Exploring the Relationship of History Characteristics and Defect Count: An Empirical Study

Timea Illes-Seifert
Institute for Computer Science
University of Heidelberg
Im Neuenheimer Feld 326, D-69120 Heidelberg
+49 (0) 6221 / 54 - 5817
illes@informatik.uni-heidelberg.de

Barbara Paech
Institute for Computer Science
University of Heidelberg
Im Neuenheimer Feld 326, D-69120 Heidelberg
+49 (0) 6221 / 54 - 5810
paech@informatik.uni-heidelberg.de

ABSTRACT
During the lifetime of a project, a huge amount of information is generated, e.g. in versioning systems or bug data bases. When analysed appropriately, the knowledge about the previous project characteristics allows estimating the project's future evolution. For example, it is very valuable to know particular history characteristics of a file indicating its fault proneness because it helps testers to focus their testing effort on these specific files. In this paper, we present the results of an empirical study, exploring the relationship between history characteristics of software entities and their defects. For this purpose, we analyze 9 open source java projects. The results show that there are some history characteristics that highly correlate with defects in software, e.g. the number of changes and the number of distinct authors performing changes to a file. The number of co-changed files does not correlate with the defect count.

Categories and Subject Descriptors
D.2.7 **[Distribution, Maintenance, and Enhancement]**: version control. [**Management]**: Software quality assurance (SQA). D.2.8 **[Metrics]**: Process metrics

General Terms
Measurement, Verification.

Keywords
Empirical study, defect database, versioning systems.

1. INTRODUCTION
Versioning control and defect tracking systems contain a large amount of information documenting the evolution of a software project. In practice, this information is often not deeply analysed in order to gain information which facilitates decisions in the present and permits reliable predictions for the future. Based on history characteristics extracted from versioning control systems, e.g. number of changes performed to a file, estimates for the future evolution can be made. Thus, for example, the expected number of changes can be predicted which allows to perform accurate cost estimates. Similarly, knowing defect detection rates over time of former releases, one can make predictions on remaining defects at the current point of time. This facilitates the decision if the software can be released or not. Information contained in versioning control and defect tracking systems can also be combined. For example, the relationship between history characteristics (e.g. change frequency) and software quality (e.g. measured by the defect count) can be explored. It is very valuable to know particular history characteristics of a file indicating its fault proneness because it helps testers to focus their testing effort on these specific files.

In this paper, we report the results of an empirical study exploring the relationship between history characteristics and quality in open source programs. For this purpose, we analyse 9 open source java projects during their whole lifetime. We use the defect count of a file as an indicator for its software quality and relate this measure to history characteristics of that file.

The remainder of this paper is organized as follows. Section 2 introduces basic definitions and concepts. Section 3 presents the design of our study. In Section 4, the data collection and analysis procedure are reported, whereas Section 5 contains the results of our empirical study. In Section 6, we discuss the threats to validity and in Section 7 an overview on related work is given. Finally, Section 8 concludes the paper and describes our future work.

2. BASIC TERMS AND DEFINITIONS
Versioning Control Systems (VCS) are useful for recording the history of documents edited by several developers. In order to edit a file, a developer has to checkout this file, edit it and commit this file back into the repository. Each time a developer commits a file, a message, describing what has been changed, can be added.

History Touch (HT). We define a history touch to be one of the commit actions where changes made by developers are submitted and include modifying, adding or removing files. Amongst others, each HT has the following attributes: author, affected file(s), date and message.

History. The history of a file subsumes all HTs that occurred to

DEFECTS'08, July 20, 2008, Seattle, Washington, USA.

that file from its birth[1] until present[2] or until its death[3]. **Release** is a point in time in the history of a project which denotes that a new or upgraded version is available. We consider only final releases.

A **defect** is "a flaw in a component or system that can cause the component or system to fail to perform its required function. A defect, if encountered during execution, may cause a **failure** of the component or system" [8]. Thus, a failure is the observable deviation of a component or system from its expected delivery.

Defect count is the number of defects identified in a file. The file *a* is more **fault-prone** than the file *b* if the defect count of the file *a* is higher than the defect count of the file *b*.

3. EXPERIMENT DESIGN

In this Section details on the experiment are described.

3.1 Goal and Research Questions

The main goal of this empirical study is to analyse the influence of a file's history on its defect count. These are our hypotheses and their rationale:

H1.1: The more distinct authors perform changes to a file, the higher the file's defect count. The rationale behind this hypothesis is that "too many cooks spoil the broth".

