Facilitating Feasibility Analysis: The Pilot Defects Prediction Dataset Maker

Davide Falessi
California Polytechnique State University
San Luis Obispo, CA, USA
dfalessi@calpoly.edu

Max Jason Moede
California Polytechnique State University
San Luis Obispo, CA, USA
mmoede@calpoly.edu

ABSTRACT

Our industrial experience in institutionalizing defect prediction models in the software industry shows that the first step is to measure prediction metrics and defects to assess the feasibility of the tool, i.e., if the accuracy of the defect prediction tool is higher than of a random predictor. However, computing prediction metrics is time consuming and error prone. Thus, the feasibility analysis has a cost which needs some initial investment by the potential clients. This initial investment acts as a barrier for convincing potential clients of the benefits of institutionalizing a software prediction model. To reduce this barrier, in this paper we present the Pilot Defects Prediction Dataset Maker (PDPDM), a desktop application for measuring metrics to use for defect prediction. PDPDM receives as input the repository's information of a software project, and it provides as output, in an easy and replicable way, a dataset containing a set of 17 well-defined product and process metrics, that have been shown to be useful for defect prediction, such as size and smells. PDPDM avoids the use of outdated datasets and it allows researchers and practitioners to create defect datasets without the need to write any lines of code.

CCS CONCEPTS

Software and its engineering → Software creation and management.

Software and its engineering → Software development process management.

KEYWORDS

Defects prediction.

ACM Reference format:

Davide Falessi and Max Jason Moede, 2018. Facilitating Feasibility Analysis: The Pilot Defects Prediction Dataset Maker. In *Proceedings of the 4th ACM SIGSOFT International Workshop on Software Analytics (SWAN '18), November 5, 2018, Lake Buena Vista, FL, USA. ACM, New York, NY, USA, 4 pages.* https://doi.org/10.1145/3278142.3278147

1 INTRODUCTION

In today's society, reducing defects is important from both economic and safety perspectives. For this reason, a significant amount of research effort has been spent trying to reduce defects by accurately estimating where they are [1] [2] [3][4][5][6][7][8] or how they can be avoided [9] [10]. A software metric is a standard of measure of a degree to which a software system or process possesses some property. It is generally agreed that software product and process metrics are vital in supporting accurate predictions of the defects existing in a software system [11]. For instance, regarding the metric size, we know that larger classes are expected to be more defect[1] prone than smaller classes [12].

Our industrial experience in institutionalizing defect prediction models in the software industry shows that the first step is to measure prediction metrics and defects to assess the feasibility of the tool, i.e., if the accuracy of the defect prediction tool is higher than of a random predictor. However, computing prediction metrics is time consuming and error prone [13] [14]. Thus, the feasibility analysis has a cost which needs some initial investment by the potential clients. This initial investment acts as a barrier for convincing potential clients of the benefits of institutionalizing a software prediction model.

Several significant efforts have been spent in creating [15], collecting [16] and selecting [17] defects datasets, i.e., files reporting metrics that can be used to estimate defects. However, existing datasets might be old and hence might not represent the current state of the software engineering practice. Moreover, metrics definitions are nebulous and even metrics that appear simple, such as size, are measured differently from application to application [18]. The absence of reproducibility in metrics measurement reduces the replicability of scientific studies [19] [20] and also acts as a barrier to students learning about software quality, safety and the application of machine learning to software engineering. Existing approaches to facilitate metrics measurements such as Alitheia Core [14], SAM [18] and BOA [21], serve as APIs or frameworks. This API-based approach

[1] As in Hall et al. [30] we use the term defect to indicate a fault or a bug. A failure is a possible result of a fault occurrence.

allows the user to configure and set up sophisticated large-scale experiments. However, one of the main drawbacks of such existing tools is the need to code thousands of lines of code; this coding implies a steep learning curve and possible defects or differences in metrics measurement. Moreover, according to our best knowledge, no existing tool measures the number of code smells; this is an important limitation as code smells have been proven to impact defects [22] [23].

