A review of process metrics in defect prediction studies

Marian Jureczko[1], Lech Madeyski[2]

[1]*Wroclaw University of Technology, Poland*
[2]*Institute of Informatics, Wroclaw University of Technology, Poland*

Abstract:

Process metrics appear to be an effective addition to software defect prediction models usually built upon product metrics. We present a review of research studies that investigate process metrics in defect prediction. The following process metrics are discussed: Number of Revisions, Number of Distinct Committers, Number of Modified Lines, Is New and Number of Defects in Previous Revision. We not only introduce the definitions of the aforementioned process metrics but also present the most important results, recent advances and the summary regarding the use of these metrics in software defect prediction models, as well as the taxonomy of the analysed process metrics.

Keywords:

defect prediction, software metrics, process metrics

1. Introduction

Process metrics, although sometimes not easy to collect, are becoming the crucial ingredients of novel, more accurate defect prediction models and systems. This paper reviews research activities regarding the use of process metrics in modern software defect prediction models. We follow the definitions of process metrics that was given by Henderson-Sellers' [1], who distinguished between product and process metrics. According to Henderson-Sellers product metric refers to software "snapshot" at a particular point of time, while process metric reflects the changes over time, e.g. the number of code changes. Recently the term historical metrics is sometimes used instead of process metrics, e.g. [2]. Nevertheless, we decided to use the traditional nomenclature which draws on Henderson-Sellers' [1], as well as Kan's nomenclature [3] and is consistent with one we assumed before [4]. We provide both, the definitions of the process metrics (Section 2), as well as presentation and discussion (in subsequent sections) of the most important results obtained in a wide range of empirical research studies regarding defect prediction models and process metrics. The studies are summarized in Section 8 and the conclusions are discussed in Section 9.

2. Metrics definitions

Number of Revisions (NR). The NR metric represents the number of revisions of a given Java class during development of the investigated release of a software system.

Number of Distinct Commiters (NDC). The NDC metric counts the number of distinct authors, usually developers, who committed their changes in a given Java class during the development of the investigated release of a software system.

Number of Modified Lines (NML). The value of the NML metric is equal to the sum of all lines of source code which were added or removed in a given Java class. Each of the revisions which were committed during the development of the investigated release of a software system is considered. According to the CVS version control system, a modification in a given line of source code is equivalent to removing the old version and subsequently adding a new version of the line.

Is New (IN). It is a nominal metric. It shows whether the given class existed in the previous version of the investigated system or whether it is a new one.

Number of Defects in Previous Version (NDPV). The NDPV metric counts the number of defects which were repaired in a given class during the development of the previous release of a software system.

3. Number of Revisions

There are several studies which investigate the number of historical revisions as a defect indicator. Weyuker et al. [5, 6, 7, 8, 9] used *the number of changes in the prior release*, as well as several other metrics, and created a very efficient defect prediction model by means of negative binomial regression. The 20% of files selected by the model contained up to 92% of the defects. They found in [6] that the files which were changed in the previous release had about three times as many faults as detected in the unchanged files.

Ratzinger et al. [10] used the number of historical revisions in three different types of defect prediction models and evaluated them on five industrial projects.

Graves et al. [11] concluded that the change history contains more useful information than could be obtained from the product (size and structure) metrics. They found specifically that the numbers of lines of code of a module (a metric widely used in the defect prediction models) are not helpful in predicting faults when the number of times a module was changed is taken into account.

Schröter et al. [12] mined the Eclipse bug and version databases and calculated the correlations of process measurement with pre- and post-release failures. In the case of pre-release failures the *number of changes* had the highest correlation coefficient among all the investigated process and product metrics. It was .34–.47 in the case of the Pearson correlation and .44–.56 in the case of the Spearman correlation. The work was later extended by using the metric in commercial projects [13] and by defining a metric which represents series of changes [14].

Illes-Seifert and Paech [2] investigated a number of process metrics, among others *Frequency of Change*. The Spearman's correlation of the *Frequency of Change* metric with number of defects was high in all nine investigated projects (.43–.64). The metric was recommended as a very good defect indicator.

Shihab et al. [15] investigated the *Eclipse* project in order to identify the metric subset that will be as good as a full metric set. 34 different metrics were analysed. The authors reduced the number of metrics to a much smaller, statistically significant and minimally collinear subset. The subset contained four metrics including the NR metric. The small set of metrics was used to build logistic regression models. The obtained models were compared to the models that make use of the full set of metrics and a very little difference in prediction accuracy was found.