H1.2: The more changes have been performed to a file, the higher the defect count. The rationale behind this hypothesis is that a high amount of changes indicates that particular parts of the problem are not well understood and often need rework resulting in fault-prone files.

H1.3: The higher the number of co-changed files, the higher the defect count. The rationale behind this hypothesis is that a local change, affecting just one file, will cause fewer defects than changes affecting more files.

3.2 Dependent and Independent Variables

The dependent variable of our study is a file's defect count between two consecutive releases of its history. We relate a file's history characteristics prior to release to defects that occurred after release. In the example in Figure 1 we consider the number of changes performed in files between Release 1 and Release 2. Then, we relate this metric to the defect count measured between Release 2 and Release 3.

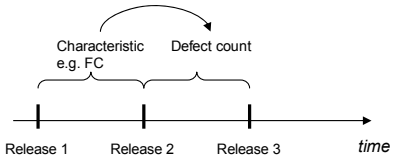

Figure 1. Defect count and history characteristics

The independent variables are history characteristics of a file. Table 1 summarizes the independent variables and their description.

[1] first occurrence in the VCS repository
[2] point of time where our empirical study started
[3] the file's removal point of time

Table 1. Dependent Variables, DA (Distinct Authors), FC (Frequency of Change), CF (Co-Changed Files)

ID	Description
DA	Number of distinct authors that performed HTs to a file between two consecutive releases.
FC	Number of HTs per file performed between two consecutive releases.
CF-[SUM, AVG]	Total/Average number of files that have been conjointly checked in with a file f between two consecutive releases.

3.3 Subject Projects

In this study we analysed 9 open source projects. We applied the following criteria to select the projects: 1) A bug tracking system is available, 2) The number of HTs > 50.000, 3) The project is written in Java. We included OSCache, a project that does not fulfil the criteria defined before, in order to compare the results obtained for all other projects with a smaller but mature project. **Apache Ant** (Ant)[4] is a Java application for automating the build process. **Apache Formatting Objects Processor** (Apache FOP)[5] reads a formatting object (FO) tree and renders the resulting pages to a specified output, e.g. PDF. **Chemistry Development Kit** (CDK)[6] is a Java library for bio- and chemo-informatics and computational chemistry. **Freenet**[7] is a distributed anonymous information storage and retrieval system. **Jetspeed2**[8] is an open portal platform and enterprise information portal. **Jmol**[9] is a „Java molecular viewer for three-dimensional chemical structures". **OSCache**[10] is a Java application, which allows the performance of fine grained dynamic caching of JSP content, servlet responses or arbitrary objects. **Pentaho**[11] is a Java based business intelligence platform. **TV-Browser** [12] is a Java based TV guide. Table 2 summarizes the attributes of the analyzed projects.

Table 2. Subject Programs

OS-Project	Project since	# Defects	# HTs	LOC	# Files	Defect Tracker
1. Ant (*1.7.0*)	2000	4804	62763	234253	1550	Bugzilla
2. FOP (*0.94*)	2002*	1478	30772	192792	1020	Bugzilla
3. CDK (*1.0.1*)	2001*	602	55757	227037	1038	Source Forge
4. Freenet (*0.7*)	1999*	1598	53887	68238	464	Mantis
5. Jetspeed2 (*2.1.2*)	2005	630	36235	236254	1410	JIRA
6. Jmol (*11.2*)	2001*	421	39981	117732	332	Source Forge
7. Oscache (*2.4.1*)	2000	2365	1433	19702	113	JIRA
8. Pentaho (*1.6.0*)	2005*	856	58673	209540	570	JIRA
9. TV-Browser (*2.6*)	2003	190	38431	170981	1868	JIRA

A „*" behind the data in the column "Project since" denotes the date of the registration of the project in SourceForge[13]. For the rest, the year of the first commit in the versioning system is indicated. The column "OS-Project" contains the name of the project followed by the project's latest version for which the metrics "LOC" (Lines of Code) and the number of files have been computed. Ant and FOP use Bugzilla[14] as defect tracking

[4] http://ant.apache.org/
[5] http://xmlgraphics.apache.org/fop/index.html
[6] http://sourceforge.net/projects/cdk/
[7] http://freenetproject.org/whatis.html
[8] http://portals.apache.org/jetspeed-2/
[9] http://jmol.sourceforge.net/
[10] http://www.opensymphony.com/oscache/
[11] http://sourceforge.net/projects/pentaho/
[12] http://www.tvbrowser.org/
[13] http://sourceforge.net/
[14] http://www.bugzilla.org/

system. CDK and Jmol use the SourceForge[13] whereas Freenet the Mantis[15] defect tracker. All other projects use JIRA[16].