To support feasibility analysis and study replicability, this paper presents the Pilot Defects Prediction Datasets Maker (PDPDM), an easy to use, completely automated desktop application for measuring defects prediction metrics. PDPDM receives, as input, the repository ID and authentication information of a software project, and it provides, in an easy and replicable way, as output, a dataset containing a set of 17 well-defined product and project metrics that have been shown to be useful for defect prediction. This set of metrics can be filtered according to experts' opinion or automated analysis. PDPDM allows researchers and practitioners to create defect datasets without the need to write any lines of code. We note that an earlier version of PDPDM was successfully used in a recent paper analyzing industrial projects of Keymind, USA [9]; we improve what we used in the previous paper by increasing its usability and with this paper we are making the tool available online to everyone.

The remainder of the paper is structured as follows. Section 2 details the inputs, outputs, and the software and system requirements to use PDPDM. Section 3 describes a practical use of PDPDM. Section 4 concludes the paper.

2 THE PDPDM TOOL

2.1 Installation and Requirements

PDPDM requires the following open source programs to be installed: SonarQube 6.7 or later, see https://www.sonarqube.org/, MySQL 14.14 or later, see https://www.mysql.com/downloads/, Python 2.7.14 or later, see https://www.python.org/downloads/, Maven 3.5.2 or later, see https://maven.apache.org/, Git 2.15.1 or later, see https://git-scm.com/downloads, Python JIRA API 1.0.16.dev24 or later, see https://jira.readthedocs.io/en/master/.

2.2 Inputs

The inputs of PDPDM are the Git URL, the name of the project, and the JIRA URL. For instance, the following command line analyzes the Tika[2] project:

python ./PDPDM.py https://github.com/apache/tika.git Tika https://issues.apache.org/jira/

2.3 Outputs

The output of PDPDM is a dataset containing a set of 17 well-defined product and project metrics that have been shown to be useful for defect prediction [9] [10] [11]. Table 1 reports such metrics. Each metric is measured on a specific class, and since the same class changes over time, PDPDM reports, in addition to the 17 metrics, the name of the module and the release number. Finally, to support defect predictions studies, PDPDM also reports the defectiveness of the module, i.e., if the module had no defect or otherwise. The dataset can be refined according to experts' judgement and/or automated techniques for feature selection.

[2] https://tika.apache.org/

Table 1: Metrics, defined per module in a release, to estimate the defective files.

Metric (File C)	Description
Size	LOC
LOC_touched	Sum over revisions of LOC added+deleted+modified
NR	Number of revisions
NFix	Number of bug fixes
NAuth	Number of Authors
LOC_added	Sum over revisions of LOC added
MAX_LOC_added	Maximum over revisions of LOC added
AVG_ LOC_added	Average LOC added per revision
Churn	Sum over revisions of added – deleted LOC in C
MAX_Churn	MAX_CHURN over revisions
AVG_Churn	Average CHURN per revision
ChgSetSize	Number of files committed together with C
MAX_ChgSet	Maximum of ChgSet_Size over revisions
AVG_ChgSet	Average of ChgSet_Size over revisions
Age	Age of C in weeks
WeightedAge	Age weighted by LOC_touched
NSmells	Number of smells

The time required to run PDPDM on a project depends on the size of the project in terms of number of commits, files, and releases and can vary from few minutes to several hours. For instance PDPDM required less than three hours to analyze Tika.

2.4 Architecture and Measurement Process

The source code of PDPDM is available online[3]. Figure 1 reports the architecture of PDPDM. The entire PDPDM project has about one thousand lines of Python code, and it consists of six main components. The JIRA and Git reader components can extract the project information. To measure the metrics, PDPDM automatically checks out the first revision of a release, compiles it, and runs its analyzer. Afterward, the release is deleted, and PDPDM proceeds with the next release.

To compute the number of smells [24] we used SonarQube[4] because it has been largely used in industry and successfully applied in past studies [10] [11]. PDPDM, can identify and measure about 800 types of smells. We note that these smells are not related only to complexity [25] but are also related to style and readability of the code. We refer to SonarQube for additional info about supported smells.

