES

[1] V. Basili, L. Briand, and W. Melo, "A validation of object-oriented design metrics as quality indicators," Software Engineering, IEEE Transactions on, vol. 22, no. 10, pp. 751–761, Oct 1996.

[2] T. Gyim´othy, R. Ferenc, and I. Siket, "Empirical validation of object-oriented metrics on open source software for fault prediction," IEEE Transactions on Software Engineering (TSE), vol. 31, no. 10, pp. 897–910, 2005.

[3] N. Ohlsson and H. Alberg, "Predicting fault-prone software modules in telephone switchess," Software Engineering, IEEE Transactions on, vol. 22, no. 12, p. 886894, 1996.

[4] N. Nagappan and T. Ball, "Static analysis tools as early indicators of pre-release defect density," in Proceedings of the 27th International Conference on Software Engineering, ser. ICSE '05. New York, NY, USA: ACM, 2005, pp. 580–586. [Online]. Available: http://doi.acm.org/10.1145/1062455.1062558

[5] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting defects for eclipse," in Proceedings of the Third International Workshop on Predictor Models in Software Engineering, ser. PROMISE '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 9–. [Online]. Available: http://dx.doi.org/10.1109/PROMISE.2007.10

[6] A. N. Taghi M. Khoshgoftaar, Nishith Goel and J. McMullan, "Detection of software modules with high debug code churn in a very large legacy system," in Software Reliability Engineering. IEEE, 1996, pp. 364–371.

[7] J. S. M. Todd L. Graves, Alan F. Karr and H. P. Siy, "Predicting fault incidence using software change history," Software Engineering, IEEE Transactions on, vol. 26, no. 7, pp. 653–661, 2000.

[8] A. E. Hassan, "Predicting faults using the complexity of code changes," in ICSE. Vancouver, Canada: IEEE Press, 2009, pp. 78–88.

[9] R. Bell, T. Ostrand, and E. Weyuker, "The limited impact of individual developer data on software defect prediction," Empirical Software Engineering, vol. 18, no. 3, pp. 478–505, 2013. [Online]. Available: http://dx.doi.org/10.1007/s10664-011-9178-4

[10] T. J. Ostrand, E. J. Weyuker, and R. M. Bell, "Programmer-based fault prediction," in Proceedings of the 6th International Conference on Predictive Models in Software Engineering, ser. PROMISE '10. New York, NY, USA: ACM, 2010, pp. 19:1–19:10. [Online]. Available: http://doi.acm.org/10.1145/1868328.1868357

[11] R. Moser, W. Pedrycz, and G. Succi, "Analysis of the reliability of a subset of change metrics for defect prediction," in Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ser.

ESEM '08. New York, NY, USA: ACM, 2008, pp. 309–311. [Online]. Available: http://doi.acm.org/10.1145/1414004.1414063

[12] R. M. Bell, T. J. Ostrand, and E. J. Weyuker, "Does measuring code change improve fault prediction?" in Proceedings of the 7th International Conference on Predictive Models in Software Engineering, ser. Promise '11. New York, NY, USA: ACM, 2011, pp. 2:1–2:8. [Online]. Available: http://doi.acm.org/10.1145/2020390.2020392

[13] W. P. Raimund Moser and G. Succi, "A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction," in International Conference on Software Engineering (ICSE), ser. ICSE '08, 2008, pp. 181–190.

[14] N. Nagappan, T. Ball, and A. Zeller, "Mining metrics to predict component failures," in Proceedings of the 28th International Conference on Software Engineering, ser. ICSE '06. New York, NY, USA: ACM, 2006, pp. 452–461. [Online]. Available: http: //doi.acm.org/10.1145/1134285.1134349 [15] M. DAmbros, M. Lanza, and R. Robbes, "Evaluating defect prediction approaches: a benchmark and an extensive comparison," Empirical Software Engineering, vol. 17, no. 4, p. 531577, 2012.

[16] J. Sliwerski, T. Zimmermann, and A. Zeller, "Don't program on fridays! how to locate fix-inducing changes," in Proceedings of the 7th Workshop Software Reengineering, May 2005.

