Impact of Version History Metrics on Maintainability

Csaba Faragó, Péter Hegedűs, Gergely Ladányi, and Rudolf Ferenc

Department of Software Engineering,
University of Szeged, Hungary
{farago,hpeter,lgergely,ferenc}@inf.u-szeged.hu

Abstract. In this study we present how some version control history based metrics affect maintainability of the source code. These metrics cover intensity of modifications, code ownership and code aging. We determine the order of source files based on each analyzed metrics, and compare it with their maintainability based order. As a cross-check we perform a comparison test with post-release defects as well.

We performed the analysis on 14 versions of 4 well-known open source software systems. The results show high correlation between the version control metrics and relative maintainability indexes, in each case. The comparison with post-release defects also support the results in most of the cases.

Keywords: version control history metrics, source code maintainability, code churn, code ownership, code aging

1 Introduction

1.1 Motivation

Software maintenance consumes huge efforts: based on the experiences, high proportion of the total amount of software development costs are spent on this activity. As maintainability is in direct connection with maintenance costs, our motivation is to investigate the effect of the development process on the maintainability of the code. Our goal is to explore typical patterns causing similar changes in software maintainability, which could either help to avoid software erosion, or provide information about how to better allocate efforts spent on improving software maintainability.

1.2 Road Map

In previous works we already tackled this area of research. In paper [1] we presented that there is a strong connection between the version control operations and the maintainability of the source code. We also performed a study [2] that revealed the connection of the version control operations and maintainability.

It turned out that file additions have rather positive, file updates have rather negative effect on maintainability, while a clear effect of file deletions was not identified. In work [3] we presented the results of a variance analysis. File additions and file deletions increase the variance of the maintainability, and operation Update decreases it. In study [4] we checked how code ownership impacts maintainability, and concluded that common code is more likely to erode further than code with clear ownership. In article [5] we analyzed code churn, i.e. the intensity of past modifications, and found that modifying high-churn code is more likely to decrease the overall maintainability of a software system.

In the two most recent works [4,5] we argued that a possible practical application of these results is to create an IDE-plugin which determines and highlights the hotspots of source code in real-time, based solely on the history of the software in the version control system. In these studies our assumption was that source code with lack of ownership and high cumulative code churn values are already the hard-to-maintain parts of the software, which only erode further. However, the first part of the statement − i.e. the source code with the mentioned properties are already eroded − were not yet proved, which is necessary for next steps, and this study fills this gap. The current paper is a step towards the manifestation of this application: we consider another aspect of the impact of code ownership and code churn on maintainability, and extend the analysis on code aging.

In this paper we examine six metrics belonging to the following three types: code modification intensity, ownership and code aging related metrics, A hotspot detector tool which calculates the hotspots of a software based solely on version control history metrics is the following step on this road and is left for future work.

1.3 Comparison with Maintainability

To check the strength of the defined metrics on maintainability, we determined the order of the source files based on their relative maintainability indexes, considering concrete revisions of some software systems. The relative maintainability index (RMI) is the maintainability of the actual source file within the actually analyzed system. A positive number indicates that it belongs to the more maintainable source files. The RMI was calculated with the QualityGate SourceAudit tool [6] implemented based on the work of Bakota et al. [7]. We checked how these orders (i.e. the version control metrics based and RMI based orders of the source files) correlate using the Spearman's rank correlation test.

The methodology of RMI calculation are described in detail in sections 3.2 and 3.3. We stress that these are calculated solely using static source code analysis. On the other hand, neither of the version control history based metrics are directly connected to the content of the source code.

For a cross-check, we also calculated the correlation between the version control based metrics and the post-release bugs in the source files found in the PROMISE bug dataset [8].

1.4 Research Question

Formally, we investigated the following research questions in this paper:

RQ1: *How modifications intensity affects maintainability?*

RQ2: *How code ownership affects maintainability?*

RQ3: *How code aging affects maintainability?*

The null hypothesis of each question is that there is no correlation between these version control history based metrics and their maintainability, because the source of information are independent.

1.5 Overview of the Article

The remaining of the paper is organized as follows. Section 2 provides a brief overview of works that are related to this research. In Section 3 we present the methodology of how we collected the data and what kinds of tests we performed. In Section 4 we describe which versions of which software systems the tests were performed on, and present the results of the statistical tests. In Section 5 we list the possible threats to the validity of the results, while Section 6 answers the research questions formally and concludes the paper.