Figure 2 reports our approach for measuring metrics and defectiveness. Smells are measured at the beginning of a release and the related defects and changes are observable at the end of the release. As such, for each release and module, we compute the size and the number of smells just before the first revision of a release and change metrics and defects at the last revision of the same release. In order to count defects and link them to specific modules and releases we used the seminal algorithm of Sliwerski et al. [26].

Once all releases are analyzed, PDPDM exports the metrics to a CSV file by querying the internal SonarQube database.

[3] https://tinyurl.com/yauvp9td
[4] http://www.sonarqube.org/

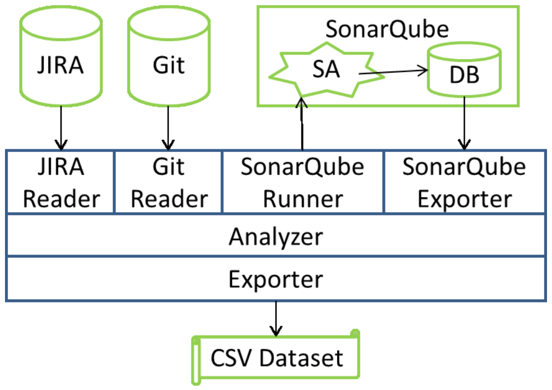

Figure 1: PDPDM architecture. In blue are the components, in green are the external entities: project repositories, SonarQube and the CSV output.

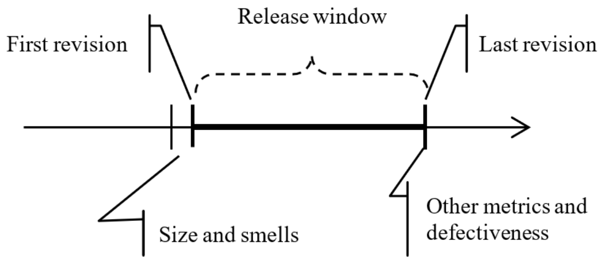

Figure 2: Data collection per release and file.

2.5 Supported Technologies

PDPDM currently supports the analysis of projects versioned with GIT, tracked with JIRA and written in Java. We chose these technologies because used by the large majority of our industrial partners. We plan to support more technologies, yet we believe the current version of PDPDM is already mature for publication as it can be used to analyze about 200 Apache projects, i.e., all projects using JIRA, Git, and Maven. We refer to a recent study on the issue of how to choose and target Apache projects to analyze [17].

2.6 Usages and Users

One important type of users is researchers in need of up-to-date datasets or that want to avoid reinventing the wheel by coding scripts to measure well known metrics that impact defects. For instance, a researcher can use the dataset provided by PDPDM to test a technology aimed at improving prediction accuracy by noise removal [3][4][5], tuning [6][7], feature selection [8], or other approaches. The datasets created by PDPDM can also be used to measure the statistical impact (e.g., correlation or test) of a metric (e.g., number of smells) on defectiveness. Another important group of users is users with limited programming knowledge or resources such as statisticians, projects managers, and undergraduate students. For instance, a manager can decide to institutionalize a prediction model by checking its accuracy; the manager could use PDPDM to create a dataset of their projects. This dataset can be processed in WEKA to check the accuracy of several machine learning models. See the online demo[5] for further details.

3 CASE STUDY

This section aims to show how PDPDM can be used to produce datasets which in turn can be used to support feasibility analysis and answer software engineering questions. To validate PDPDM we analyze the Apace project called Tika; the Tika dataset is available online[6]. We selected Tika by using STRESS [17]. Tika spans from May 2010 to Apr 2018 and it has a total of 47 releases and an average of 352 classes per release. Thus, the Tika datasets consist of 330,740 data; 20 columns (see Section 2.3) and 16,537 rows. The second author of the paper manually analyzed the first two releases data to ensure metrics measurement is correct.
In order to validate PDPDM we replicated a previous study [11] by using WEKA on the Tika dataset created by PDPDM. Specifically, we computed and compared the accuracy, as measured in AUC [27], of a dummy (i.e., random) model versus RandomForest [28]. Our results show that the accuracy of RandomForest is 63% higher than the dummy model. Thus, the use of the metrics provided by PDPDM supports defects prediction. Moreover, results show that using the number of smells increases the accuracy of RandomForest. See the online demo[5] for more details.