[17] L. T. Jon Eyolfso and P. Lam, "Do time of day and developer experience affect commit bugginess?" in Proceedings of the 8th Working Conference on Mining Software Repositories, ser. MSR '11, 2011, pp. 153–162.

[18] F. Rahman and P. Devanbu, "Ownership, experience and defects: a fine-grained study of authorship," in Proceedings of the 33rd International Conference on Software Engineering, ser. ICSE '11, 2011, pp. 491–500.

[19] E. J. W. J. Sunghun Kim and Y. Zhang, "Classifying software changes: Clean or buggy?" IEEE Transactions on Software Engineering (TSE), vol. 34, no. 2, pp. 181–196, 2008.

[20] C. Bird, N. Nagappan, B. Murphy, H. Gall, and P. Devanbu, "Don't touch my code!: Examining the effects of ownership on software quality," in Proceedings of the

19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, ser. ESEC/FSE '11. ACM, 2011, pp. 4–14.

[21] G. Bavota, B. De Carluccio, A. De Lucia, M. Di Penta, R. Oliveto, and O. Strollo, "When does a refactoring induce bugs? an empirical study," in Proceedings of the 12th International Working Conference on Source Code Analysis and Manipulation, ser. SCAM '12, 2012, pp. 104–113.

[22] D. Posnett, R. D'Souza, P. Devanbu, and V. Filkov, "Dual ecological measures of focus in software development," in Proceedings of the 2013 International Conference on Software Engineering, ser. ICSE '13. IEEE Press, 2013, pp. 452–461.

[23] D. D. Nucci, F. Palomba, S. Siravo, G. Bavota, R. Oliveto, and A. D. Lucia, "On the role of developer's scattered changes in bug prediction," in Proceedings of the 31st International Conference on Software Maintenance and Evolution, ICSME '15, Bremen, Germany, 2015, pp. 241–250.

[24] V.Basili,G.Caldiera,andD.H.Rombach,TheGoalQuestionMetric Paradigm. John Wiley and Sons, 1994.

[25] D. D. Nucci, F. Palomba, G. D. Rosa, G. Bavota, R. Oliveto, and A. D. Lucia. (2016) A developer centered bug prediction model - replication package - https://figshare.com/articles/A Developer Centered Bug Prediction Model/3435299.

[26] W. M. Khaled El Emam and J. C. Machado, "The prediction of faulty classes using object-oriented design metrics," Journal of Systems and Software, vol. 56, no. 1, p. 6375, 2001.

[27] R. Subramanyam and M. S. Krishnan, "Empirical analysis of ck metrics for object-oriented design complexity: Implications for software defects," Software Engineering, IEEE Transactions on, vol. 29, no. 4, p. 297310, 2003. [28] A. P. Nikora and J. C. Munson, "Developing fault predictors for evolving software systems," in Proceedings of the 9th IEEE International Symposium on Software Metrics. IEEE CS Press, 2003, pp. 338–349.

[29] Y. Zhou, B. Xu, and H. Leung, "On the ability of complexity metrics to predict fault-prone classes in object-oriented systems," Journal of Systems and Software, vol. 83, no. 4, pp. 660–674, 2010.

[30] N. Nagappan and T. Ball, "Use of relative code churn measures to predict system defect density," in Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on. IEEE, 2005, pp. 284–292. [31] A. E. Hassan and R. C. Holt, "Studying the chaos of code development," in Proceedings of the 10th Working Conference on Reverse Engineering, 2003.

[32] ——, "The top ten list: dynamic fault prediction," in Proceedings of the 21st IEEE International Conference on Software Maintenance, 2005, ser. ICSM '05. IEEE Computer Society, 2005, pp. 263–272.

[33] S. Kim, T. Zimmermann, E. J. Whitehead Jr, and A. Zeller, "Predicting faults from cached history," in Proceedings of the 29th international conference on Software Engineering. IEEE Computer Society, 2007, pp. 489–498.