2 Related Work

This study is located between two major research fields: *maintainability analysis* and *mining software repositories.*

Several papers are dealing with how various object oriented metrics can be used for fault prediction. Brito and Melo [9] found that method hiding factor, method inheritance factor, attribute inheritance factor, polymorphism factor has negative, coupling factor has positive and attribute hiding factor has no correlation with software faults. Briand et al. [10] found that the coupling and inheritance measures are strongly related to the probability of fault, but they did not find significant impact of cohesion on fault proneness. Subramanyan and Krishnan [11] examined the connection between the number of defects and the following object-oriented metrics: methods per class, coupling between objects, depth of inheritance tree and number of children. In their study [12], Gyimóthy et al. found a strong positive correlation between the number of faults and the following metrics: number of methods per class, depth of inheritance tree, response for class, coupling between objects, lack of cohesion and number of logical lines of code. The quality model used by us (Bakota et al. [13]) rely on the listed findings.

Mining software repositories is large and evolving research area. In the annual conference of MSR [14] a great number of studies appear in connection with this research field. Kagdi et al. provide a taxonomy of articles in this area [15], based on the following aspects: software evolution, purpose, representation and information sources. Hassan [16] also provide a detailed overview about the MSR field.

3 Methodology

3.1 Version Control History Metrics

We calculated the orders of files based on the following version control history metrics (each metric defined an own order).

Modification intensity related metrics include cumulative code churn and number of modifications. *Cumulative code churn* is the absolute sum of number of added and removed lines of code so far. *Number of modifications* is the number of times the file in question has been modified so far.

Ownership related metrics include contributors and the contributors with tolerance. *Contributors* is the number of different contributors of the file so far. *Contributors with tolerance* is the number of different contributors of the file so far, but if someone contributed to the file only once, then that contribution is not considered.

Aging related metrics include age and last modification date. *Age* is date when the file was added. *Last modification date* is the date of the last modification.

Each of these metrics determines an order of files.

3.2 Measuring Maintainability on System Level

To calculate the maintainability values of the systems we used ColumbusQM, our probabilistic software quality model [13] that is able to measure the quality characteristics defined by the ISO/IEC 25010 standard. The model considers the following class related low-level quality properties: clone coverage, logical lines of code, rule violations, weighted methods per class, number of attributes, number of methods, comment density, commented lines of code, API documentation, lack of cohesion, number of ancestors and coupling between object. These metrics are compared with those of other systems in a benchmark, and then the results of the comparisons are aggregated using a probabilistic statistical algorithm utilizing also weights provided by experts.

The model was empirically validated, resulting that there is a correlation between the calculated maintainability values and the real development costs [17]. This was implemented primarily to analyze source code of Java programs.

3.3 Relative Maintainability Indexes

The above approach is used to obtain a system-level measure for source code maintainability. We extended the ColumbusQM with a drill-down approach [7], which provides the so called Relative Maintainability Indexes (the RMIs) for lower-level source code elements (e.g. classes or methods).

The RMI calculation works as follows: the maintainability analysis was performed on the whole system, and on the system without the analyzed source code element (which can be a package, a class or a function; in this study the

classes were considered). The RMI is difference between the original maintainability value and the maintainability value without that source code element. If the actual source code element is a hard to maintain code, compared to the rest of the system, then the maintainability value without that will be higher than the original one, therefore the RMI index of that element will be negative.

The RMIs were calculated for every class of the source code. The reason of this choice was the fact that in Java the source files and the classes are strongly correlated with each other: most of the source files contain exactly one class.

3.4 Number of Bugs

We considered bug data found in the PROMISE bug database [8], where the number of post release bugs of each source files of given release are made public.

3.5 Correlation Tests

We performed the Spearman's rank correlation check on every combination of version control history metrics on one hand, and RMIs and number of bugs on the other hand. This resulted $6 \cdot 2 = 12$ combinations of every analyzed versions. We performed the analysis using the R statistical software [18], using the `cor.test()` function.

4 Results

4.1 Results of the Statistical Tests

We selected the following open source software systems for analysis, all written in Java, developed in SVN version control systems and post release bug data available (analyzed versions in brackets): **Ant** (1.3, 1.4, 1.5, 1.6, 1.7), **JEdit** (4.0, 4.1, 4.2, 4.3), **Log4J** (1.0, 1.1, 1.2), **Xerces** (1.3, 1.4). Tables 1 and 2 contain the results of the correlation tests between the version control history metrics, and RMI and bug, respectively.

The rows of the tables contain the name of the analyzed software system, its version, and the results of correlation between RMI (or bug) based order, and the order of the following: cumulative code churn, number of modifications, ownership, ownership with tolerance, added date and last modified date.

The correlation test between cumulative code churn and RMI resulted a very strong negative result, meaning the higher the cumulative code churn of a file is (i.e. it has been more intensively modified in the past), it is more likely that the RMI is lower (i.e. its maintainability is worse). This result is also supported by the bug comparison as well, with positive correlation (higher cumulative code churn results in higher number of post-release bugs), with weaker correlation in absolute values.