4 CONCLUSIONS

Our industrial experience shows that collecting defect metrics is an initial investment that acts as a barrier for convincing potential clients of the benefits of institutionalizing a software prediction model. To reduce this barrier, in this paper we present the Pilot Defects Prediction Dataset Maker (PDPDM), a desktop application for measuring defect prediction metrics. PDPDM receives, as input, the repository ID and authentication information of a software project, and it provides, in an easy and replicable way, as output, a dataset containing a set of 17 well-defined product and project metrics that show to be useful for defect prediction. PDPDM aims at facilitating the feasibility analysis phase of institutionalizing prediction models. PDPDM also aims at promoting study replication and supporting users with limited programming experience in collecting data related to software defects.
We plan to improve PDPDM in several directions including the following:

- Supporting more technologies: we plan to create new components such as an SVN and a Trello reader so that we can analyze more projects.
- Finer-grained information: we plan to improve the SonarQube exporter by counting individual smell rather than only the total number.

In the context of institutionalizing a software analytics tool, we bring the following questions to the workshop:

- How can we better facilitate feasibility analysis?
- What other phases can be automatically supported?
- Do we have a list of challenges and lessons learned other than the ones reported by Diep et al. [29]?

[5] https://tinyurl.com/ycuez9bl
[6] https://tinyurl.com/ya6yfjp7

- Do institutionalization processes change across contexts (size, domain, etc.) or tool types (defect prediction model vs. project competition status)?

ACKNOWLEDGMENTS

This work has been partially supported by the grant Empirically Establishing Actionable Causal Relationships between Code Smells and Software Maintenance Effort provided by the Cal Poly College of Engineering.