4. DATA COLLECTION AND ANALYSIS

In order to analyse the relationship between defect count and history characteristics of files, the number of defects per file has to be computed. Defect tracking systems usually do not give any information on which files are affected by the defect. In order to find out this, information contained in VCS has to be analysed. For this purpose, we use a 3-level algorithm to determine the defect count per file. **Direct search:** First, we search for messages in the VCS containing the defect-IDs recorded in the defect tracking system. Messages containing the defect-ID and a text pattern, e.g. "fixed" or "removed", are indicators for defects that have been removed. In this case, the number of defects of the corresponding file has to be increased. **Keyword search:** In the second step, we search for keywords, e.g. "defect fixed", "problem fixed", within the messages which have not been investigated in the step before. We use about 50 keywords. **Multi-defects keyword search:** In the last step, we search for keywords which give some hints that more than one defect has been removed (e.g. „two defects fixed"). In this case, we increase the number of defects accordingly. A similar approach was previously presented e.g. in [16] and in [6]. In contrast to our 3-level algorithm, these approaches only perform a direct search. In order to validate our keyword search, we have randomly chosen messages from each project. We then classified them as messages in which a defect had been corrected or not and compared the results obtained manually with the results computed by running our algorithm. We used SPSS[17], version 11.5, for all statistical analyses.

5. RESULTS AND ANALYSIS

In this section, the results of the empirical study are presented.

5.1 Exploring the relationship between distinct authors and defect count

On average, 1.14 – 2.91 distinct authors performed HTs to a file. The minimum count of distinct authors is 1 in all projects, whereas the maximum count is 40 authors in case of the Freenet project. Table 3 summarizes basic statistical characteristics.

Table 3. Descriptive Statistics, Correlation Analysis for DA, All correlations are significant at 0.01 level.

	OS-Program	MAX authors	Min authors	Mean	Median	Std. Deviation	Variance	Spearm. DA
1	Ant	17.00	1.00	2.09	1.00	2.01	4.03	0.664
2	Apache-FOP	14.00	1.00	2.91	2.00	2.03	4.12	0.380
3	CDK	10.00	1.00	1.71	1.00	1.21	1.47	0.415
4	Freenet	40.00	1.00	1.94	1.00	1.88	3.53	0.741
5	Jetspeed2	7.00	1.00	2.00	2.00	0.98	0.96	0.741
6	Jmol	11.00	1.00	1.65	1.00	1.17	1.37	0.160
7	Oscache	4.00	1.00	1.54	1.00	0.79	0.62	0.480
8	Pentaho	3.00	1.00	1.14	1.00	0.37	0.13	0.352
9	TV-Browser	7.00	1.00	1.15	1.00	0.61	0.38	0.471

In order to analyse H 1.1, we first computed the correlation between the number of distinct authors, performing HTs to a file (DA), and its defect count. The results are listed in Table 3. The last column indicates the Spearman rank-order correlation coefficient. This coefficient is a measure for the dependency between two variables [18], in our case the number of distinct authors performing HTs to a file and its defect count. The coefficient can take values between -1 and 1, whereas 0 represents no linear correlation. In 6 of 9 projects, the correlation coefficient is higher than 0.4. This confirms H1.1 for the most part, i.e. the more authors touch a file, the higher its defect count after release.

In the second step, we performed the Mann-Whitney [18] non-parametric test[18]. For this test we divided the data in each project into two groups: one group contains files that have been changed by number of authors above average and a second group that contain files that have been changed by distinct authors below average. Differences between two populations can be analyzed with the help of Mann-Whitney test, in our case, differences between files with lower than average (la) and higher than average (ha) DA. The null hypothesis is that the defect count is the same in both groups; the alternative hypothesis is that it is not. Table 4 summarizes the results of the Mann-Whitney test. The results are significant for any chosen significance level. Based on this test, it can be concluded, that there is strong evidence from the data that files that have been touched by distinct authors above average have higher defect count compared to the files touched by distinct authors below average. This test also supports H1.1.