Moser et al. [16] conducted a comparative analysis of the efficiency of the process and the product metrics for defect prediction. The process metrics set contained metrics similar

to NR, NDC, NML, NDPV and IN. The authors built three types of models: process metric models, product metric models and combined models. The process metric models and the combined models were more efficient than the product metrics models. Therefore, the authors recommended using the process metrics in defect prediction.

4. Number of Distinct Commiters

The metric is considered in several studies. Weyuker et al. used the number of distinct developers in a defect prediction model in [5] and then investigated the relevance of the metric in [8] and [9]. The experiment included the addition of the developer's information to the defect prediction model, which resulted in a slight improvement of the prediction efficiency. Without the developer information, the model was able to identify correctly 20% of the files containing 76.4–93.8% of the defects. Inclusion of the developer information yielded 76.4–94.8%. Weyuker et al. investigated three large industrial systems with a succession of releases over years of development.

Ratzinger et al. [10] included the number of developers in their defect prediction models. The authors did not study the usefulness of the NDC metric per se. Nevertheless, they concluded that it is not the size and complexity measures that dominate defect-proneness but rather many people-related issues.

The *Number of developers* metric was also used by Zimmermann et al. [17]. Zimmermann et al. analysed twelve open-source and industrial projects in order to assess the possibility of a successful cross project defect prediction.

Different results were obtained by Graves et al. [11]. A study of the code from a 1.5 million line subsystem of a telephone switching system gave no evidence that a large number of developers working on a module caused it to be more faulty. The generalized linear models were used to assess the usefulness of the metric.

Schröter et al. [12] analysed the Eclipse bug and version database and found a high correlation coefficient of the number of authors metric with pre- and post-release failures. It was .15–.41 in the case of the Pearson correlation and .13–.49 in the case of the Spearman correlation. The findings were confirmed by Illes-Seifer and Paech [2]. The Spearman's correlation coefficients of the number of distinct authors with the number of defects vary from .16 to .74 in the nine software projects which were investigated by lles-Seifer and Paech.

The most sophisticated studies on developer-based defect predictions were described in [18] and [19]. Both papers present the analysis of the relationship between a given developer and the density of defects. The usefulness of that approach cannot be actually estimated, since the papers come up with conflicting results.

Nagappan et al. [20] considered the metrics which describe organizational structure, among others *Number of Engineers*. The authors used the metrics to make predictions for *Windows Vista*. The obtained models had a precision and recall of 86.2% and 84.0%, respectively.

5. Number of Modified Lines

Considerable research has been performed on the extent to which the number of modified lines of code impacts the defect counts. The metric was already investigated in 1996 by Khoshgoftaar et al. [21]. The authors investigated two consecutive releases of a large legacy software system for telecommunications and employed 16 static software product metrics and the and the number of added or modified code lines in the prediction.

Layman et al. [22] defined four metrics that depend on the number of modified blocks:

— *Churn = NewBlocks + ChangedBlocks*;
— *RelativeChurn = (NewBlocks + ChangedBlocks) / TotalBlocks*;
— *Deleted churn = Deleted blocks / Total blocks*;
— *NCD churn = (NewBlocks + ChangedBlocks) / DeletedBlocks*.

The models which identify fault-prone components were built using the four aforementioned, as well as several other, structure metrics. Principal Component Analysis and Stepwise Logistic Regression were used in the construction procedure. The prediction accuracy of the models was very high: 73.3–86.7%. The authors used them in *Windows Server 2003* [23] and *Windows Vista* [20, 14] projects.

Purushothaman et al. [24] investigated the number of lines that were changed and described the distribution of modification size across files in a large, written in C/C++, telephone switch which consisted of 50 subsystems. They found out that the probability of a one-line change introducing one error was less then 4 percent.

A NML related metric was also investigated and advocated by Nagappan and Ball [23]. They concluded that it was an excellent predictor of defect density in a large industrial software system.

The finding was later partly confirmed by Zimmermann et al. [17] through investigating twelve open-source and industrial software projects and by Ratzinger et al. [10], who examined five industrial projects.

Śliwerski et al. [25] investigated *Eclipse* and *Mozilla*, and found that the larger the modification, the greater is the defect introduction probability.

Hassan [26] investigated five large open-source projects. He used the size of modification to define module entropy and subsequently used the entropy to predict successfully defects in the modules.