REFERENCES

[1] T. Menzies, Z. Milton, B. Turhan, B. Cukic, Y. Jiang, and A. Bener, "Defect prediction from static code features: Current results, limitations, new approaches," Autom. Softw. Eng., vol. 17, no. 4, pp. 375–407, 2010.
[2] B. Turhan, T. Menzies, A. B. Bener, and J. Di Stefano, "On the relative value of cross-company and within-company data for defect prediction," Empir. Softw. Eng., vol. 14, no. 5, pp. 540–578, 2009.
[3] S. Kim, H. Zhang, R. Wu, and L. Gong, "Dealing with noise in defect prediction," in Proceeding of the 33rd international conference on Software engineering - ICSE '11, 2011, p. 481.
[4] K. Herzig, S. Just, and A. Zeller, "It's not a bug, it's a feature: how misclassification impacts bug prediction," in 2013 International Conference on Software Engineering (ICSE '13), 2013, pp. 392–401.
[5] F. Rahman, D. Posnett, I. Herraiz, and P. Devanbu, "Sample size vs. bias in defect prediction," in Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2013, 2013, p. 147.
[6] W. Fu, T. Menzies, and X. Shen, "Tuning for software analytics: Is it really necessary?," Inf. Softw. Technol., vol. 76, pp. 135–146, 2016.
[7] C. Tantithamthavorn, S. Mcintosh, A. E. Hassan, and K. Matsumoto, "Automated Parameter Optimization of Classification Techniques for Defect Prediction Models," Proc. Int. Conf. Softw. Eng., p. To appear, 2016.
[8] K. Gao, T. M. Khoshgoftaar, H. Wang, and N. Seliya, "Choosing software metrics for defect prediction: An investigation on feature selection techniques," Softw. - Pract. Exp., vol. 41, no. 5, pp. 579–606, 2011.
[9] D. Falessi, B. Russo, and K. Mullen, "What if I had no smells," in ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM2017), 2017.
[10] D. Falessi and A. Voegele, "Validating and prioritizing quality rules for managing technical debt: An industrial case study," in 2015 IEEE 7th International Workshop on Managing Technical Debt, MTD 2015 - Proceedings, 2015.
[11] M. D'Ambros, M. Lanza, and R. Robbes, "Evaluating defect prediction approaches: a benchmark and an extensive comparison," Empirical Software Engineering, vol. 17. pp. 531–577, 2012.
[12] D. I. K. Sjoberg, A. Yamashita, B. C. D. Anda, A. Mockus, and T. Dyba, "Quantifying the Effect of Code Smells on Maintenance Effort," IEEE Trans. Softw. Eng., vol. 39, no. 8, pp. 1144–1156, Aug. 2013.
[13] G. Robles, "Empirical Software Engineering Research on Free/Libre/Open Source Software," 2006 22nd IEEE Int. Conf. Softw. Maint., no. January, pp. 347–350, 2006.
[14] T. Menzies, B. Caglayan, E. Kocaguneli, J. Krall, F. Peters, and B. Turhan, "The Promise Repository of Empirical Software Engineering Data," North Carolina State University, Department of Computer Science, 2012. .
[15] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting defects for eclipse," in Proceedings - ICSE 2007 Workshops: Third International Workshop on Predictor Models in Software Engineering, PROMISE'07, 2007.
[16] G. Gousios and D. Spinellis, "Conducting quantitative software engineering studies with Alitheia Core," Empir. Softw. Eng., vol. 19, no. 4, pp. 885–925, 2014.
[17] D. Falessi, W. Smith, Serebrenik, and Alexander, "STRESS: A Semi-Automated, Fully Replicable Approach for Project Selection," in ACM/IEEE Conference on Empirical Software Engineering, 2017.
[18] A. Edelman, W. Frakes, and C. Lillie, "SAM: Simple API for object-oriented code metrics," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2008, vol. 5030 LNCS, pp. 347–359.
[19] F. J. Shull, J. C. Carver, S. Vegas, and N. Juristo, "The role of replications in Empirical Software Engineering," Empir. Softw. Eng., vol. 13, no. 2, pp. 211–218, 2008.
[20] J. M. González-Barahona and G. Robles, "On the reproducibility of empirical software engineering studies based on data retrieved from development repositories," Empir. Softw. Eng., vol. 17, no. 1–2, pp. 75–89, 2012.
[21] R. Dyer, H. A. Nguyen, H. Rajan, and T. N. Nguyen, "Boa: Ultra-Large-Scale Software Repository and Source-Code Mining," ACM Trans. Softw. Eng. Methodol., vol. 25, no. 1, pp. 1–34, 2015.
[22] F. Palomba, M. Zanoni, F. Arcelli Fontana, A. De Lucia, and R. Oliveto, "Toward a Smell-aware Bug Prediction Model," IEEE Transactions on Software Engineering, 2017.
[23] D. Falessi and A. Voegele, "Validating and prioritizing quality rules for managing technical debt: An industrial case study," in IEEE 7th International Workshop on Managing Technical Debt (MTD) Validating and prioritizing quality rules for managi, 215AD.
[24] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts, "Refactoring: Improving the Design of Existing Code," Xtemp01, pp. 1–337, 1999.
[25] M. Lanza and R. Marinescu, Object-Oriented Metrics in Practice. 2006.
[26] J. Śliwerski, T. Zimmermann, and A. Zeller, "When do changes induce fixes?," ACM SIGSOFT Softw. Eng. Notes, vol. 30, no. 4, p. 1, 2005.
[27] Chakkrit Tantithamthavorn and Ahmed E. Hassan, "An Experience Report on Defect Modelling in Practice: Pitfalls and Challenges," in ICSE-SEIP, 2018.
[28] L. Breiman, "Random forests," Mach. Learn., vol. 45, no. 1, pp. 5–32, 2001.
[29] M. Diep, L. Esker, D. Falessi, L. Layman, M. Shaw, and F. Shull, "Applying Software Data Analysis in Industry Contexts: When Research Meets Reality," in The Art and Science of Analyzing Software Data, 2015, pp. 327–348.
[30] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A systematic literature review on fault prediction performance in software engineering," IEEE Transactions on Software Engineering, vol. 38, no. 6. pp. 1276–1304, 2012.