Table 4. Mann-Whitney Test for DA

		Distinct authors	N	Mean Rank	Rank Sum	Mann-Whitney-U	Z
1	Ant	la	4471.0	3345.7	14958799.5	4961643.5	-33.7
		ha	3306.0	4623.7	15285953.5		
2	Apache-FOP	la	709.0	958.8	679756.5	428061.5	-12.7
		ha	1740.0	1333.5	2320268.5		
3	CDK	la	2645.0	2693.4	7124088.0	3624753.0	-25.5
		ha	3676.0	3497.4	12856593.0		
4	Freenet	la	1633.0	1725.8	2818188.0	1484027.0	-29.9
		ha	3182.0	2758.1	8776332.0		
5	Jetspeed2	la	1670.0	1569.4	2620896.5	1225611.5	-14.3
		ha	1967.0	2030.9	3994806.5		
6	Jmol	la	839.0	1164.9	977352.0	624972.0	-12.0
		ha	1914.0	1470.0	2813529.0		
7	Oscache	la	107.0	84.4	9026.5	3248.5	-7.7
		ha	110.0	133.0	14626.5		
8	Pentaho	la	4913.0	5043.1	24776740.5	12705499.5	-37.1
		ha	7964.0	7300.1	58138262.5		
9	TV-Browser	la	8807.0	5239.9	46147607.0	7361579.0	-40.6
		ha	2065.0	6275.1	12958021.0		

5.2 Exploring the relationship between frequency of change, co-changed files and defect count

In order to analyse H1.2 and H1.3, we first computed the Spearman rank correlation coefficient between the FC, CF_MAX and CF_SUM of a file and its defect count. The results are listed in Table 5. A high correlation between FC and its defect count can be determined in all projects. Additionally, the Spearman rank-order correlation coefficient is higher than 0.4; in 3 projects this coefficient is even higher than 0.6 for all metrics. This confirms H1.2, i.e. the more changes have been performed to a file, the higher its defect count. The defect count of a file does not depend on the number of files simultaneously

[15] http://www.mantisbt.org/
[16] http://www.atlassian.com/software/jira/
[17] SPSS, http://www.spss.com/

[18] A non-parametric test does not make any assumptions concerning the distribution of parameters (in contrast to parametric tests).

checked in. For CF_MAX, there is no project with a Spearman rank-order correlation coefficient higher than 0.4. For CF_SUM, in only three cases, a high correlation can be determined. Thus, H1.3, i.e. the more files have been checked in simultaneously, the higher the defect count, has to be rejected.

Table 5. Correlation Analysis for FC, CF-MAX and CF-SUM. Correlations significant at 0.01 level (**), and at 0.05 level (*)

	OS-Program	Spearman FC	Spearman CF-SUM	Spearman CF-MAX
1	Ant	0.597	0.504	0.399
2	Apache-FOP	0.431	0.285	0.203
3	CDK	0.437	0.211	0.142
4	Freenet	0.641	0.220	-0.020
5	Jetspeed2	0.641	0.220	-0.020
6	Jmol	0.408	0.141	-0.042
7	Oscache	0.626	0.517	*0.036*
8	Pentaho	0.503	0.416	0.272
9	TV-Browser	0.442	-0.198	-0.248

In order to analyse H1.2 more in detail, we performed the Mann-Whitney non-parametric test. For this purpose, we divided the data in each project into two groups, one group containing stable files and another group containing unstable files. Stable files subsume all files that have the FC metric lower than average, unstable files have an FC value above average. The null hypothesis is that the defect count is the same in both groups; the alternative hypothesis is that it is not. Table 6 summarizes the results of the Mann-Whitney test. The results are significant at any chosen significance level. Based on the results of this test, it can be concluded that there is strong evidence from the data that unstable files have a higher defect count as compared with the stable files. This additional test also supports H1.2.

Table 6. Mann-Whitney Test Statistics for files with lower average (la) and higher average (ha) DA.