Giger et al. [27] employed in defect prediction a metric derived from the NML, namely fine-grained source code changes (SCC). The SCC metric captures the semantics of changes and was suggested by Fluri et al. [28, 29]. The SCC metric was investigated and compared with the NML metric with regard to its correlation with the number of defect as well as the prediction power (in univariate model). It was an empirical study conducted on 15 Eclipse plug-ins. The results showed significant strong correlation between defects and SCC. Furthermore, the SCC metric had a stronger correlation with defects than the NML metric. The defect prediction was investigated with respect to two different prediction aims: classifying a file as defect-prone or not defect-prone and predicting the number of defects. Based on the obtained results the authors concluded that SCC is a good predictor for defect-prone and not defect-prone files. Moreover, the SCC based models outperformed the models build with the NML metric in both aforementioned cases. Nevertheless, the authors admitted that from external validity point of view their work could be biased by the sole focus on Eclipse projects. A further study with regard to the SCC metric was presented in [30]. The set of investigated projects was extended to 16 Eclipse plug-ins. The authors defined a Gini coefficient that reflects code ownership (i.e. developers were used as the ”population” and NR, NML or SCC as the ”wealth”), hence the obtained metrics are similar to the NDC metric to some extent. The Gini coefficients calculated for different metrics (i.e NR, NML and SCC) were investigated with regard to the correlation with defects and prediction power (in univariate model). The correlation coefficients for NR related metrics varies from -0.05 to -0.79 (median=-0.55); NML from -0.34 to -0,75 (median=-0.54) and SCC from -0.29 to -0.76 and according to the conduced tests the difference between correlations were not significant. Similar results (i.e. no significant difference) were obtained in the case of prediction.

6. Is New

The IN metric as well as the age of a class or file were considered in several studies [5, 31, 11, 2, 32, 33, 7, 9], but the definitions used in those works are not uniform and sometimes differ from ours.

Ostrand et al. [33, 31] defined two file categories: *New File* and *Old File*; Illes-Seifer and Paech [2] defined three categories: *Newborn*, *Young*, *Old*. In [5] the *File age* means the number of months a given file has existed, in [7] and [9] — the number of consecutive prior releases in which the file appeared. Graves et al. [11] measured the average age of the code. They calculated the weighted average of the dates of the changes to the module, weighted by the size of the change.

Khoshgoftaar et al. [32] defined two metrics which were connected with the age: *Is New* (1 - If the module did not exist in the "ending" version (accepted by the customer or released for operational testing) of prior build, 0 - Otherwise) and *Age* (0 - If the module is new, 1 - If the module was new in the prior build, 2 - Otherwise).

The age of a class or a file was usually reported as a good defect count indicator. However, the explanation of the nature of the relation between the age and the defect count differs among studies.

Ostrand et al. [33, 31] successfully used the *Is New* metric in their model and discovered that in every release the percentage of the faulty new files is larger than the percentage of the faulty pre-existing files. The finding was confirmed by Bell et al. [5] — an exploratory analysis indicated that fault-proneness decreases with the age of a file. Similar findings were also obtained by Weyuker et al. [7, 8].

Khoshgoftaar et al. [32] observed that *Is New* varied in their significance and *Age* was a variable with a negative coefficient with the number of defects.

The result supports the finding that older modules are more reliable.

Different results were obtained by Illes-Seifert and Peach [2]. They concluded that "a file's age is a good indicator for its defect count (...), but *Newborn* and *Young files* are not the most fault-prone files".

Graves et al. [11] found that the *Age* metric, when combined with the number of deltas, greatly improved the fit of a defect prediction model to a point where it was slightly better than the reference model.

7. Number of Defects in Previous Version

Another issue to which extensive research was devoted is the extent to which the number of defects from the previous version impacts the defect counts in the current version.

Wahyudin et al. [34] constructed a Reliability Growth Model using the defects from previous version and successfully applied the model to a new release.

Arisholm and Briand [35] scrutinized the number of faults corrected in release $i - 1$ and the number of faults corrected in release $i - 2$, as well as several other metrics in a large, object-oriented, evolving legacy software system. They build a multivariate prediction model using logistic regression on log-transformed variables. The obtained predictions were correct in more than 80% cases.

Ostrand et al. [31, 7, 9] used negative binomial regression to develop models which predict faults in software systems. One of the metrics employed in the project was the square root of the number of faults identified during the previous release. The authors obtained very good prediction efficiency, as it was reported above.