The problem with cumulative code churn calculation is that it is very slow. On the other hand, the intensity of past modification can also be expressed by

Table 1. Spearman's Correlation ρs of RMI Comparison

Name	Version	Churn	Modifications	Ownership	Own.tolerance	Added	Modified
Ant	1.3	−0.861	−0.598	−0.392	−0.556	0.239	−0.563
	1.4	−0.867	−0.656	−0.475	−0.609	0.339	−0.373
	1.5	−0.747	−0.631	−0.550	−0.628	0.269	−0.592
	1.6	−0.852	−0.719	−0.636	−0.704	0.276	−0.464
	1.7	−0.702	−0.612	−0.560	−0.532	0.279	−0.268
jEdit	4.0	−0.712	−0.506	NA	−0.160	0.098	−0.442
	4.1	−0.681	−0.552	−0.515	−0.461	0.105	−0.466
	4.2	−0.713	−0.505	NA	−0.103	0.091	−0.478
	4.3	−0.302	−0.570	−0.488	−0.553	0.226	−0.044
Log4J	1.0	−0.823	−0.351	NA	−0.055	0.221	−0.283
	1.1	−0.873	−0.779	−0.556	−0.504	0.227	−0.535
	1.2	−0.854	−0.410	−0.481	−0.362	0.167	−0.102
Xerces	1.3	−0.660	−0.468	−0.217	−0.430	0.069	−0.100
	1.4	−0.481	−0.523	−0.322	−0.455	0.151	−0.355

Table 2. Spearman's Correlation ρs of Bug Comparison

Name	Version	Churn	Modifications	Ownership	Own.tolerance	Added	Modified
Ant	1.3	0.371	0.358	0.197	0.364	−0.142	0.398
	1.4	0.067	0.080	−0.029	0.108	0.138	0.326
	1.5	0.316	0.314	0.275	0.321	−0.176	0.231
	1.6	0.517	0.394	0.277	0.332	−0.174	0.393
	1.7	0.534	0.400	0.319	0.362	−0.172	0.236
jEdit	4.0	0.502	0.585	NA	0.063	−0.190	0.462
	4.1	0.546	0.653	0.539	0.536	−0.164	0.558
	4.2	0.405	0.501	NA	0.141	−0.177	0.345
	4.3	0.145	0.059	0.087	0.105	0.073	0.162
Log4J	1.0	0.589	0.388	NA	0.116	−0.304	0.301
	1.1	0.738	0.722	0.457	0.307	−0.204	0.531
	1.2	0.409	0.426	0.427	0.387	−0.294	−0.063
Xerces	1.3	0.365	0.357	0.203	0.270	−0.082	0.124
	1.4	0.255	0.408	0.090	0.484	−0.044	0.389

the mere number of past modifications. The connection is similar to cumulative code churn, with weaker correlation.

The correlation between ownership and RMI is similar to churn or modification comparison, with somewhat moderate results. In 3 cases the comparison was not applicable, as all the affected files had the same number of contributors. In one case of bug comparison there was a contradiction. We think this was casual, and it was resulted by the small number of post release bugs of that version and the small number of contributors.

The ownership with tolerance comparison resulted in more significant correlation. All the tests resulted a value, and there was no contradictory result.

In case of added date analysis the results indicate that the later added source files have better maintainability. That was a surprising result. Somewhat ironic explanation of this result can be the following: an early added source file had enough time to erode. However, the correlation in absolute value is weak, with two contradictions in case of bug comparison.

Finally, we found that the recently modified files are more likely to have worse maintainability, with higher number of bugs, and the correlations are significantly higher than those of file addition dates. In case of bug comparison cross check there was one contradicting result.

5   Threats to Validity

We performed the analysis on open source software systems exclusively, all implemented in Java. The lack of industrial systems threatens the general validity of our findings. Furthermore, the lack of the analysis covering other programming languages also threatens the generality. A possible direction of development is analysis of C# programs; the quality model can be based on the study of Hegedűs [19]. Anyway, we think that the results of the 14 analyzed version of 4 software systems are quite convincing.

The metrics calculation was performed on file basis, and the bug database is also file based. On the other hand, the RMI calculation was performed on class basis. Taking the common intersection lead to loss of the maintainability indexes of subclasses. This could lead some loss of precision in the results.

The bug database come from external source, and we do not know anything about the methodology of the data collection. Therefore it is not possible to analyze it deeper, and to find other connections.

The bug database contained data about a bit old version of software. We could not perform the general analysis (i.e. comparing with bug database as well) on more recent versions or on more recent projects.