	OS Program	Stability	Mean Rank	Mann-Whitney-U	Z
1	Ant	Stable	3606.4	5220899.5	-34.9
		Unstable	5091.1		
2	Apache-FOP	Stable	2384.1	1849127.0	-31771.0
		Unstable	3677.0		
3	CDK	Stable	6649.8	19324818.5	-33.6
		Unstable	8149.6		
4	Freenet	Stable	3878.4	2595450.5	-50.5
		Unstable	6680.5		
5	Jetspeed2	Stable	3536.5	4176924.5	-24.5
		Unstable	4736.4		
6	Jmol	Stable	2757.4	2724067.0	-22.4
		Unstable	3487.9		
7	Oscache	Stable	169.8	11068.0	-7.9
		Unstable	239.3		
8	Pentaho	Stable	9196.6	27871921.0	-80.4
		Unstable	15671.9		
9	TV-Browser	Stable	6749.3	15070004.5	-37.7
		Unstable	7988.8		

6. THREATS TO VALIDITY

Internal validity is concerned with the degree to which conclusions about the causal effect of the independent variables on the dependent variable can be drawn [18]. One threat to validity is that not all developers deliver meaningful messages when they check-in files. Developers, for example, can also check in files without specifying any reason, even though they had corrected a defect. Thus, the defect count of a file can be higher than the defect count computed by our algorithm. This concern is alleviated by the size of the analysed OSPs. Nevertheless, we assume that meaningless messages are uniformly distributed among all developers.

External validity is concerned with the degree to which results can be generalized [18]. This issue is alleviated by the number and diversity of the analysed OSPs. The more OSP programs show the same characteristics, the higher the probability that other OSP programs would also show these characteristics. Additionally, we choose programs from different application domains in order to increase the representativeness of the study results. However, history characteristics of OSP programs and of commercially produced software may differ from each other. Furthermore, analyses of additional programs that are intended in our future work would increase the external validity.

7. RELATED WORK

To our knowledge, this is the first study that deeply analyses the influence of a file's history on its defect count.

The most similar work to our study is presented by Graves et al. in [7]. The authors explore to which extent history characteristics of a file are successful in predicting its defects. In contrast to our study, this study focuses on the development of several statistical models in order to identify the best performing one. Additionally, the authors in [7] perform their analysis on one legacy commercial system *in detail*, whereas our study analyses 9 open source projects *in breadth*. Another difference between our study and the study presented in [7] is the granularity of the analysed entity. Graves et al. analyse history characteristics of modules and relate these characteristics to the module's defect count. We compare both studies below. The best statistical model that predicts a module's defect count in [7] uses (beside others) the number of changes of a module in its history as parameter. This result is similar to our results. Another result derived in [7] is that the number of changes to a module is a better indicator for its defect count than its length. We computed the Spearman correlation coefficient of the LOC metric and the defect count in all projects. In 8 of 9 projects, the FC metric has higher correlation coefficients than the LOC metric. Thus, we can confirm the results in [7]. The main difference between the two studies is that in [7] the number of developers who had changed a module is not a good indicator for its defect count. This contrasts with our results. A possible explanation is that there is no common understanding of the problem domain in open source development, so that changes to a software entity, performed by different developers induce more defects than it is in the case of commercial development.

There are several other studies that focus on predicting the defect count of a software entity by combining product metrics and history metrics ([1], [9], [12], [14], [3], [17], [16]). One of the main differences that distinguishes our study from these studies is its magnitude. While most of the studies considered only one program, we have analysed 9 open source projects. Additionally, in contrast to our study, the aim of these studies is defect prediction. Our main goal is to analyse to what extend history characteristics influence software's defect count without selecting the best prediction model. Another difference to these studies, except of the study reported in [16], is that all other studies analyse commercial software. Except the study reported in [16], all other studies support our findings with respect to the

influence of changes on the defect count of a software entity. In [16], only pre-release defects (these are defects found 6 month before release) correlate with the number of changes performed to software entities. Only the studies presented in [3], [17] and [16] consider the influence of the number of authors performing changes to a software entity on the entity's defect count. The study reported in [17] confirms our results. In [16], only pre-release defects correlate with the number of authors performing changes. The results reported in [3] differ to our results. The number of co-changed files is not reported in any study. Most studies analysing this relationship are more fine-grained, i.e. they analyse to what extent the number of changed lines of code have impact on the defect count e.g. in [15] or [10]. A huge amount of research papers analyses the influence of other metrics of a software entity and its defect count, amongst others in [5], [2], [4], and in [11].

8. DISCUSSION AND FUTURE WORK

In this paper, we investigated the influence of a file's history on its defect count. Two of our initial hypotheses could be confirmed: The defect count of a file is influenced by the number of changes performed to that file and the number of authors that have touched it. This knowledge is useful for different roles in the development process. Testers can focus their testing activities on particularly these files. Additionally, they can use these metrics as parameters to build more complex prediction models. Quality engineers can monitor development activities and initiate reviews for often changed files in order to prevent a high defect count. Additionally, files changed by too many authors can be indicators for bad design. Thus, maintainers can identify candidates for refactorings. The hypotheses concerning the number of co-changed files could not be confirmed. Certainly, more precise metrics could be defined for the size of change, e.g. the number of lines of code edited. But we wanted to explore a coarser metric which can be easily computed.