Graves et al. [11] described a study using the fault history of the modules of a large telecommunications system. The authors constructed several defect prediction models. One of the models predicted the number of faults to be a constant multiple of the number of faults that had been found in a period of time in the module. The model was not the best one but the authors found it challenging to improve it.

Kim et al. [36] analysed seven open-source software systems and developed a cache based algorithm to predict future faults. Two of the principles behind the algorithm were connected with the NDPV metric: *temporal locality* (If an entity introduces a fault recently, it will tend to introduce other faults soon) and *spatial locality* (If an entity introduced a fault recently, "nearby entities" (in the sense of logical coupling) will also tend to introduce faults soon). The algorithm was able to cover 73%–95% of faults by selecting 10% of the most fault prone source code files.

Khoshgoftaar et al. [32] analysed a real-time military system and concluded that modules with faults in the past are likely to have faults in the future.

Ostrand and Weyuker [33] investigated whether faultiness persists between the subsequent releases and, according to the authors, there is moderate evidence that files remain high-fault till later releases. 17% to 54% of the high-fault files of release i are still high-fault in release $i + 1$.

Gyimothy et al. [37] calculated the correlations between the numbers of defects associated with the different versions of classes (Mozilla versions 1.0 to 1.6 were analysed). The correlations were on a high level and varied from .69 to .9.

Contradictory results were obtained by Illes-Seifert and Peach [2]. The authors concluded that, according to the results of their correlation analysis, "the number of defects found in the previous release of file does not correlate with its current defect count."

One might also doubt whether the NDPV metric is useful in defect prediction once the results obtained by Schröter et al. [12] are taken into consideration. Schröter et al. [12] calculated the correlation between pre- and post-release failures, but the obtained coefficients were smaller than the correlation coefficients calculated from the two metrics mentioned before (NR and NDC). Only three process metrics were considered in [12].

The analysis by Shihab et al. [15] showed that the NDPV metric is relevant in defect prediction.

8. Summary

The summary of the studies on the use of process metrics in defect prediction is presented in Table 1. The goals of each study are denoted as prediction (represents defect prediction improvement) or correlation (represents the investigation of the correlation with number of defects), while recommendations are denoted as: "↗" (represents positive recommendation), "↘" (represents negative recommendation) or "–" (no recommendation).

There is a number of works that recommend process metrics. However, it could be risky to arrive at a very optimistic conclusion, since the studies employed many different techniques and investigated a limited number of software projects, which influences their external validity. Furthermore, the positive recommendations describe only the fact that the authors find the metric useful in defect prediction. Specifically, a metric can be recommended in 2 studies but in the first one — as positively correlated, and in the second one — as negatively correlated with the number of defects. Nevertheless, that is a promising direction for further research,

and the level of prediction accuracy of the obtained results makes them potentially useful when employed in industry.

Table 1: Using process metrics in defect prediction: a summary of studies

Metric	Paper	Data sets	Goal of study	Recommendation
NR	Bel et al. [5]	proprietary systems over a 2.5 year period	prediction; correlation	↗
	Ostrand et al. [6]	13 releases of a proprietary system	prediction	–
	Weyuker et al. [7]	35 release of a proprietary system	prediction	↗
	Weyuker et al. [8]	35 release of a proprietary system	prediction using developer information	–
	Weyuker et al. [9]	3 proprietary systems with 17, 9 and 35 respectively and 1 system without formal releases	prediction	–
	Ratzinger et al. [10]	a proprietary system	Q-Mine: Predicting Short-Term Defects for Software Evolution	–
	Graves et al. [11]	subsystem of a telephone switching system	prediction	↗
	Schröter et al. [12]	Eclipse	correlation	–
	Illes-Seifert and Paech [2]	9 open-source projects	correlation	↗
	Shihab et al. [15]	Eclipse	prediction	↗
	Moser et al. [16]	Eclipse	prediction	↗
	Kim et al. [36]	7 open–source projects	prediction	–
	Gao et al. [38]	a proprietary system	methods of limiting the metrics set	↗
	Jureczko [39]	48 releases of 15 open-source and 38 releases of 7 proprietary projects	prediction; correlation	↗
NDC	Weyuker et al. [8]	35 release of a proprietary	prediction using developer information	↗
	Weyuker et al. [9]	3 proprietary systems with 17, 9 and 35 respectively and 1 system without formal releases	prediction	↗
	Ratzinger et al. [10]	a proprietary system	Q-Mine: Predicting Short-Term Defects for Software Evolution	–
	Zimmermann et al. [17]	4 open-source and 7 proprietary projects	cross-project defect prediction	–
	Graves et al. [11]	subsystem of a telephone switching system	prediction	↘
	Schröter et al. [12]	Eclipse	correlation	–
	Matsumoto et al. [18]	Eclipse	the effect of developer features	↗
	Weyuker et al. [19]	16 releases of a proprietary system	improving defect prediction using developer information	–