6   Conclusions and Future Work

In this study we considered 6 version control history metrics, and performed correlation test between these on one hand, and relative maintainability index and number of post release bugs on the other hand. Based on the results we can answer research questions.

RQ1: *Higher intensity of file modifications result in worse maintainability and higher post release bugs.* The cumulative code churn turned out to result in the highest correlation values. The number of file modifications also resulted in high, but lower correlation values.

RQ2: *Source files of lacking clean code ownership result in worse maintainability and higher post release bugs.* The mere number of contributors so far has

a moderate strength of predicting maintainability and post release bugs, and it is somewhat higher if we apply a tolerance, not considering only one contributions.

RQ3: *Earlier added and later last modified files result in worse maintainability and higher post release bugs.* However, the strength of these correlations are the smallest, with a few contradictory results in case of bug comparison.

Next step is planned to be the normalization of the metrics and aggregating them for hotspot detection.

Acknowledgment

References

1. C. Faragó, P. Hegedűs, Á. Z. Végh, and R. Ferenc, "Connection between version control operations and quality change of the source code," *Acta Cybernetica*, vol. 21, pp. 585–607, 2014.
2. C. Faragó, P. Hegedűs, and R. Ferenc, "The impact of version control operations on the quality change of the source code," in *Computational Science and Its Applications (ICCSA)*. Springer, 2014, pp. 353–369.
3. C. Faragó, "Variance of source code quality change caused by version control operations," *Acta Cybernetica*, vol. 22, pp. 35–56, 2015.
4. C. Faragó, P. Hegedűs, and R. Ferenc, "Code ownership: Impact on maintainability," in *Computational Science and Its Applications–ICCSA 2015*. Springer, 2015, pp. 3–19.
5. ——, "Cumulative code churn: Impact on maintainability," in *15th IEEE International Working Conference on Source Code Analysis and Manipulation–SCAM 2015*, 2015.
6. T. Bakota, P. Hegedűs, I. Siket, G. Ladányi, and R. Ferenc, "QualityGate SourceAudit: a Tool for Assessing the Technical Quality of Software," in *Proceedings of the CSMR-WCRE 2014 Software Evolution Week (Merger of the 18th IEEE European Conference on Software Maintenance and Reengineering & 21st IEEE Working Conference on Reverse Engineering - CSMR-WCRE 2014)*. IEEE, Febr 2014, pp. 440–445.
7. P. Hegedűs, T. Bakota, G. Ladányi, C. Faragó, and R. Ferenc, "A drill-down approach for measuring maintainability at source code element level," *Electronic Communications of the EASST*, vol. 60, pp. 1–21, 2013.
8. "Promise defect database," http://openscience.us/repo/defect/.
9. F. Brito e Abreu and W. Melo, "Evaluating the impact of object-oriented design on software quality," in *Software Metrics Symposium, 1996., Proceedings of the 3rd International*. IEEE, 1996, pp. 90–99.
10. L. C. Briand, J. Wüst, J. W. Daly, and D. Victor Porter, "Exploring the relationships between design measures and software quality in object-oriented systems," *Journal of systems and software*, vol. 51, no. 3, pp. 245–273, 2000.

11. 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, pp. 297–310, 2003.

12. T. Gyimóthy, R. Ferenc, and I. Siket, "Empirical validation of object-oriented metrics on open source software for fault prediction," *IEEE Transactions on Software Engineering*, vol. 31, no. 10, pp. 897–910, 2005.

13. T. Bakota, P. Hegedűs, P. Körtvélyesi, R. Ferenc, and T. Gyimóthy, "A probabilistic software quality model," in *2011 27th IEEE International Conference on Software Maintenance*. IEEE, 2011, pp. 243–252.

14. "Mining Software Repositories (MSR) conference," `http://www.msrconf.org`.

15. H. Kagdi, M. L. Collard, and J. I. Maletic, "A survey and taxonomy of approaches for mining software repositories in the context of software evolution," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 19, no. 2, pp. 77–131, 2007.

16. A. E. Hassan, "The road ahead for mining software repositories," in *Frontiers of Software Maintenance, 2008. FoSM 2008*. IEEE, 2008, pp. 48–57.

17. T. Bakota, P. Hegedűs, G. Ladányi, P. Körtvélyesi, R. Ferenc, and T. Gyimóthy, "A cost model based on software maintainability," in *2012 28th IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 2012, pp. 316–325.

18. R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2015. [Online]. Available: http://www.R-project.org/

19. P. Hegedűs, "A probabilistic quality model for C# – an industrial case study," *Acta Cybernetica*, vol. 21, no. 1, pp. 135–147, 2013.