In our future work we will focus on analysing the relationship between the file's age and its defect count. Additionally, we will analyse to what extent history characteristics combined with code characteristics, e.g. code complexity metrics, can be considered as good indicators for a file's defect count.

9. REFERENCES

[1] Arisholm, E. and Briand, L. C. 2006. Predicting fault-prone components in a java legacy system. In Proceedings of the 2006 ACM/IEEE international Symposium on Empirical Software Engineering (Rio de Janeiro, Brazil, September 21 - 22, 2006). ISESE '06. ACM, New York, NY, 8-17.

[2] Basili, V. R., Briand, L. C., and Melo, W. L. 1996. A validation of object-oriented design metrics as quality indicators. IEEE Transactions on Software Engineering vol. 22, pp. 751-761.

[3] Bell, R. M., Ostrand, T. J., and Weyuker, E. J. 2006. Looking for bugs in all the right places. 2006. In Proceedings of the 2006 international Symposium on Software Testing and Analysis ISSTA '06. ACM, New York, NY, 61-72

[4] Denaro, G., Morasca, S. and Pezzè, M. 2002. Deriving models of software fault-proneness. In Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering Ischia, Italy, pp. 361 - 368.

[5] Denaro, G and Pezzè, M. 2002. An empirical evaluation of fault-proneness models. In Proceedings of the International Conference on Software Engineering (ICSE 2002), Orlando, Florida, USA, pp. 241-251.

[6] Fischer, M., Pinzger, M., and Gall, H. 2003. Populating a Release History Database from Version Control and Bug Tracking Systems. In Proceedings of the international Conference on Software Maintenance. ICSM. IEEE Computer Society, Washington, DC, 23.

[7] Graves, T. L., Karr, A. F., Marron, J. S., and Siy, H. 2000. Predicting fault incidence using software change history. IEEE Transactions on Software Engineering, vol. 26.

[8] International Software Testing Qualifications Board. 2005. ISTQB Standard Glossary of Terms used in Software Testing V1.1.

[9] Khoshgoftaar, T. M., Allen, E. B., Halstead, R., Trio, G. P., and Flass, R. M. 1998. Using Process History to Predict Software Quality. Computer 31, 4 (Apr. 1998), 66-72.

[10] Nagappan, N. and Ball, T. 2005. Use of relative code churn measures to predict system defect density. In Proceedings of the 27th international Conference on Software Engineering. ICSE '05. ACM, New York, NY, 284-292.

[11] Nagappan, N., Ball, T., and Zeller, A. 2006. Mining metrics to predict component failures. In Proceedings of the International Conference on Software Engineering (ICSE 2006), Shanghai, China.

[12] Ohlsson, M. C., von Mayrhauser, A., McGuire, B.,Wohlin, C. 1999. Code Decay Analysis of Legacy Software through Successive Releases. Proceedings of IEEE Aerospace Conference, pp 69-81.

[13] Ostrand, T. J., Weyuker, E.J, Bell, R.M. 2004. Where the Bugs Are, Proceedings of the 2004 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA), pp 86-96.

[14] Ostrand, T. J., Weyuker, E. J., and Bell, R. M. 2005. Predicting the location and number of faults in large software systems. IEEE Trans. Software Eng., vol. 31, pp. 340-355.

[15] Purushothaman, R. 2005. Toward Understanding the Rhetoric of Small Source Code Changes. IEEE Trans. Softw. Eng. 31, 6 (Jun. 2005), 511-526.

[16] Schröter, A., Zimmermann,T. Premraj, R., and , R.,Zeller, A. If Your Bug Database Could Talk. 2006. In Proceedings of the 5th International Symposium on Empirical Software Engineering. (Rio de Janeiro, Brazil, September 2006), Volume II: Short Papers and Posters, pp. 18-20, 2006.

[17] Weyuker, E. J., Ostrand, T. J., and Bell, R. M. 2007. Using Developer Information as a Factor for Fault Prediction. In Proceedings of the Third international Workshop on Predictor Models in Software Engineering (May 20 - 26, 2007). International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 8.

[18] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslén, A. 2000. Experimentation in Software Engineering: an Introduction. Kluwer Academic Publishers.