Continued on next page

Table 1 – *Continued from previous page*

Metric	Paper	Data sets	Goal of study	Recommendation
	Nagappan et al. [20]	Windows Vista	improving defect prediction using organizational complexity	↗
	Illes-Seifert and Paech [2]	9 open-source projects	correlation	↗
	Moser et al. [16]	Eclipse	prediction	↗
	Bell et al. [40]	3 proprietary systems, each containing 18 releases	improving defect prediction using developer information	–
	Jureczko [39]	48 releases of 15 open-source and 38 releases of 7 proprietary projects	prediction; correlation	↗
NML	Khoshgoftaar et al. [21]	2 releases of proprietary system	prediction	↗
	Layman et al. [22]	proprietary system	prediction	↗
	Nagappan and Ball [23]	Windows Server 2003	prediction	↗
	Nagappan et al. [20]	Windows Vista	improving defect prediction using organizational complexity	–
	Nagappan et al. [14]	Windows Vista	prediction	↗
	Purushothaman et al. [24]		distribution of defects with regard to change size	↗
	Zimmermann et al. [17]	4 open-source and 7 proprietary projects	cross-project defect prediction	–
	Ratzinger et al. [10]	a proprietary system	Q-Mine: Predicting Short-Term Defects for Software Evolution	–
	Śliwerski et al. [25]	Mozilla, Eclipse	prediction	↗
	Hassan [26]	6 open–source projects	prediction	↗
	Giger et al. [27]	15 Eclipse plug–ins	prediction	↗
	Giger et al. [30]	16 Eclipse plug–ins	prediction	–
	Graves et al. [11]	subsystem of a telephone switching system	prediction	↗
	Moser et al. [16]	Eclipse	prediction	↗
	Bell et al. [41]	18 releases of a proprietary system	prediction	↗
	Shin et al. [42]	2 open–source projects	predicting locations of vulnerable code	↗
	Sisman et al. [43]	AspectJ	prediction	↗
	Krishnan et al. [44]	Eclipse	prediction	↗
	Khoshgoftaar et al. [32]	subsystem a real time military system	prediction	↗
	Gao et al. [38]	a proprietary system	methods of limiting the metrics set	↗
	Jureczko [39]	48 releases of 15 open-source and 38 releases of 7 proprietary projects	prediction; correlation	↗
IN	Ostrand et al. [33]	13 releases of a proprietary system	distribution of faults with regard to files age	↗

Continued on next page

Table 1 – *Continued from previous page*

Metric	Paper	Data sets	Goal of study	Recommendation
	Ostrand et al. [31]	26 releases of 2 proprietary system	prediction	–
	Illes-Seifert and Paech [2]	9 open-source projects	correlation	–
	Bel et al. [5]	proprietary systems over a 2.5 year period	prediction; correlation	–
	Weyuker et al. [7]	35 release of a proprietary	prediction	↗
	Weyuker et al. [8]	35 release of a proprietary	prediction using developer information	–
	Weyuker et al. [9]	3 proprietary systems with 17, 9 and 35 respectively and 1 system without formal releases	prediction	–
	Graves et al. [11]	subsystem of a telephone switching system	prediction	↗
	Khoshgoftaar et al. [32]	subsystem a real time military system	prediction	–
	Ostrand et al. [6]	13 releases of a proprietary system	prediction	–
	Moser et al. [16]	Eclipse	prediction	↗
	Kim et al. [36]	7 open–source projects	prediction	–
NDPV	Wahyudin et al. [34]	Apache MyFaces	prediction; correlation	↗
	Arisholm and Briand [35]	proprietary system	prediction	↗
	Ostrand et al. [33]	13 releases of a proprietary system	persistence of high–fault files	↗
	Ostrand et al. [31]	26 releases of 2 proprietary system	prediction	–
	Weyuker et al. [7]	35 release of a proprietary	prediction	↗
	Weyuker et al. [9]	3 proprietary systems with 17, 9 and 35 respectively and 1 system without formal releases	prediction	–
	Graves et al. [11]	subsystem of a telephone switching system	prediction	–
	Kim et al. [36]	7 open–source projects	prediction	↗
	Khoshgoftaar et al. [32]	subsystem a real time military system	prediction	↗
	Gyimothy et al. [37]			
	Illes-Seifert and Paech [2]	9 open-source projects	correlation	–
	Schröter et al. [12]	Eclipse	correlation	–
	Shihab et al. [15]	Eclipse	prediction	↗
	Ostrand et al. [6]	13 releases of a proprietary system	prediction	–
	Moser et al. [16]	Eclipse	prediction	↗
	Sisman et al. [43]	AspectJ	prediction	↗
	Gao et al. [38]	a proprietary system	methods of limiting the metrics set	↗
	Jureczko [39]	48 releases of 15 open-source and 38 releases of 7 proprietary projects	prediction; correlation	↗