[34] N. Nagappan, A. Zeller, T. Zimmermann, K. Herzig, and B. Murphy, "Change bursts as defect predictors," in Software Reliability Engineering (ISSRE), 2010 IEEE 21st International Symposium on. IEEE, 2010, pp. 309–318.

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

[36] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," IEEE Transactions on Software Engineering (TSE), vol. 20, no. 6, pp. 476–493, June 1994.

[37] G. Bavota, A. D. Lucia, A. Marcus, and R. Oliveto, "Using structural and semantic measures to improve software modularization," Empirical Software Engineering, vol. 18, no. 5, pp. 901–932, 2013.

[38] R. Baeza-Yates and B. Ribeiro-Neto, Modern Information Retrieval. Addison-Wesley, 1999.

[39] L. M. Y. Freund, "The alternating decision tree learning algorithm," in Proceeding of the Sixteenth International Conference on Machine Learning, 1999, pp. 124–133.

[40] R. Kohavi, "The power of decision tables," in 8th European Conference on Machine Learning. Springer, 1995, pp. 174–189.

[41] S. le Cessie and J. van Houwelingen, "Ridge estimators in logistic regression," Applied Statistics, vol. 41, no. 1, pp. 191–201, 1992.

[42] F. Rosenblatt, Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms. Spartan Books, 1961.

[43] G. H. John and P. Langley, "Estimating continuous distributions in bayesian classifiers," in Eleventh Conference on Uncertainty in Artificial Intelligence. San Mateo: Morgan Kaufmann, 1995, pp. 338–345.

[44] T. Menzies, B. Caglayan, Z. He, E. Kocaguneli, J. Krall, F. Peters, and B. Turhan. (2012, June) The promise repository of empirical software engineering data. [Online]. Available: http://promisedata.googlecode.com

[45] J. Sliwerski, T. Zimmermann, and A. Zeller, "When do changes induce fixes?" in Proceedings of the 2005 International Workshop on Mining Software Repositories, MSR 2005. ACM, 2005.

[46] L. Moonen, "Generating robust parsers using island grammars," in Reverse Engineering, 2001. Proceedings. Eighth Working Conference on, 2001, pp. 13–22.

[47]R.KohaviandG.H.John,"Wrappersforfeaturesubsetselection," Artif. Intell., vol. 97, no. 1-2, pp. 273–324, Dec. 1997. [Online]. Available: http://dx.doi.org/10.1016/S0004-3702(97)00043-X

[48] S. Mancoridis, B. S. Mitchell, C. Rorres, Y.-F. Chen, and E. R. Gansner, "Using automatic clustering to produce high-level system organizations of source code," in Proccedings of 6th International Workshop on Program Comprehension. Ischia, Italy: IEEE CS Press, 1998.

[49] W. J. Conover, Practical Nonparametric Statistics, 3rd ed. Wiley, 1998. [50] R. J. Grissom and J. J. Kim, Effect sizes for research: A broad practical approach, 2nd ed. Lawrence Earlbaum Associates, 2005.

[51] C. Bird, A. Bachmann, E. Aune, J. Duffy, A. Bernstein, V. Filkov, and P. Devanbu, "Fair and balanced?: Bias in bug-fix datasets," 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, ser. ESEC/FSE '09. New

York, NY, USA: ACM, 2009, pp. 121–130. [Online]. Available: http://doi.acm.org/10.1145/1595696.1595716

[52] K. Herzig and A. Zeller, “The impact of tangled code changes,” in Proceedings of the 10th Working Conference on Mining Software Repositories, MSR ’13, San Francisco, CA, USA, May 18-19, 2013, 2013, pp. 121–130.

[53] G. Bavota, G. Canfora, M. Di Penta, R. Oliveto, and S. Panichella, “The evolution of project inter-dependencies in a software ecosystem: The case of apache,” in Software Maintenance (ICSM), 2013 29th IEEE International Conference on, Sept 2013, pp. 280–289.

[54] I. Jolliffe, Principal Component Analysis. John Wiley & Sons, Ltd, 2005. [Online]. Available: http://dx.doi.org/10.1002/0470013192. bsa501

[55] P. A. Devijver and J. Kittler, Pattern Recognition: A Statistical