9. Conclusions and Further Work

On the basis of the review of process metrics presented in this paper we may set forth a taxonomy of the metrics according to the information sources required to calculate them. It is presented in Table 2.

Table 2. Taxonomy of process metrics

Metric	Source code	Developers	Defects
Number of Revisions (NR)	X		
Number of Modified Lines (NML)	X		
Is New (IN)	X		
Number of Distinct Commiters (NDC)	X	X	
Number of Defects in Previous Version (NDPV)	X		X

We may see that all of the process metrics rely on the information concerning the source code artefact changes, but only some of them combine it with the number of developers responsible for the aforementioned changes or the defects in earlier version. It is easy to imagine that new process metrics may include other sources of information than just source code artefacts. Changes in UML models or other artefacts, as well as a wide range of fluctuations in development teams (with respect to their size or structure), serve as good examples how to extend further the proposed taxonomy. It is also worth mentioning that some of the changes may be not directly related to the software process used.

This paper shows, in our opinion, that process metrics can be an effective addition to software defect prediction models usually built upon product metrics; however, the issue might be elaborated on and developed further. Our next aim is to evaluate the added predictive ability due to process metrics. Demonstration of a statistically significant association of a new process metric with defect risk is not enough. Hence, the next aim is to propose the evaluation framework in which it would be possible to assess the incremental value of new process metrics.

References

[1] B. Henderson-Sellers. *Object-oriented metrics: measures of complexity*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

[2] T. Illes-Seifert, B. Paech. *Exploring the relationship of a file's history and its fault-proneness: An empirical method and its application to open source programs*. *Information and Software Technology*, 52(5):539–558, 2010.

[3] S. H. Kan. *Metrics and Models in Software Quality Engineering*. Addison-Wesley, Boston, MA, USA, 2002.

[4] L. Madeyski. *Test-Driven Development: An Empirical Evaluation of Agile Practice*. Springer, (Heidelberg, Dordrecht, London, New York), 2010.

[5] R. M. Bell, T. J. Ostrand, E. J. Weyuker. *Looking for bugs in all the right places*. *ISSTA '06: Proceedings of the 2006 international symposium on Software testing and analysis*, pp. 61–72, New York, NY, USA, 2006. ACM.

[6] T. J. Ostrand, E. J. Weyuker, R. M. Bell. *Where the bugs are*. *ISSTA '04: Proceedings of the 2004 ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 86–96, New York, NY, USA, 2004. ACM.

[7] E. J. Weyuker, T. J. Ostrand, R. M. Bell. *Adapting a fault prediction model to allow widespread usage*. *PROMISE'06: Proceedings of the 4th International Workshop on Predictor Models in Software Engineering*, pp. 1–5, New York, NY, USA, 2006. ACM.

[8] E. J. Weyuker, T. J. Ostrand, R. M. Bell. *Using Developer Information as a Factor for Fault Prediction*. *PROMISE '07: Proceedings of the Third International Workshop on Predictor Models in Software Engineering*, p. 8, Washington, DC, USA, 2007. IEEE Computer Society.

[9] E. J. Weyuker, T. J. Ostrand, R. M. Bell. *Do too many cooks spoil the broth? Using the number of developers to enhance defect prediction models*. *Empirical Software Engineering*, 13(5):539–559, 2008.

[10] J. Ratzinger, M. Pinzger, H. Gall. *EQ-Mine: Predicting Short-Term Defects for Software Evolution*. M. Dwyer, A. Lopes, editors, *Fundamental Approaches to Software Engineering*, vol. 4422 of *Lecture Notes in Computer Science*, pp. 12–26. Springer Berlin / Heidelberg, 2007.

[11] T. L. Graves, A. F. Karr, J. S. Marron, H. Siy. *Predicting Fault Incidence Using Software Change History*. *IEEE Transactions on Software Engineering*, 26(7):653–661, 2000.

[12] A. Schröter, T. Zimmermann, R. Premraj, A. Zeller. *If your bug database could talk*. *In Proceedings of the 5th International Symposium on Empirical Software Engineering, Volume II: Short Papers and Posters*, pp. 18–20, 2006.

[13] N. Nagappan, T. Ball. *Using Software Dependencies and Churn Metrics to Predict Field Failures: An Empirical Case Study*. *ESEM '07: Proceedings of the First International Symposium on Empirical Software Engineering and Measurement*, pp. 364–373, Washington, DC, USA, 2007. IEEE Computer Society.

[14] N. Nagappan, A. Zeller, T. Zimmermann, K. Herzig, B. Murphy. *Change Bursts as Defect Predictors*. *Proceedings of the 2010 IEEE 21st International Symposium on Software Reliability Engineering*, ISSRE '10, pp. 309–318, Washington, DC, USA, 2010. IEEE Computer Society.

[15] E. Shihab, Z. M. Jiang, W. M. Ibrahim, B. Adams, A. E. Hassan. *Understanding the impact of code and process metrics on post-release defects: a case study on the Eclipse project*. *ESEM '10: Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, pp. 1–10, New York, NY, USA, 2010. ACM.

[16] R. Moser, W. Pedrycz, G. Succi. *A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction*. *ICSE '08: Proceedings of the 30th International Conference on Software Engineering*, pp. 181–190, New York, NY, USA, 2008. ACM.

[17] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, B. Murphy. *Cross-project defect prediction: a large scale experiment on data vs. domain vs. process*. *ESEC/FSE '09: Proceedings of the 7th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pp. 91–100, New York, NY, USA, 2009. ACM.

[18] S. Matsumoto, Y. Kamei, A. Monden, K. ichi Matsumoto, M. Nakamura. *An Analysis of Developer Metrics for Fault Prediction*. *PROMISE '10: Proceedings of the Sixth International Conference on Predictor Models in Software Engineering*, pp. 18:1–18:9. ACM, 2010.

[19] E. J. Weyuker, T. J. Ostrand, R. M. Bell. *Programmer–based Fault Prediction*. *PROMISE '10: Proceedings of the Sixth International Conference on Predictor Models in Software Engineering*, pp. 19:1–19:10. ACM, 2010.

[20] N. Nagappan, B. Murphy, V. Basili. *The influence of organizational structure on software quality: an empirical case study*. *ICSE '08: Proceedings of the 30th International Conference on Software Engineering*, pp. 521–530, New York, NY, USA, 2008. ACM.

[21] T. M. Khoshgoftaar, E. B. Allen, N. Goel, A. Nandi, J. McMullan. *Detection of software modules with high debug code churn in a very large legacy system*. *Proceedings of the The Seventh International Symposium on Software Reliability Engineering*, ISSRE '96, pp. 364–, Washington, DC, USA, 1996. IEEE Computer Society.

[22] L. Layman, G. Kudrjavets, N. Nagappan. *Iterative identification of fault-prone binaries using in-process metrics*. *ESEM '08: Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, pp. 206–212, New York, NY, USA, 2008. ACM.

[23] N. Nagappan, T. Ball. *Use of relative code churn measures to predict system defect density*. *ICSE '05: Proceedings of the 27th International Conference on Software Engineering*, pp. 284–292, New York, NY, USA, 2005. ACM.

[24] R. Purushothaman, D. E. Perry. *Toward Understanding the Rhetoric of Small Source Code Changes*. *IEEE Transactions on Software Engineering*, 31:511–526, 2005.

[25] J. Śliwerski, T. Zimmermann, A. Zeller. *When do changes induce fixes? MSR '05: Proceedings of the 2005 International Workshop on Mining Software Repositories*, pp. 1–5, New York, NY, USA, 2005. ACM.

[26] A. E. Hassan. *Predicting faults using the complexity of code changes*. *ICSE '09: Proceedings of the 31st International Conference on Software Engineering*, pp. 78–88, Washington, DC, USA, 2009. IEEE Computer Society.

[27] E. Giger, M. Pinzger, H. C. Gall. *Comparing fine-grained source code changes and code churn for bug prediction*. *Proceedings of the 8th Working Conference on Mining Software Repositories*, MSR '11, pp. 83–92, New York, NY, USA, 2011. ACM.

[28] B. Fluri, H. C. Gall. *Classifying Change Types for Qualifying Change Couplings*. *Proceedings of the 14th IEEE International Conference on Program Comprehension*, ICPC '06, pp. 35–45, Washington, DC, USA, 2006. IEEE Computer Society.

[29] B. Fluri, M. Wuersch, M. PInzger, H. Gall. *Change Distilling: Tree Differencing for Fine-Grained Source Code Change Extraction*. *IEEE Trans. Softw. Eng.*, 33(11):725–743, November 2007.

[30] E. Giger, M. Pinzger, H. Gall. *Using the gini coefficient for bug prediction in eclipse*. *Proceedings of the 12th International Workshop on Principles of Software Evolution and the 7th annual ERCIM Workshop on Software Evolution*, IWPSE-EVOL '11, pp. 51–55, New York, NY, USA, 2011. ACM.

[31] T. J. Ostrand, E. J. Weyuker, R. M. Bell. *Predicting the Location and Number of Faults in Large Software Systems*. *IEEE Transactions on Software Engineering*, 31(4):340–355, 2005.

[32] T. M. Khoshgoftaar, E. B. Allen, R. Halstead, G. P. Trio, R. M. Flass. *Using Process History to Predict Software Quality*. *Computer*, 31(4):66–72, 1998.

[33] T. J. Ostrand, E. J. Weyuker. *The distribution of faults in a large industrial software system*. *ISSTA '02: Proceedings of the 2002 ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 55–64, New York, NY, USA, 2002. ACM.

[34] D. Wahyudin, A. Schatten, D. Winkler, A. M. Tjoa, S. Biffl. *Defect Prediction using Combined Product and Project Metrics - A Case Study from the Open Source "Apache" MyFaces Project Family*. *SEAA '08: Proceedings of the 2008 34th Euromicro Conference Software Engineering and Advanced Applications*, pp. 207–215, Washington, DC, USA, 2008. IEEE Computer Society.

[35] E. Arisholm, L. C. Briand. *Predicting fault-prone components in a java legacy system*. *ISESE '06: Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering*, pp. 8–17, New York, NY, USA, 2006. ACM.

[36] S. Kim, T. Zimmermann, E. J. Whitehead Jr., A. Zeller. *Predicting Faults from Cached History*. *ICSE '07: Proceedings of the 29th International Conference on Software Engineering*, pp. 489–498, Washington, DC, USA, 2007. IEEE Computer Society.

[37] T. Gyimothy, R. Ferenc, I. Siket. *Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction*. *IEEE Transactions on Software Engineering*, 31(10):897–910, 2005.

[38] K. Gao, T. M. Khoshgoftaar, H. Wang, N. Seliya. *Choosing software metrics for defect prediction: an investigation on feature selection techniques*. *Software: Practice and Experience*, 41(5):579–606, 2011.

[39] M. Jureczko. *Significance of Different Software Metrics in Defect Prediction*. *Software Engineering: An International Journal*, 1(1):86–95, 2011.

[40] R. Bell, T. Ostrand, E. Weyuker. *The limited impact of individual developer data on software defect prediction*. *Empirical Software Engineering*, pp. 1–28, 2011. 10.1007/s10664-011-9178-4.

[41] R. M. Bell, T. J. Ostrand, E. J. Weyuker. *Does measuring code change improve fault prediction? Proceedings of the 7th International Conference on Predictive Models in Software Engineering*, Promise '11, pp. 2:1–2:8, New York, NY, USA, 2011. ACM.

[42] Y. Shin, A. Meneely, L. Williams, J. Osborne. *Evaluating Complexity, Code Churn, and Developer Activity Metrics as Indicators of Software Vulnerabilities*. *Software Engineering, IEEE Transactions on*, 37(6):772 –787, nov.-dec. 2011.

[43] B. Sisman, A. C. Kak. *Incorporating version histories in Information Retrieval based bug localization*. *Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on*, pp. 50 –59, june 2012.

[44] S. Krishnan, C. Strasburg, R. R. Lutz, K. Goševa-Popstojanova. *Are change metrics good predictors for an evolving software product line? Proceedings of the 7th International Conference on Predictive Models in Software Engineering*, Promise '11, pp. 7:1–7:10, New York, NY, USA, 2011. ACM.